# The TUIO 2.0 Protocol: An Abstraction Framework for Tangible Interactive Surfaces

MARTIN KALTENBRUNNER, Kunstuniversität Linz, Austria
FLORIAN ECHTLER, Bauhaus-Universität Weimar, Germany

Since its introduction in 2005, the TUIO protocol has been widely employed within a multitude of usage contexts in tangible and multi-touch interaction. While its simple and versatile design still covers the core functionality of interactive tabletop systems, the conceptual and technical developments of the past decade also led to a variety of ad-hoc extensions and modifications for specific scenarios. In this paper, we present an analysis of the strengths and shortcomings of TUIO 1.1, leading to the constitution of an extended abstraction model for tangible interactive surfaces and the specification of the second-generation TUIO 2.0 protocol, along with several example encodings of existing tangible interaction concepts.

CCS Concepts: • **Networks** → Application layer protocols; • **Human-centered computing** → *HCI theory, concepts and models*; **User interface toolkits**; • **Software and its engineering** → *Graphical user interface languages*;

Additional Key Words and Phrases: TUIO; tangible; touch; surface computing; input processing; software architecture

**ACM Reference Format:**
Martin Kaltenbrunner and Florian Echtler. 2018. The TUIO 2.0 Protocol: An Abstraction Framework for Tangible Interactive Surfaces. *Proc. ACM Hum.-Comput. Interact.* 2, EICS, Article 8 (June 2018), 34 pages. https://doi.org/10.1145/3229090

## 1 INTRODUCTION

Interaction with touch and tangible objects on large-scale displays has been a focus of research since the early 2000s, initiated through several seminal publications by Ishii and Ulmer [25, 26]. These interactive systems have often been based on custom-built sensor and display devices situated in research labs until approximately 2010, often based on the FTIR concept popularized by Han [11]. Since that time, commercial devices with support for these interaction modalities have also become available and are now often used to implement such systems. Nevertheless, for more specialized application or research scenarios, custom devices are still widely used.

Consequently, we are facing a wide array of devices that each implement some subset out of the far-ranging design space encompassing tangible and touch interaction. From a software developer's point-of-view, it is desirable to have some unified high-level view of all these varying devices to provide generic implementations that are not dependent on one single type of hardware. For the last 10 years, this role was filled by the TUIO protocol which has served as a quasi-standard for this entire domain. It was introduced in 2005 by Kaltenbrunner et al. [14], and was quickly adopted

Authors' addresses: Martin Kaltenbrunner, Kunstuniversität Linz, Linz, Austria, modin@yuri.at; Florian Echtler, Bauhaus-Universität Weimar, Weimar, Germany, florian.echtler@uni-weimar.de.


2573-0142/2018/6-ART8 $15.00
https://doi.org/10.1145/3229090

across a wide range of research, art, and application scenarios, despite its domain-specific origin in tangible interaction for electronic music generation.

However, we can observe that TUIO in its current version has become somewhat dated, which often led other researchers to the implementation of ad-hoc extensions to the original TUIO protocol that support additional sensor or component attributes. The original TUIO 1.0 specification defined two principal interface components: **Objects** represent simple tangible tokens, which can be identified through attached symbols, and localized with their position and orientation within an interactive surface. **Cursors** represent the basic gestural interaction from touch input or other pointing devices, and are referenced through their surface position only. Since neither objects or cursors provide any information about their actual geometry, TUIO 1.1 additionally introduced **Blobs** in order to specify the spatial extension for objects and cursors, as well as for generically untagged physical objects. TUIO also allows the representation of multiple objects, cursors and blobs through a unique **Session ID**, which is the essential attribute for the realization of multi-touch and multi-user object interaction.

In this paper, we therefore present and discuss an updated and extended version 2.0 of the TUIO protocol. We introduce a generic abstraction model of interactive surfaces, present the formal specification of the protocol itself and provide several examples of how well-known existing research projects in this context can be fully represented using the TUIO model and protocol.

## 2 TUIO1 LIMITATIONS AND AD-HOC MODIFICATIONS

It has become clear that even with the intermediate protocol extensions, this currently rather simple approach is by far not sufficient for the description of a generalized tangible interaction surface environment. While TUIO 1.1 already addressed the basic needs for an additional descriptor of the object geometry, the strict separation of Cursor, Object and Blob profiles remained one of the major limitations for any further protocol extensions. For example, the existing object profile is lacking the capability of describing different marker types, while the cursor profile is missing important attributes such as cursor IDs, types and pressure data. Finally, TUIO 1.1 is also missing detailed timing information for gesture analsyis, which unfortunately cannot be retrieved from the underlying Open Sound Control protocol bundle as originally intended. The number of potential enhancements and changes to the current protocol structure eventually justifies the following introduction of a new protocol specification, which intends to resolve the shortcomings and design limitations of the previous protocol generation.

