TUIO: A Protocol for Table-Top Tangible User Interfaces

Martin Kaltenbrunner[1], Till Bovermann[2], Ross Bencina[1] and Enrico Costanza[3]

[1] Music Technology Group, IUA, Universitat Pompeu Fabra, Barcelona, Spain.
[2] Neuroinformatics Group, Faculty of Technology, Bielefeld University, Germany.
[3] Liminal Devices Group, Medialab Europe, Dublin, Ireland.

Abstract. In this article we present TUIO, a simple yet versatile protocol designed specifically to meet the requirements of table-top tangible user interfaces. Inspired by the idea of interconnecting various existing table interfaces such as the reacTable* [1], being developed in Barcelona and the tDesk [2] from Bielefeld, this protocol defines common properties of controller objects on the table surface as well as of finger and hand gestures performed by the user. Currently this protocol has been implemented within a fiducial marker-based computer vision engine developed for the reacTable* project. This fast and robust computer vision engine is based on the original d-touch concept [3], which is also included as an alternative to the newer fiducial tracking engine. The computer vision framework has been implemented on various standard platforms and can be extended with additional sensor components. We are currently working on the tracking of finger-tips for gestural control within the table interface. The TUIO protocol has been implemented using OpenSound Control [4] and is therefore usable on any platform supporting this protocol. At the moment we have working implementations for Java, C++, PureData, Max/MSP, SuperCollider and Flash.

1 General Observations

This protocol definition is an attempt to provide a general and versatile communication interface between tangible table-top controller interfaces and underlying application layers. It was designed to meet the needs of table-top interactive surfaces, where the user is able to manipulate a set of objects. These objects are tracked by a sensor system and can be identified and located in position and orientation on the table surface. Additionally we defined a special cursor object, which doesn't have a unique ID and doesn't provide rotation information.

The protocol's flexible design offers methods for selecting which information will be sent. This flexibility is provided without affecting existing interfaces, or requiring re-implementation to maintain compatibility.

2 Implementation Details

The TUIO protocol defines two main classes of messages: set messages and *alive* messages. *Set* messages are used to communicate information about an object's state such as position, orientation, and other recognized states. *Alive* messages indicate the current set of objects present on the surface using a list of unique session IDs.

To avoid possible errors evolving out of packet loss, no explicit add or remove messages are included in the TUIO-protocol. The receiver deduces object lifetimes by examining the difference between sequential alive messages.

In addition to *set* and *alive* messages, *fseq* messages are defined to uniquely tag each update step with a unique frame sequence ID. To summarize:

- object parameters are sent after state change using a *set* message
- on object removal an *alive* message is sent
- the client deduces object addition and removal from *set* and *alive* messages
- *fseq* messages associate a unique frame id with a set of *set* and *alive* messages

2.1 Efficiency & Reliability

In order to provide low latency communication our implementation of the TUIO protocol uses UDP transport. When using UDP the possibility exists that some packets will be lost. Therefore, our implementation of the TUIO protocol includes redundant information to correct possible lost packets, while maintaining an efficient usage of the channel. An alternative TCP connection would assure the secure transport but at the cost of higher latency.

For efficiency reasons *set* messages are packed into a bundle to completely use the space provided by a UDP packet. Each bundle also includes a redundant *alive* message to allow for the possibility of packet loss. For larger object sets a series of packets, each including an *alive* message are transmitted. When the surface is quiescent, emphalive messages are sent at a fixed rate dependent on the channel quality, for example once every second, to ensure that the receiver eventually acquires a consistent view of the set of alive objects.

The state of each alive but unchanged object is periodically resent with additional *set* messages. This redundant information is resent at a lower rate, and includes only a subset of the unchanged objects at each update. The subset is continuously cycled so that each object is periodically addressed.

Finally, each packet is marked with a frame sequence ID (*fseq*) message: an increasing number which is the same for all packets containing data acquired at the same time. This allows the client to maintain consistency by identifying and dropping out-of-order packets . To summarize:

- *set* messages are bundled to fully utilize UDP packets
- each bundle of *set* messages includes an *alive* message containing the session IDs of all currently alive tangible objects

– when the surface is quiescent the *alive* message is resent periodically
– the state of a cycling subset of alive but unchanged objects is continuously resent via redundant *set* messages
– each bundle contains a frame sequence (*fseq*) message

It should be noted that the retransmission semantics described here are only one possible interpretation of the protocol. Other possible methods include: (1) weighting the frequency of retransmission according to recency of value changes using a logarithmic back-off scheme and, (2) trimming the set of values to be retransmitted using asynchronous acknowledgments from the client.

