Software requirements for a (more) manageable multi-touch ecosystem

Paolo Olivo[1,3], Damien Marchal[2,3] & Nicolas Roussel[1,3]
[1]INRIA, [2]CNRS, [3]MINT (Univ. Lille 1, CNRS, INRIA)
IRCICA, Parc Scientifique de la Haute Borne, Villeneuve d'Ascq, France
paolo.olivo@inria.fr, damien.marchal@lifl.fr, nicolas.roussel@inria.fr

ABSTRACT
Having designed, implemented and evaluated multi-touch techniques, applications and systems, members of our research team have had a number of needs that were not necessarily met by current tools. These tools notably lack proper support for standard but extensible device and event descriptions, test-driven development and flexible networking configurations, for example. In this paper, we detail and explain these different needs, discuss potential solutions and present the first steps we have taken towards a new multi-touch toolkit to address them.

Author Keywords
Multi-touch, development, testing, event descriptions, service discovery, control protocol, data transfer.

INTRODUCTION
Our research team focuses on *gestural interaction*, movements that a computing system can sense and respond to. A gesture in this context can be seen as a function of time into a set of sensed dimensions that might include but are not limited to positional information. Simple gestures have long been supported by interactive graphics systems and the advent of robust and affordable sensing technologies has somewhat broadened their use. Yet the expressive power of current gestures remains limited while the increasing diversity and complexity of computer-supported activities calls for more powerful interactions.

Our general goal is to foster the emergence of these new interactions, to broaden the use of gestures by supporting more complex operations. We are developing the scientific and technical foundations required to facilitate the design, implementation and evaluation of these interactions. Our interests include gestures captured using held, worn and touched objects or contactless technologies; computational representations of these gestures; methods for characterizing and recognizing them; transfer functions for non-isometric manipulations; feedback mechanisms, and more particularly haptic ones; engineering tools to facilitate the implementation of gestural interaction techniques; and evaluation methods to assess their usability.

To be presented at the EICS 2011 Workshop on Engineering Patterns for Multi-Touch Interfaces.

Within the field of multi-touch interaction, our team has particularly focused on the design, implementation and evaluation of innovative techniques for 3D environments. Recent research include the design of a 3D positioning technique [6] and a study of the effect of DOF separation in 3D manipulation tasks [7], for example. Our team collaborates with an SME on the design of specific navigation and manipulation techniques for commercial 3D applications (e.g. virtual shopping, room planning). We have also worked on the integration of multi-touch support in X3D, an ISO ratified standard for 3D scenes in XML, and Blender, a popular open source 3D content creation suite.

Members of our team have developed multi-touch software on Microsoft Windows, Ubuntu Linux and Apple OS X using different programming languages, (e.g. C/C++, Java, JavaScript, Python) and toolkits (e.g. Ogre+PhysX, BS Contact, PyMT). These software have been used on a variety of multi-touch devices, from smart phones and tablets to screens and tables (both commercial and custom-made). The diversity of software and hardware platforms combined with the specific needs of our research and development activities have been the source of recurring frustrations with existing development tools.

In this paper, we describe the various needs we have had that were not necessarily met by current tools. We then discuss different solutions that could help alleviate these problems. We finally describe the first steps we have taken in this direction.

WHAT IS NEEDED
Solutions exist on most modern operating systems to communicate with multi-touch devices and route their events to applications. While some systems natively support a limited set of specific devices, efforts have been made on some others (e.g. Windows 7 or Linux with mtdev) to transparently support a greater number of them, from multiple vendors. Yet multi-touch devices can greatly vary in their characteristics, even in their most basic ones, e.g. their latency, the number of simultaneous contacts they support, the range and resolution of their different measures, and their sampling frequency. Among others, all these *device characteristics* need

to be programmatically accessible in relevant standard units (e.g. $meter$, $kilogram$, $second$, $newton$, $hertz$). Access to this information is required to properly compare devices, to understand the effect of a particular characteristic or combination, and to adapt the applications accordingly.