As further indication of the need for a revised and updated TUIO protocol, we now briefly present examples where other researchers have built their own work on TUIO1, but have found it insufficient to address all their requirements.

One older example is [7], in which a large-scale FTIR-based touchscreen is combined with an overhead light source to create a shadow tracker, thereby allowing both touch points and hovering hands to be tracked. TUIO1 did not offer any way to link individual tracked objects, which was required in this case to link touch points and hands. Consequently, the authors added an ad-hoc parent-child relationship to the original TUIO protocol.

Similarly, [9] also detect a relationship between fingers and hands on a diffuse illumination tabletop system, and also had to modify the original TUIO protocol in order to convey this type of information.

[4] extend the original reacTable musical instrument to include EEG/ECG data. This involves additional tangible tokens which represent the new data sources, but also a side channel outside of the TUIO protocol to deliver the physiological signals to the music generation engine.

A more recent example, Bendy [20] uses reacTIVision and TUIO to deliver the overall position of an interactive object, but also establishes a secondary non-TUIO communications channel to deliver information about the object's deformation, which is not representable in TUIO1. Both data channels are then merged in the application.

In a similar way, the Immertable [2] uses a combination of TUIO1 and other OSC sources to fuse data from tangible tokens, a Kinect camera, Leap Motion sensors, and hand-held controllers.

Together, these examples highlight the fact that today's complex, multimodal input environments have outgrown the limited capabilities of TUIO1. Two common themes emerge among the ad-hoc extensions that other researchers have implemented: *a)* support for more complex tracking setups (e.g. hierarchical object relations such as hand and fingers, more object classes, additional information about individual objects such as color), and *b)* support for additional data channels from/to tracked objects (e.g. embedded sensors and actors inside tangible objects, buttons on hand-held controllers).

As the multitude of fragile one-shot solutions leads to a proliferation of incompatible sub-protocols that undermine the wide-ranging compatibility which was a fundamental benefit of TUIO 1, we conclude that it is time for an update of the core protocol.

## 3 AN EXTENDED ABSTRACTION MODEL

The updated version of TUIO 2.0 is based on an abstraction model designed to describe interaction on and around any interactive surface. Several aspects and details of this abstraction model arose from discussions during and after the "TUIO Hackathon" workshop at ACM ITS 2014 [15], and are also reflected in more detail in [13].

The main motivation for the conception of a generalized abstraction model was the simplification of the design and implementation process for the casual user but also the professional developer of a surface-based tangible user interface. This simplification should allow easier access to technologies, which until recently have only been available to a knowledgeable research community, which is technologically literate and also has access to the tools, documentation and education, which have been necessary to work with this kind of novel interaction paradigms. Hence the major design goal of this interaction framework is a low level of complexity, which should ideally not exceed the complexity of the single mouse pointer control in standard GUI applications. Therefore the first design iteration of the TUIO framework focused on multi-touch input and its integration into standard applications, and also added the handling of physical tokens in a similar manner. These two basic components already allow the development of basic interactive multi-touch surfaces with integrated tangible object handling, which apart of the handling of multiple controller components does in fact not introduce much additional complexity to the actual application design. Since the proposed framework targets a similar level of simplification, the discussed platform definitions may seem often very straightforward, which is of course fully intentional. The reduction of the complex interaction mechanisms, which are necessary for the construction of tangible user interfaces, to natural interaction events as well as the description of tangible components and their attributes in simple terms is one of the most important design principles of this framework.

Our approach aligns with earlier work by [3], in which the author argues that *interaction architectures* such as the one presented here must fulfill three properties: *reinterpretability*, *resilience*, and *scalability*. TUIO2 is reinterpretable due to its open design, and can be adapted to a wide range of usage scenarios as discussed below in section 6. It is also *resilient*, both to transient errors such as those introduced for example by network packet loss, and to lack of information due to a limited tracker implementation. Finally, it is scalable to large systems with hundreds of objects, as the highly

space-efficient OSC encoding will only use approximately 20 bytes per message and object, which will add up to a few kB at most. Even at frame rates of 60 Hz which are desirable for interactive systems, the required bandwidth is still easily provided by today's network infrastructures.

### 3.1 Global Context

*3.1.1 Interactive Surface.* Our abstraction model has been designed for the specific context of a two-dimensional interactive surface. This includes the surface area itself, which in the most common case is a horizontally mounted rectangular table surface, while other geometries such as square and round surfaces, or even more complex topographies are of course not excluded per-se, but not covered explicitly. The idea of an interactive surface can also be translated to a vertically organized environment, such as an augmented whiteboard for example. This surface context can also be scaled to larger dimensions covering a whole floor or wall, or scaled down to an interactive surface of a mobile input device with a small-scale touch sensitive screen.

Fig. 1. fishtank scenario: space extending above an interactive surface.