2.2 Message Format

Since TUIO is implemented using Open Sound Control (OSC) [4] it follows its general syntax. An implementation therefore has to use an appropriate OSC library such as [5] and has to listen to the following message types:

```
/tuio/[profileName] set sessionID [parameterList]

/tuio/[profileName] alive [list of active sessionIDs]

/tuio/[profileName] fseq int32
```

2.3 Parameters

The parameters defined in this section reflect the object properties we considered important for an interactive surface interface. Some of these parameters (id, position and angle) are retrieved directly by the sensor. Others (speed, acceleration) are derived from these primary parameters using timing information. Computing these parameters on the low level side of an tangible user interface system allows a more efficient computation, since the necessary timing information does not need to be transferred to clients.

`s`	sessionID, temporary object ID, int32
`i`	classID, fiducial ID number, int32
`x, y, z`	position, float32, range 0...1
`a, b, c`	angle, float32, range 0..2PI
`X, Y ,Z`	movement vector (motion speed & direction), float32
`A, B, C`	rotation vector (rotation speed & direction), float32
`m`	motion acceleration, float32
`r`	rotation acceleration, float32
`P`	free parameter, type defined by OSC packet header

Table 1. semantic types of *set* messages

A session ID number is assigned to each object. This is necessary to uniquely identify untagged objects across successive frames, and in the case where multiple objects tagged with the same classID are simultaneously present on the surface. The semantic types allowed in a *set* message are shown in Tab.1.

2.4 Profiles

We define a set of profiles, which apply to most table-style tangible user interfaces. This allows the tracking of objects and cursors on two dimensional surfaces and in special cases also in the 3D space above the table surface. If one of these predefined profiles doesn't meet a system's requirements we also allow so-called raw profiles that send the raw sensor data, as well as free form profiles, which allow a user defined set of parameters to be transmitted.

2D Interactive Surface

```
/tuio/2Dobj set s i x y a X Y A m r
/tuio/2Dcur set s x y m r
```

2.5D Interactive Surface

```
/tuio/25Dobj set s i x y z a X Y A m r
/tuio/25Dcur set s x y z m r
```

3D Interactive Surfaces

```
/tuio/3Dobj set s i x y z a X Y Z A m r
/tuio/3Dcur set s x y z m r
```

raw profile

```
/tuio/raw_[profileName]
/tuio/raw_dtouch set i x y a
```

custom profile

```
/tuio/_[formatString]
/tuio/_sixyP set s i x y 0.57
```

For the last two profiles the parameters of the *set* message are in a user defined format. Raw profiles correspond to a specific sensor type, such as d-touch, and carry its standard parameters. The completely free-form profile carries its format within its name, similar to the OSC header.

3 Conclusion

A protocol called TUIO was presented which supports communication of all required information between the object recognition layer and interaction layer of a tangible user interface system. This protocol supports communication between several totally different tangible user interfaces including the REACTABLE* and the TDESK. It thus facilitates interaction between people at different locations, including the possibility of distributed musical performance.

4 Acknowledgments

This work has been partially supported by the European Commission Cost-287 ConGAS action on Gesture Controlled Audio Systems and will be released under an open source license.

References

1. Kaltenbrunner, M., Geiger, G., Jorda, S.: "Dynamic Patches for Live Musical Performance". Proceedings of the 4th Conference on New Interfaces for Musical Expression (NIME 04), Hamamatsu (Japan)
2. Hermann, T., Henning, T., Ritter, H.: "Gesture Desk - An Integrated Multi-Modal Workplace for Interactive Sonification". Int. Gesture Workshop 2003, Genova, Italy.
3. Costanza, E., Shelley, S. B., Robinson, J.: "D-touch: A Consumer-Grade Tangible Interface Module and Musical Applications". Proceedings of Conference on Human-Computer Interaction (HCI03), Bath, UK, 2003.
4. Wright, M., Freed, A., Momeni A.: "OpenSound Control: State of the Art 2003". Proceedings of the 3rd Conference on New Instruments for Musical Expression (NIME 03), Montreal, Canada, 2003.
5. Bencina, R.: oscpack `http://www.audiomulch.com/~rossb/code/oscpack/`