Although high-level general-purpose frameworks such as Windows Presentation Foundation, Cocoa, Android SDK or Qt are now multi-touch aware, they expose the corresponding events in very generic ways[1]. In some cases, the information available for a contact can be limited to a simple $2D$ position. As a consequence, again, specific information that could be provided by a particular device or low-level library is often unavailable to developers and applications. As one targets multiple software platforms, one is usually constrained by the lowest common denominator, further reducing the available information. In some cases, the missing low-level information can be recomputed by the application, e.g. the speed of a moving contact. In some others, it is unfortunately impossible, e.g. the orientation of a contact or the raw camera image from which it was detected.

We believe there is a need for *extensible multi-touch event descriptions*. Although it is sometimes useful to abstract them, we need to preserve their specificities all the way from the device to the application. Events should not be described by fixed-size records but rather dictionary-like structures (or a combination of the two) to support heterogeneity and lazy, incremental extension. It should be possible to compute the low-pass filtered position of a contact or its moving speed and add this information to the event if not already present, for example. Or to tag all present contacts with cluster IDs. Such processing should be implemented in libraries that are not tied to a particular device, system or framework. We need independent multi-touch event processing libraries, just like we have media processing ones that make no assumption on the origin or final destination of the processed data.

The final destination of multi-touch events is generally a process that performs a high-level interpretation of the corresponding gestures in terms of user manipulations and commands. In order to tune and evaluate these processes, we need to be able to replicate and reproduce multi-touch input, reproducibility requiring changes while replicability avoids them [1]. We thus need ways to record events, to replay them identically or in altered ways (e.g. with added noise), and possibly also to synthesize them. *Synthesizers*, *recorders* and *players* support a test-driven development approach of high-level interpretation processes that can be combined with other methods such as fuzz or unit testing. To further facilitate testing and evaluation, applications need to be able to access devices without being physically connected to them. We thus also need ways to transmit multi-touch events over computer networks. Figure 1 summarizes the overall envisioned life-cycle of these events.

To facilitate the development and deployment of networked multi-touch applications, we must move beyond hard-wired and hand-specified IP addresses and port numbers. Dynamic

[1]frameworks more specifically targeted at multi-touch interfaces will be discussed in the next section

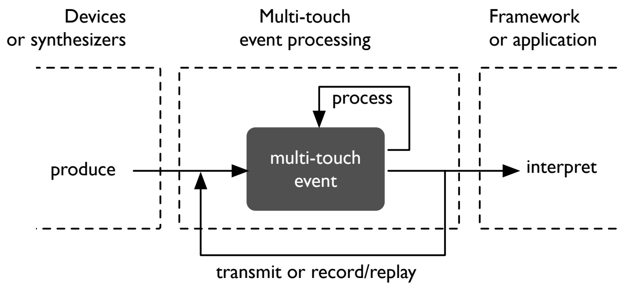

Figure 1. Lifecycle of multi-touch events.

configurations are needed, and developers and users should not care about their details. A zero configuration approach is needed, instead: one should be able to assemble the different entities involved in a multi-touch application at run-time, with no particular knowledge of the underlying network(s). This requires *service publishing and discovery protocols* so that the entities are aware of each other, but also appropriate presentation and interaction techniques so that users can easily understand the possibilities and create the desired links.

Once a link is established between two multi-touch entities, they need some protocol to understand their respective capabilities and negotiate the parameters of future data exchanges. This *control protocol* should make it possible to query the aforementioned characteristics of a device and then request that only certain dimensions be transmitted at a certain rate and in a certain format. The receiver should be able to request values in standard (e.g. in m) or normalized units (e.g. between 0.0 and 1.0 based on the device dimensions), for example. It might request some basic pre-processing on the data, e.g. low-pass filtering. Using the control protocol, the two parties should finally agree on an out-of-band channel and a *data format* to be used for multi-touch events. While the control channel should always support reliable and ordered delivery of messages, receivers might have different preferences for the data one. A recorder might still want reliable and ordered event delivery, while other applications might prefer reduced latency for example.