This two-dimensional surface plane can be also extended to the third dimension considering the space that elevates from the surface, which can be fully described by an absolute 3D coordinate system or partially described by a 2.5D model, which adds the relative distance to the surface (sometimes called "fishtank scenario", see figure 1).

The surface model is limited to the extension of all active surface areas, which are covered by the input sensing and multi-modal feedback hardware and will therefore neglect the description of passive surface areas, such a table border or object repository, which are nevertheless considered to be an integral part of the complete surface infrastructure, but are only relevant to the individual application scenario.

*3.1.2 Sensor Data Normalization.* The abstraction model is fully independent of the actual sensor technology used for the tracker implementation. This can include computer vision based solutions, but also optical or capacitive touch screen technology, electro-magnetic sensing as well as any other possible sensor hardware that may be employed to retrieve the necessary input data for the real-time capturing of all relevant surface, object and gesture attributes.

In order to provide a generalized abstraction for all possibly used sensor systems, which can differ significantly in their spatial or temporal resolution, this model introduces a normalized, orientation invariant coordinate system for the position data, as well as uniform component attributes where possible, or clearly defined normalized ranges of control values. These uniform data structures can then be scaled back to the desired surface dimensions by the underlying application layer, in order to provide visual feedback for example.

Additionally the model maintains a simple and straightforward data format which improves understandability and also allows for direct data interpretation by environments with limited computational infrastructure.

*3.1.3 Time Model.* The temporal resolution of the gestural input is fully based on the tracker sensor capabilities and its according configuration. Therefore all model states need to be identified with a fine-grained time stamp, which allows the later reconstruction of the full gesture including its dynamics. Since our abstraction layer provides a detailed description of the gestural input data, the application layer can perform the actual gesture recognition task based on the position and time based information. Although our model does not provide any gesture recognition or interpretation infrastructure, it provides the relevant time based input data for an intermediate gesture recognition layer such as the "Dollar" family of recognizers [1, 27, 29], a more complex gesture framework (e.g. GeForMT [16], Midas [24], GISpL [6], Proton++ [17]), or directly to the application layer itself.

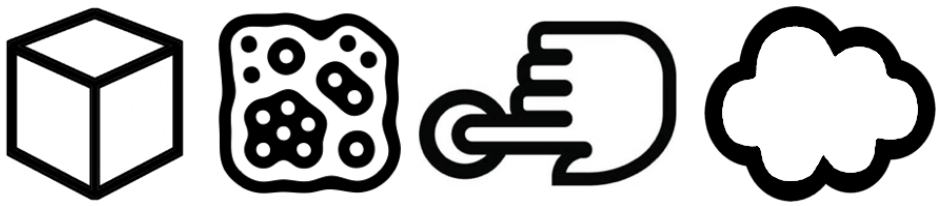

Fig. 2. TUIO 2.0 components: tokens, symbols, pointers & geometries

### 3.2 Explicit Interface Components

TUIO 2 is centered around 4 types of interface entities as shown in figure 2.

*3.2.1 Tokens.* Tokens represent **abstract tangible objects** without explicit descriptors for their actual physical appearance or geometric properties. Within the boundaries of the interactive surface these individual tangible objects can be selected and manipulated by moving and rotating them in direct contact with the surface or freely within the space above this surface. The model describes several state properties of these tokens such as their absolute position and orientation on the surface, while it also defines various classes of distinguishable tokens, which can be individually identified and tracked through fiducial markers for example.

On the other hand the model also introduces untagged generic objects, which only can be approximately described by the system, specifying their overall geometric boundaries and physical appearance.

*3.2.2 Symbols.* Physical tokens can be tagged with dedicated **marker symbols**, which primarily support the tracking system with the identification of each individual token. Using a computer vision system for example, symbols can be implemented with fiducial markers, which allow the robust identification of known visual symbols. On the other hand, simple colour tracking methods can also generate various classes of distinguishable physical tokens, while alternatively electromagnetic tags

such as RFID tags can be employed as a symbol for tagging of physical tokens. Some visual markers, such as barcodes or QR codes, as well as advanced RFID chips can also embed additional data content apart from a single identification number. Therefore in addition to the simple identification tag the model also specifies the actual symbol type, its encoding scheme and data content for each individual symbol component. Furthermore a symbol can be also used without the direct association to an actual physical object or location (see section A.3.4 for details). Examples for symbol objects are shown in figure 3.

