DOI: 10.1109/MPRV.2016.3

Software Support for Multi-Touch Interaction: a Survey from the Perspective of End-User Programming

Authors. **(1** Andrea Bellucci, abellucc@inf.uc3m.es (Universidad Carlos III de Madrid - UC3M, Madrid, Spain), corresponding author; **(2** Marco Romano, mromano@inf.uc3m.es (UC3M); **(3** Ignacio Aedo, aedo@ia.uc3m.es (UC3M) and;
(4 Paloma Díaz, pdp@inf.uc3m.es (UC3M)

[illegible]

There is a large body of research on EUP [1][3][4][5]. It provides tools and techniques to allow the end-users to write a program to create, modify or extend a software system. The concept of EUP has evolved from providing support to inexperienced users [4] to a perspective that focuses on users' goals [1]. An end-user, conversely from a professional developer, writes a program primarily for personal, rather than public use: the computer program becomes "a means to an end and only one of potentially many tools that could be used to accomplish a goal" [1].

EUP offers a variety of techniques to support users in the programming task [5], such as: **scripting languages,** which are fast to learn and commonly imply a lightweight syntax and semantics when compared to conventional programming languages, **Domain-Specific Languages (DSL ,** specialized programming languages that domain experts may better know and understand (e.g., LateX for typesetting or SQL for database queries); **Configurability, Customization** and **Parameterization,** or the possibility of configuring the parameters of basic elements or choose among different behaviors already available in the system (e.g., look and feel); **Programming by Example (PbE** or **Programming by Demonstration (PbD ,** in which "users provide example interactions and the system infers a routine from them" without requiring textual programming [5]; and, **Visual Programming,** which allows to program by manipulating graphic elements such as visual expressions, spatial arrangements of text and graphic symbols rather than by textual specification, thus providing a higher level of abstraction where no textual syntax is involved.

EUP focuses on the "creation" of programs, emphasizing speed and ease of development while neglecting aspects of the software lifecycle that are crucial for professional programmers, such as quality, robustness, re-use and maintainability of the code. For instance, while PbE may allow the straightforward implementation of complex touch behaviors inferred from sample interactions, the specification is not packaged as a reusable component and therefore it is difficult to incorporate in future programs.

End-user programmers for multi-touch are domain experts. Examples could be UX researchers or interaction designers that need to explore different multi-touch techniques in cross-device environments, or even curators of a cultural heritage institution or new media artists that want to include multi-touch sensing in interactive installations. In these cases, EUP takes the form of a prototyping activity, in which end-users implement many ideas by quickly and easily building functional test-bed applications, which can be dismissed later on. They therefore apply opportunistic practices for code generation that span from using high-level tools to copy-paste working examples from others.

Multi-Touch Interaction

Multi-touch is one of the dominant input modalities in pervasive computing. Nevertheless, the definitions of *multi-touch*, *gestures* and *widgets* are uncanny, due to their adoption both by the academia and the industry. In the rest of this section we provide the definitions we use in this survey.

[illegible]

[illegible]

- [illegible] '[illegible]'
- [illegible] "[illegible]"
- [illegible] '[illegible]'

[illegible]

[illegible]

- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]

[illegible]

[illegible]

- [illegible]

- **Multi-Touch Gestures**: high-level approaches that support the definition of gestures are needed for the end-user to overcome the complexity of programming touch behaviors. The software tool supports (1) the incorporation of common gestures via pre-defined gesture recognizers or (2) the creation of custom gestures by using, for instance, PbE ;
- **Gesture Widgets**: a change of the input device —direct touch instead of mouse pointer— implies a change of the user interface to achieve the so much advocated naturalness of multi-touch interaction. It is mandatory to overtake WIMP widgets and develop widgets designed for gestural manipulation. Again, building such interface elements can be tiresome and error-prone. Software tools in this category provide ready-to-use gesture-widgets together with the possibility to seamlessly develop custom components.

Technical Support

We consider three levels of *technical support*, or the extent users are supported while learning and using the system:

- **Documentation**: basic support is available, such as the Application Programming Interface (API) documentation;
- **Tutorials**: tutorials or help documentation are provided;
- **Community**: a community of users (e.g., forums) offer guidance on specific issues or share code ready for re-use.

Survey

Table I presents a survey of software tools for the end-user programming of multi-touch interaction according to the three dimensions. In Table II we provide technical details of the selected tools.

Table I. Survey of development tools for multi-touch interaction.
Table II. Technical details of selected development tools (last checked on February, 9th, 2015 .

The ***iOS***, ***Windows Phone*** and ***Android*** are Software Development Kits (SDK) that target the corresponding Operating System (OS) and provide an API to receive and handle input from multiple pointers on capacitive screens. All of them define an event-based architecture: *1)* the display screen receives an input from the user, *2)* the OS generates the event message that *3)* is sent to the application to update the user interface according to the message information.

Programming Abstraction: each SDK includes an Integrated Development Environment (IDE) supplying visual elements for the definition of widgets behavior, thus integrating textual programming with visual tools for graphic design. For instance, they offer a palette of graphic components and gesture widgets that can be dragged directly into the user interface. Users can then program the behavior of the graphic element by completing the automatically generated template through textual programming. Other features are provided, such as word completion, which reduces the cognitive burden of remembering long lists of APIs.