The above use of the term *entity* is important. The terms *server* and *client* are confusing because their definition usually depends on whether one sees the value of the service in data production or use[2]. A networked multi-touch entity should be able to receive, process and transmit events at the same time. One might receive events from a device, annotate them and forward them to a recorder for centralized storage, for example. It should also be possible to connect an entity to arbitrary numbers of event sources and destinations. Connecting one source to multiple destinations notably supports the use of external debugging tools during application development and the synchronized comparative evaluation of interpretation processes. Connecting multiple sources to the same destination is a requirement for multi-device and multi-

[2]For example, although TUIO is based on OSC, the two specifications use opposite conventions: TUIO defines a server as an entity that sends messages, while OSC defines it as one that receives them. TUIO emphasizes the production of OSC packets while OSC emphasizes their use to produce something else (e.g. sounds).

user applications. In these scenarios, extensible descriptions can help avoiding name or ID collisions by tagging all events with their respective source.

In addition to those we already mentioned, one can easily think of other entity classes. *Tunnels* could act as blind relays between entities on otherwise separate networks, for example. *Gateways* and *proxies* could be used to create wrappers around existing applications, the former simply passing data and the latter having the possibility to process it on the way. As we combine entities in more and more sophisticated ways, we will also need ways to precisely evaluate the impact of the networked infrastructure, e.g. to measure lags, jitters and packet losses.

Our general desire is to move from relatively static and ad-hoc combinations of applications towards a more dynamic and flexible multi-touch ecosystem. Doing so, we would like to reduce the number of constrained choices to a minimum. We are particularly reluctant to impose a particular user interface toolkit or programming language to developers: experience shows that not only has everyone his/her own preferences and constraints, but some combinations are also clearly more appropriate than others in some contexts.

WHAT WE HAVE

A number of specific frameworks have been proposed to facilitate the development of multi-touch applications by abstracting input devices [4]. Although each of these frameworks meets some of our requirements, none of them really meets them all. They expose little information about the devices they support, for example, and actually tend to hide their differences by using normalized coordinates. Using device-dependent measures instead of standard units makes it impossible to compute meaningful velocities from positions when the dimensions of the device are unknown. The event descriptions used by these frameworks can hardly be extended by applications anyway, except in the case of PyMT [3] by virtue of the Python language. Except for the TouchToolkit [5] that was specifically designed for that purpose, the support for test-driven development is usually limited. Network configuration support also lacks the flexibility we desire. And to the notable exception of Sparsh-UI [8] and libTISCH [2], most of the existing frameworks target a single programming language. Some go as far as imposing their own widgets and event loop, which makes them incompatible with general-purpose user interface libraries.

Several solutions exist to discover services on a local area network. Based on mDNS[3] and DNS-SD[4], Apple's *Bonjour* is probably one of the easiest to use. Its mDNSResponder compiles and runs on all major operating systems. It then only takes a few lines of C code to publish a service by naming it and specifying its type and the corresponding host name and TCP or UDP port number. A DNS TXT record can be used to provide a small extra amount of structured information about the service. A few lines of code also allow to create and maintain a list of all available instances of a particular type of service and be notified of TXT records updates. The fact that networked entities can become aware of each others does not necessarily mean that users will be able to distinguish them in a list and figure out how to connect them. Yet simple interaction techniques exist that can help associate devices by detecting similar and synchronous actions on them for example [9].

Bonjour supports wide-area service discovery, but configuring one's system for doing so is not easy. XMPP[5] is a valuable alternative for reaching out to other networks. This protocol offers decentralized, reliable, secured, flexible and extensible support for both "presence" (information about network availability) and real-time XML-based communication. Although server-based in the general case, it supports local serverless presence and messaging using mDNS and DNS-SD. It also supports the initiation and management of out-of-band real-time communication channels using a wide variety of transport methods.