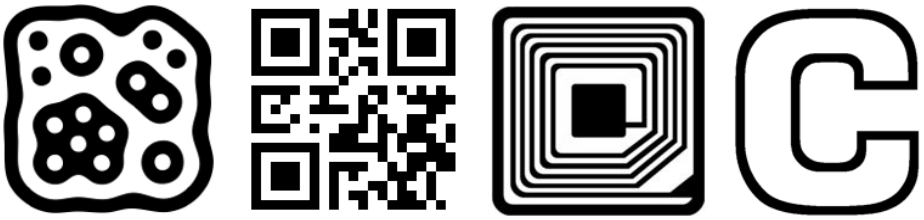

Fig. 3. symbol examples: a) fiducial b) QR code c) RFID tag d) character

*3.2.3 Pointers.* Pointers are generally one-dimensional moving pointing gestures within the two-dimensional plane, which are not directly referenced through a physical object, and therefore are conceptually intangible interfaces components. Cursor states are either generated by direct manipulation of the surface, using finger touch gestures or alternatively through dedicated pointer devices such as a mouse or pen. Cursor pointers are therefore primarily defined by their absolute position on the surface, but can also encode common additional attributes such as applied pressure, rotation angle or a region of influence. Possible examples for pointers include a touch contact, a common mouse pointer, a (digital or regular) pen, or a laser pointer.

*3.2.4 Geometries.* While Tokens and Pointers are abstract interface components, associated **geometry descriptors** can be employed to provide additional information about the spatial extension and shape of physical objects or pointer devices. These geometries define several levels of abstraction, which can be refined incrementally up to a higher level of detail if necessary. This detailed description of the object geometry can be also interpreted within the application layer, through additional object recognition for example. The various geometry descriptors can be also used as additional control dimensions within the application layer, such as mapping interactive object deformations to the control dimensions of a synthesizer for example. Possible geometry descriptors are illustrated in figure 4.

*3.2.5 Model Extension.* For platform specific adaptations, the underlying OSC data structures also allow the definition of additional components through the composition of custom messages. Therefore the TUIO protocol also provides an additional custom component mechanism, which is more tightly bound to the overall protocol semantics.

### 3.3 Component Relation

*3.3.1 Associations.* While spatial systems are generally defined by the absolute location of a physical component on an interactive surface, relational systems on the other hand are purely defined by **physical and logical component relations** [26]. Therefore this model also establishes the additional layer of component associations, which allow the representation of physical chains and even complex tree topologies of adjacent or physically connected tangible interface components.

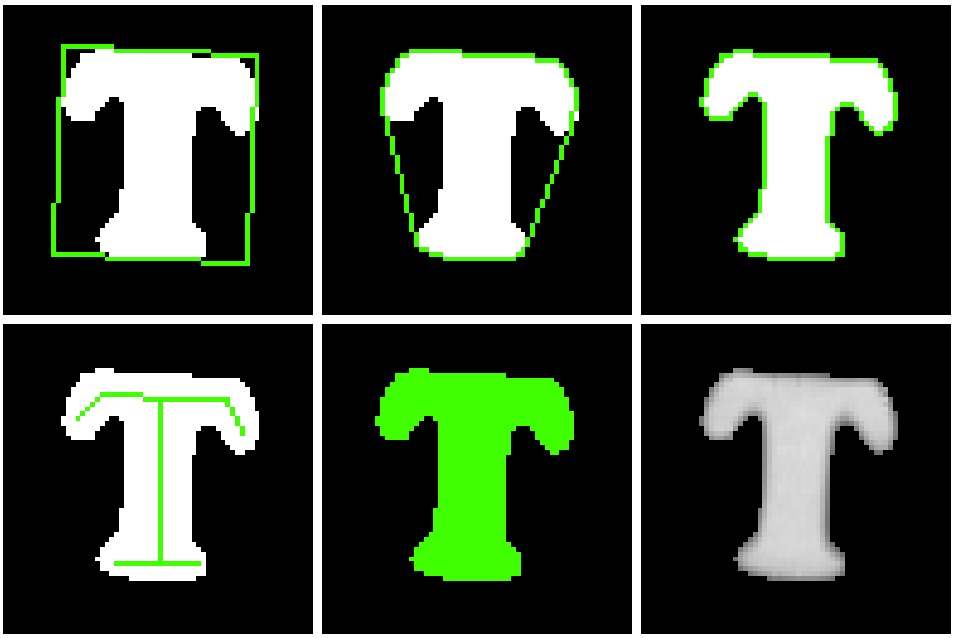

Fig. 4. geometry detail: a) bounding box b) convex hull c) contour
d) skeleton e) area spans f) raw data

Additional container associations allow the representation of **physically nested components**. Association examples are provided in 5.

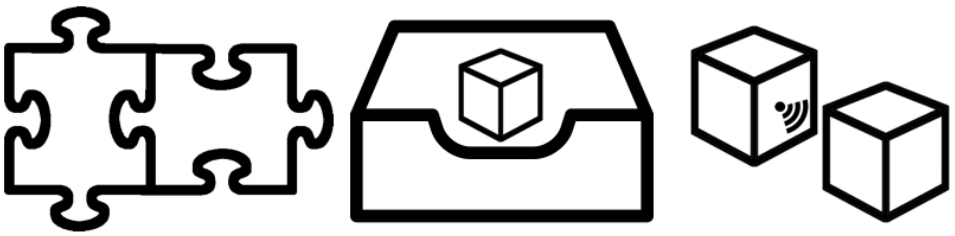

Fig. 5. component relations: a) physical link b) container c) signal

*3.3.2 Controls and Signals.* Tokens or other physical interface components can also be augmented with additionally associated **control dimensions** for the description of the state of associated buttons, knobs, faders, or other dedicated control values retrieved from pressure or temperature sensors etc., which can be optionally attached to the physical body of each individual component.