Touch interaction: the SDKs offer ready-to-use standard gestures to integrate in applications. It exists the possibility to define custom gestures, but only for

experienced programmer, since it requires to override the default gesture listeners.
Technical support: learning is facilitated by the large documentation covering API documentation and tutorials, the official supporting community and, by several unofficial communities that provide further documentation and tutorials.

ZOIL [18] has the broad scope of reducing the complexity of building interactive spaces augmented with pervasive computing technologies, including multi-touch, tangible and motion sensing devices.
Programming Abstraction: applications are developed through classic textual programming or by using a XML-based syntax. End-users can therefore define the visual appearance of interface components and their touch behavior with a declarative approach, that lowers the threshold of using procedural languages.
Touch interaction: ZOIL spans from high-level common gestures to low-level touch input handling. It also provides users with an extensible library of off-the-shelf behaviors that can be easily assigned to objects and customized according to users needs.
Technical support: ZOIL targets experienced users. Even if the website provides examples and tutorials, previous experience with the Windows Presentation Framework is needed to start building applications.

Kivy and ***GestureWorks*** are cross-platform, device-independent tools that have been developed from scratch following guidelines for multi-touch post-WIMP interaction, thus dismissing the legacy of old models from the desktop paradigm. Both of them are versatile and flexible: with few lines of code it is possible to quickly set up, connect and arrange running applications. *GestureWorks* license is not free, but a parallel open-source project ***Open Exhibits*** is available, which focuses on the development of digitally-augmented interactive museum exhibits.
Programming Abstraction: *Kivy* allows the definition of interface components through an XML-based DSL which behavior can be programmed in Python. *GestureWorks* goes one step further, providing the *Gesture Markup Language GML)* to define touch events and API wrappers for a wide range of programming languages.
Touch interaction: *GestureWorks* provides a large set of pre-built touch gestures and it supports the definition of custom gestures and gesture sequences through GML. *Open Exhibits* —its open-source ActionScript3 implementation— introduces the *Creative Markup Language CML)* DSL for the definition of gesture-widgets. CML is used in combination with GML and Cascade Style-Sheet (CSS) to define, interface modules, their behavior and the look-and-feel. A Runtime Player parses the description files and automatically builds and runs the resulting application. The same process can be done via source code, either importing CML, GML and CSS files or from scratch. In *Kivy*, the creation of standard and custom multi-touch gestures is allowed only via source code. *Kivy* provides just a basic set of gesture-widgets compared with *GestureWorks/Open Exhibits*.
Technical support: both tools are easy to use and easy to learn thanks to the documentation available by the official websites, such as full API documentation and complete tutorials. In addition, they are backed up by an active official community.

[illegible]

[illegible]

[illegible]

Being a software framework that builds upon a complex API such as Windows Presentation Framework, C# and XAML, neither *ZOIL* can be considered a low threshold toolkit for rapid prototyping. Authors report that use cases required 4 or 5 weeks to be developed [18]. However, its high ceiling could be worth the hassle since, conversely from the previous tools, it inscribes multi-touch interaction in the wider scenario of interactive spaces, supporting multi-display interaction, distributed user interfaces designed for touch manipulation and visualization tools for creating alternative visual representations.

Multi-touch tools for gesture prototyping fall into two families, according to the EUP technique they adopt: end-users can either choose PbE (e.g., - -*family* or *Touch*) or DSL-based declarative approaches (e.g., *Proton/Proton++*, *JMidas*, *GeForMTjs*). Both approaches are flexible and extensible, allowing end-user developers to create custom gestures without handling raw input data. Moreover, they create platform-independent descriptions of touch interaction, thus empowering EUP tools to keep pace with the incessant availability of new input devices (e.g., Microsoft Kinect, Leap Motion) and avoid them to fall into disuse. Defining gestures from examples has the advantage that end-users do not need to learn a new modeling language and it is easy to design gestures with complex trajectories. The main drawback is that current PbE approaches rely on 2D spatial pattern-matching and, therefore, do not take advantage of rich gestural information, such as temporal relationships (e.g., speed), finger orientation or touch area, which are, conversely, supported by formalisms. Moreover, high-level declaration languages support modularization and composition of simple gestures into more articulated ones. Nevertheless, they often lead to complex specifications that are difficult to maintain. In addition, these specification languages lack support for modeling motions that involve geometric transformations, which are often unintuitive and difficult to describe manually. To ease authoring gestures, *Proton/Proton++* introduces a graphical interface for the definition of gestures as regular expressions: the visual tool demonstrated to be faster for generating basic gestures and easier to understand for trajectory gestures [13].

In some cases the choice of a tool is guided by the application context instead of the EUD technique and a thoroughly inspection of features is required by the end-user. For instance, multi-touch applications for large surfaces such as interactive tabletops require spatial invariance of gestures, because users need to perform gestures from any position around the table. For mobile device applications, on the contrary, this feature is not relevant. JMidas and *Proton/Proton++* do not support orientation invariance, which is provided by - -*family* algorithms.

- -*family* deserves special attention. Even if each version presents its peculiar limitations, they are lightweight, fast and accurate algorithms, and represent a good solution for rapid prototyping, where the quick exploration of a design space is a priority. $-family algorithms make it easy to modify the gestures set and they can be easily integrated into other tools by tailoring the source code to specific needs, as demonstrated by *GeForMTjs*.

[illegible]

[illegible]

[illegible]

[illegible]

[5] [illegible]

[illegible]

[illegible]

[14] [illegible]

[15] [illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]