XMPP meets both the service discovery and control protocol requirements described in the previous section. But other protocols and formats could be used for the control channel. One could use an XML-over-HTTP protocol such as SOAP[6] or JSON-RPC[7], for example. Or follow a REST[8] approach to better leverage HTTP's capabilities. One could of course also implement his/her own custom control protocol over any reliable transport. It should be noted that reliability has a cost in terms of packet overhead that protocol designers have often traded for latency. When using TCP, this trade-off can be adjusted by setting a `NO_DELAY` option to disable the default congestion control algorithm.

Although multi-touch events can perfectly be streamed over a TCP connection, e.g. using SLIP[9] framing, the common practice is to send them as UDP datagrams. UDP is presumably faster than TCP and supports multicast. But it is connectionless and unreliable: datagrams can get lost or arrive in a wrong order. As a consequence, the communicating parties can have a difficult time maintaining an accurate idea of the remote state. When transmitting multi-touch events as UDP datagrams, one must at least check the receiving order, e.g. using timestamps. In case the event holds information that depends on previous events, one should also make sure that packet losses can be detected, e.g. by numbering datagrams instead or in addition to using timestamps. The use of a reliable control channel in addition to a UDP data channel allows both parties to periodically send important data that might get lost otherwise, and to cleanly put an end to the communication.

The common practice to serialize multi-touch events is to encode them in OSC[10] packets conforming to a TUIO[11] specification. Considering the comprehensive nature of the coming 2.0 specification, it might take some time before it re-

[3]http://www.multicastdns.org/
[4]http://www.dns-sd.org/
[5]http://xmpp.org/
[6]http://www.w3.org/TR/soap/
[7]http://json-rpc.org/
[8]http://en.wikipedia.org/wiki/REST
[9]https://datatracker.ietf.org/doc/rfc1055/
[10]http://opensoundcontrol.org/
[11]http://www.tuio.org/

places the current 1.x ones that are widely used. The 2.0 version seems however preferable for new developments as it uses a more compact message syntax and leaves room for extensions. Note however that although TUIO messages are predominantly sent as UDP datagrams with an OSC payload, one could send them in any other format (e.g. XML or JSON) over any other transport (e.g. TCP).

WHAT WE HAVE DONE

We have started developing a toolkit based on the above standards in an attempt to meet the requirements identified at the beginning of the paper. With our current implementation, an entity can create a *service provider* to advertise a TCP-based service. Other entities will discover it using a *service browser* (Figure 2, arrow 1). They will then be able to create a *service connector* using the discovered hostname and port number, which will initiate a TCP control connection with the provider (2). The provider will wrap the TCP connection in a service connector. The two connectors will then negotiate a separate data channel (3).

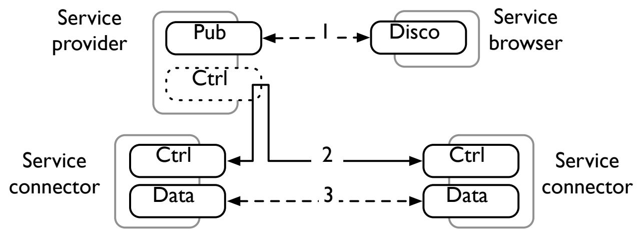

Figure 2. Service-oriented architecture.

We have implemented two discovery protocols (DNS-SD and XMPP), three custom control protocols (XMPP, SOAP and OSC-based) and two OSC-based data protocols (Figure 3). At present, the control channel is used to retrieve informational key/value pairs from the provider (e.g. the dimensions of an interactive surface), to negotiate the data protocol and support clean termination of both communications. We plan to use it in the near future to support content negotiation and flow control at any time.

	Pub / Disco		Ctrl			Data	
Format		XML	XML	XML	OSC	OSC	OSC
Protocols	DNS-SD mDNS	XMPP HTTP	XMPP HTTP	SOAP HTTP	SLIP		SLIP
Transport	UDP	TCP	TCP	TCP	TCP	UDP	TCP

Figure 3. Discovery, control and data protocols implemented.