In addition to the establishment of physical and logical associations, physical interface components may also exchange simple **signals**. Therefore a directional signal infrastructure between individual components allows to trigger events or **exchange data** blocks.

### 3.4 Component Context

*3.4.1 Surface Constraints.* While the spatial extension of the interactive surface represents the principal constraint for the overall tangible interaction environment, TUIO provides no explicit

definition of a dedicated class of constraint interface elements. Constraints may be defined in the application domain, digitally within the visual surface layout or physically with the dedicated definition of tokens or physical geometries that serve as constraint interface components. If the sensor system is capable of determining the physical relation of a dedicated token-constraint system, the model allows the explicit definition of a container object and its associated tokens.

*3.4.2 User Representation.* The user as such is not directly represented in the abstraction model, but is obviously driving the continuous state changes of pointers and tokens, which are generated by the user actions and gestures actuating upon the physical objects and the surface, and can be indirectly associated to a user. Since some input devices allow user identification, all components can be associated to a specific anonymous user ID, which allows the user association of gestures generated by an individual. Furthermore, pointer components allow the specification of various body parts such as finger, hand, body and head which also can be associated to an individual user.

### 3.5 Component Gestures

Various types of gestures that are being performed with or on the physical interface components can be deduced from consecutive attribute updates, such as the position and movement of pointers and tokens, the rotation of tokens and untagged objects as well as the continuous geometry changes of malleable tangible objects.

*3.5.1 Pointing Gestures.* can be generated directly from single or multiple pointer positions, primarily when an event for adding or removing a pointer component has been decoded from the state model. Within the application model, these pointing gestures can be associated with either digitally generated surface components or with a nearby physical interface component.

*3.5.2 Motion Gestures.* can be derived from continuous pointer, token and geometry displacements, typically marked as a session that develops over the lifespan of each component from its association with the surface context until its removal.

*3.5.3 Manipulation Gestures.* are derived from the interpretation of object state or geometry changes, which result from the rotation and eventually the deformation of generic tangible objects, resulting in continuous changes of the object's bounds, contour or skeleton, which can be interpreted as squeezing or bending gestures.

*3.5.4 Control Gestures.* can be equivalently derived from the same data used for manipulation gestures by directly mapping these changes of object geometry to control parameters within the application model. The handling of the dedicated controller components directly provides additional control dimensions to the application model.

*3.5.5 Association Gestures.* can be either derived from the direct interpretation of the two component association types for connect and disconnect events as well as container events. On the other hand the model also allows the interpretation of changes in spatial adjacencies as indirect object association gestures.

## 4 MODEL INTEGRATION

As shown above, this new abstraction model does not intend to define an universal real-world description of any observable interaction environment, neither does it provide an infrastructure for the direct encoding of raw sensor data. It is rather defining a simplified semantics for an abstract description of the physical environment, which primarily has been designed for the representation of interface components states within tangible multi-touch applications based on interactive surfaces.

Thus our model is fully hardware independent and application invariant since it basically provides a normalized representation of physical interface components, without any reference to specific hardware configurations or application scenarios. An underlying application layer therefore can build upon the provided component abstraction, defining its actual application model only locally. This eventually allows the installation of such applications on different hardware platforms, which provide their feature set through an adequate implementation of our semantic model.

The recognition and representation of high-level user gestures performed with the hands or the full body are not represented within this model though. The gesture modelling and recognition should be generally bound to the application domain, which is not explicitly part of our purely physical interaction model and the according abstractions. The detailed component attributes and the integrated timing infrastructure provide the necessary high-level input data as well as the necessary semantic context for gesture recognition and even object recognition based on their geometries.

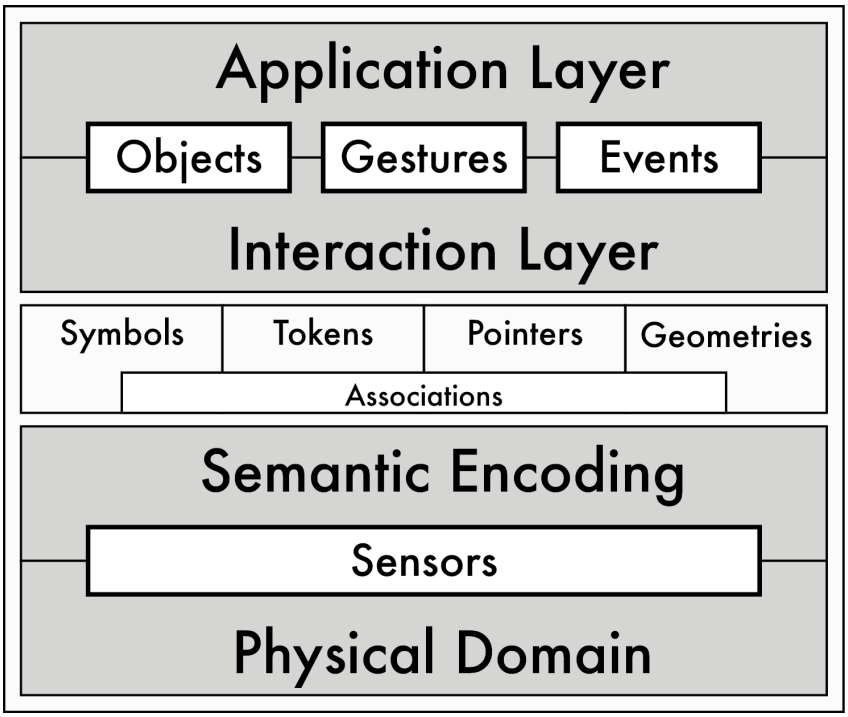


Fig. 6. Layers of the Abstraction Model

A typical physical layer is usually comprised of an interactive surface and the related tangible objects. These interface components usually contain an embedded sensor system or are observed externally, which provides the raw input data that is retrieved from surface interactions or object manipulations.

Each individual hardware configuration therefore needs to integrate the low-level data acquisition with the high-level semantic model representation. This usually involves the mapping from a device-specific input system with a software implementation of our model. This infrastructure generally takes care of the proper encoding at the hardware level, its transmission through an

adequate channel, as well its decoding at the application layer.

After the transmission and decoding, the semantic description of the physical interaction environment is provided through an application programming interface, which encapsulates object states and surface events. This interface can either be integrated directly into the final application or previously processed by an intermediate gesture recognition layer. The actual application logic finally determines the appearance of the according digital feedback through the integrated visual, acoustic or haptic displays. Although our model does not explicitly define any feedback channel, it can be also employed to encode physical interface changes for actuated system configurations. A visual depiction of how TUIO's concepts provide an interface between the physical domain and the interaction layer is shown in figure 6.

Taking the example of Echtler's et al. multitouch software architecture[8], the TUIO abstraction can be simply represented within the hardware abstraction (and optionally calibration) layer in this case, allowing for the integration of any TUIO pointer enabled hardware implementation into its architecture. While this particular multitouch model only integrates the subset of pointer and token components, it could be subsequently extended with geometry and symbol descriptors that are provided by the abstraction model.

## 5 TUIO 2.0 PROTOCOL SPECIFICATION

This chapter presents the implementation of an abstraction framework, which shall meet the requirements for the comprehensive description of state-of-the-art tangible interfaces and multi-pointer surfaces. For that purpose an extensive protocol has been defined, which allows the encoding and transmission of a tangible interface component abstraction, such as tokens, pointers and geometries as well as additional symbols and controls in the context of an interactive surface. The protocol extends and replaces the original TUIO specification, therefore the reader should be familiar with the general idea and structure of the previous TUIO 1.1 protocol generation [14]. As in its predecessor, the TUIO 2.0 components and attributes are encoded using the Open Sound Control (OSC) format. Details regarding the individual protocol messages are presented in appendix A.

This protocol version introduces many additional features compared to TUIO 1.1. This includes timing information, several additional component attributes such as finger pressure, and code descriptors for symbol types such as data matrix labels or RFID tags. TUIO 2.0 also encodes the precise object geometry with general bounds, contour and skeleton descriptors which can be used to describe untagged tangible objects or retrieve additional geometry information for tokens or pointers. Many syntactic changes have become necessary in order to define more OSC compliant messages and bundles, which together with the additional features should justify the version jump at the cost of backwards-compatibility at the protocol level. TUIO 2.0 client implementations can provide additional backwards compatibility by handling both TUIO 1.* and TUIO 2.0 protocol generations though. An overview of and comparison between the entity types available in TUIO 1.1 and TUIO 2.0 is presented in table 1.