We have implemented TUIO 1.1 service providers for several multi-touch devices. We have also implemented a Qt-based application with a service browser that allows to visualize the gestures received from one or more TUIO providers. The Qt service browser is used by a number of other applications that record OSC streams, replay hierarchical playlists at a variable speed, or visualize statistical information about the streams. A *bridge* application allows to wrap any existing OSC/UDP application as a service provider. A *multiplexer* serves the opposite purpose, i.e. it allows to select one or more OSC providers and to use them as a "standard" OSC/UDP source.

All our current code is written in Python and runs on Ubuntu Linux and Mac OS X. Although we use PyQt in certain applications, care has been taken so that the eventing and networking code remains as independent as possible from general-purpose frameworks. The bridge and the multiplexer allow our applications to communicate with others written in different languages, with other frameworks and on other platforms. The design choices we made should be easily ported to another language.

CONCLUSION

We have described various needs that we have had while implementing and evaluating multi-touch interactions. We have briefly discussed a number of potential solutions and presented the first steps we have taken towards a new multi-touch toolkit. Future work on that toolkit will focus on stabilizing the various network protocols and adding event filtering (low-level) and interpreting (high-level) processes.

REFERENCES

1. C. Drummond. Replicability is not reproducibility: nor is it good science. In *Proceedings of the 26th International Conference on Machine Learning: Workshop on Evaluation Methods for Machine Learning IV*, 2009. 4 pages.

2. F. Echtler. *Tangible Information Displays*. PhD thesis, Technische Universität München, Nov. 2009.

3. T. E. Hansen, J. P. Hourcade, M. Virbel, S. Patali, and T. Serra. PyMT: a post-WIMP multi-touch user interface toolkit. In *Proceedings of ITS '09*, 17–24. ACM, 2009.

4. D. Kammer, M. Keck, G. Freitag, and M. Wacker. Taxonomy and overview of multi-touch frameworks: Architecture, scope and features. In *Proceedings of the EICS '10 workshop on Engineering Patterns for Multi-Touch Interfaces*, 2010.

5. S. H. Khandkar, S. M. Sohan, J. Sillito, and F. Maurer. Tool support for testing complex multi-touch gestures. In *Proceedings of ITS '10*, 59–68. ACM, 2010.

6. A. Martinet, G. Casiez, and L. Grisoni. 3D positioning techniques for multi-touch displays. In *Proceedings of VRST '09*, 227–228. ACM, 2009.

7. A. Martinet, G. Casiez, and L. Grisoni. The effect of DOF separation in 3D manipulation tasks with multi-touch displays. In *Proceedings of VRST '10*, 111–118. ACM, 2010.

8. P. Ramanahally, S. Gilbert, T. Niedzielski, D. Velázquez, and C. Anagnost. Sparsh UI: A multi-touch framework for collaboration and modular gesture recognition. In *Proceedings of WINVR '09*, 137–142. ASME, 2009.

9. J. Rekimoto. SyncTap: synchronous user operation for spontaneous network connection. *Personal Ubiquitous Computing*, 8:126–134, May 2004.

ABOUT THE AUTHORS

The three authors are members of MINT, a joint research team between INRIA Lille and the LIFL (UMR CNRS 8022) and L2EP laboratories of Lille 1 University.

Paolo Olivo (MSc, Milano Bicocca, 2009) is a contract engineer at INRIA Lille. His interests include software engineering and development for tactile and gestural interaction. The present focus of his work is on programming tools for supporting research in Human-Computer Interaction.

Damien Marchal (PhD, Lille, 2006) is a research engineer at CNRS. His current activities include the development of multi-touch frameworks and interaction techniques for 3D content. He is the author of extensions to support multi-touch in X3D and Blender. His other interests include shape modelling and geometric representations, field interpolation and logic based semantic reasoning.

Nicolas. Roussel (PhD/Hab, Paris-Sud, 2000/2007) is a senior researcher at INRIA Lille. His research lies in the field of Human-Computer Interaction with current primary interests in the design, implementation and evaluation of simple yet powerful tactile and gestural interactions. Other interests include engineering of interactive systems, window systems, computer-mediated communication and groupware. He has published and served as program committee member in conferences such as ACM CHI, ACM UIST, ACM CSCW, and ACM Multimedia.