The various types of identifiers used in TUIO 2.0 have distinct meaning, which shall be briefly differentiated here. The session ID is an ephemeral number which identifies a single, specific object during its lifetime within the interaction space. When an object is removed from and reintroduced into this space, it will receive a new session ID. The component ID, on the other hand, is used for objects which can be uniquely identified even after removal and re-introduction, such as a specific printed fiducial. Several fiducials of the same type will share the same type ID. Finally, if a sensor system is capable of identifying individual users and assigning objects to them, this can be reflected through the user ID.

| Abstract Concept | TUIO 1.1 entities | TUIO 2.0 entities |
|---|---|---|
| Tangible objects | 2Dobj, 25Dobj, 3Dobj | tok, t3d |
| Pointers/cursors | 2Dcur, 25Dcur, 3Dcur | ptr, p3d |
| Bounding boxes | 2Dblb, 25Dblb, 3Dblb | bnd, b3d |
| Object geometry | – | chg[1], ocg/icg[2], skg/s3d[3], svg[4], arg[5], raw[6] |
| Object associations | – | coa[7], lia[8], lla[9], lta[10] |
| Symbols, Data | – | sym, dat |
| Controls, Signals | – | ctl, sig |
| Unique lifetime identifier | session ID s | session ID s_id |
| Persistent object identifier | "class" ID c | component ID c_id |
| Class identifier | – | type ID t_id |
| User identifier | – | user ID u_id |

Table 1. Comparison of TUIO 1.1 and TUIO 2.0 entity types. Abbreviations (see also section A) – Geometries: 1. convex hull, 2. outer/inner contour, 3. skeleton, 4. skeleton volume, 5. area, 6. raw sensor data. – Associations: 7. container, 8. link, 9. linked list, 10. linked tree.

As an example, all reacTIVision Amoeba symbols will have the same type ID, but different persistent component IDs and also different ephemeral session IDs. This allows several instances of the same symbol to be used at the same time and differentiated through their session IDs, while the meaning of each symbol is codified through the component ID, and differentiation from other types of tokens is possible through the type ID. For pointer objects, a list of pre-defined type IDs for common pointing devices can be found in section A.3.2.

## 6 EXAMPLE PLATFORM ENCODINGS

This section intends to showcase the potential of the proposed abstraction model and its protocol implementation by defining several example encodings for various existing tangible interactive surface platforms. Due to the previously discussed application context, this list includes seminal examples from the interactive surface research community, more recent commercial products, as well as some tangible musical applications. In terms of [3], we focus on the *descriptive power* of TUIO 2.0, i.e. its capability to describe existing interactive systems within the underlying model. This is also in line with more recent work on toolkit evaluation by Ledo et al. [19] - within the framework introduced in their paper, we provide validation of our toolkit in the form of 1 - Demonstration (Replicated Examples + Design Space Exploration, see also section 3), and in the form of 4 - Heuristics (Discussion).

The individual component features and attributes that are relevant for the representation of each example platform will be highlighted. Since there does not yet exist an application or platform that implements the full feature set of the TUIO 2.0 model capabilities, most applications will generally implement a specific sub-set of the available components and their attributes.

### 6.1 Commercial Input Devices

In this section, we discuss common commercially available input devices and how their features can be represented using TUIO2.

*6.1.1 Pointer Types: iPad Pro.* Apple's current iPad Pro tablet devices[1] provide a comprehensive feature set of a state-of-the-art tablet platform. In addition to the common multi-touch interaction

[1] http://www.apple.com/ipad-pro/

this tabled also allows additional pencil input, providing extended attributes such as pressure, rotation angle and shear angle. Finger touches can be distinguished from the pencil through use of the Type ID field, and provide additional information about the finger footprint size. While the current iPhone modules also provide pressure sensitive (force) touch input, this feature is not (yet) available on the tablets. A TUIO2 implementation for iOS devices such as the iPad and iPhone can be therefore realized using most of the available Pointer attributes: type, position, pressure, angle, shear angle and size.

```
/tuio2/ptr s_id tu_id c_id x_pos y_pos angle shear radius press
```

Fig. 7. iPad Pro pencil and touch interaction. Photo: Apple Incorporated

TUIO1 would not be able to represent shear angle and pressure (except as a hack via the area attribute), and would require an ad-hoc mapping of an arbitrary class ID to represent the pen (for differentiation from touch points).

*6.1.2 Pointers and Controls: Wheel Mouse.* The primary functionality of a common computer mouse can be easily reflected by the encoding of its position attributes with an according Pointer message, where the Type ID is set to the Mouse pointer type ID 13. In addition to that, a simple mouse-click for a generic mouse can be encoded, by setting the pressure attribute to one. Since such a device has often three buttons, where the middle button has been designed as a wheel, these additional control dimensions have to be encoded using a complementary Control message. The according CTL message needs to encode three button states plus the up and down movements of the wheel. This can be simply represented by four control values, where the first three button controls can be set to 1 for the *pressed* state and the last wheel control attribute can be set to 1 for *upward* movements and -1 for *downward* movements. All four values are set to 0 while they are not activated.

```
/tuio2/ptr s_id tu_id c_id x_pos y_pos angle shear radius press
/tuio2/ctl s_id left_btn wheel_btn right_btn wheel
```

TUIO1 would not be able to differentiate between hover state and individual button presses, and would have no straightforward way to communicate the state of the scroll wheel at all. As an example, TISCH [7] integrated mouse wheel data via TUIO1 by (ab-)using the rotation angle attribute of a generic cursor object.

*6.1.3 Pointers and Pressure Maps: Sensel Morph.* Professional drawing tablets, such as the popular Wacom devices generally provide a feature set, which is comparable to the pencil input from the iPad Pro as described above. This generally includes many advanced pointer attributes such as type, position, rotation and shear angle as well as pressure. While in such a device the pressure attribute is directly sensed from within the pen controller, today there also exist pressure sensitive tablets such as the Sensel Morph.[2] Although TUIO 2.0 can encode generic multi-touch input, also including finger pressure with the according Pointer attribute, this device is generally capable of providing an overall "pressure image" of any object in contact with its surface. Thus for the encoding of the total capabilities of this device not only a Pointer message can be employed for touch interaction, but also additional Bounds or Geometry messages, such as the Outer Contour of a touching object. Furthermore TUIO 2.0 can provide the full pressure map for each object through an additional Area and Raw Geometry message.

```
/tuio2/ptr s_id tu_id c_id x_pos y_pos angle shear radius press
/tuio2/bnd s_id x_pos y_pos angle width height area
/tuio2/ocg s_id x_p0 y_p0 ... x_pN y_pN
/tuio2/arg s_id x0 y0 w0 ... xN yN wN
/tuio2/raw s_id width data
```

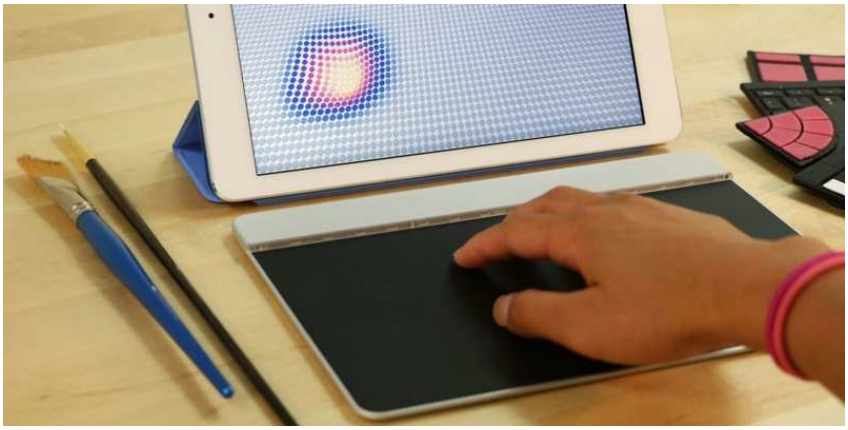

Fig. 8. The Morph tablet and its pressure map display
Photo: Sensel Incorporated

With TUIO1, only the abstracted pointer messages (again without pressure and shear angle) and their bounding ellipses could be communicated, but none of the richer contour and pressure information.

*6.1.4 Tokens as Pointer: Surface Studio.* Since Microsoft abandoned its tangible Surface tables, it concentrated on the further integration of multi-touch and pen input into its current Windows 10 operating systems and the related tablet and notebook hardware. Nevertheless with the release of its latest Surface Studio platform the company reintroduced the concept of the tangible object with the Surface Dial interface. Although the device has the affordance of a physical token, by providing an additional rotary knob, it can be generically represented through a dedicated Pointer

[2] http://www.sensel.com/

type by using the according pressure attribute for the push button, the rotation attribute for the wheel controller and the radius attribute for the controller's size. Alternatively the Dial could be also represented through a TOK message plus an additional CTL message for the push button, but in this case the PTR already provides all necessary attributes and also corresponds to its actual representation within Windows.

**/tuio2/ptr** s_id **tu_id** c_id x_pos y_pos **angle** shear **radius press**

TUIO1 could only provide basic coordinate information for the pointers and the rotation angle, but would again require an ad-hoc mapping for class IDs to differentiate pen, touch and dial. Also, pressure data for the knob could again only be encoded by misusing an attribute with different intended semantics.

Fig. 9. The Surface Studio with pen and dial.
Photo: Microsoft Corporation

### 6.2 Tabletop Systems

In this section, we review both commercial and custom-built tabletop interface systems, and discuss their representation in TUIO2.

*6.2.1 Pointer Identification: Diamondtouch.* The MERL Diamondtouch platform[5] represents a multi-touch surface with the special capability of user identification. Therefore each individual touch input can be assigned to a specific user. In our model these capabilities can be encoded by making use of the Pointer component only, since this component alone is capable of encoding the provided location information and User ID. Additionally the Pointer Type ID can be preset to the generic Finger type, although the system is not capable of distinguishing an individual finger or hand. The platform is neither capable of detecting pressure values nor the hovering state, therefore the according attribute will be neglected in the encoding. Since the system is only reporting the interpolated centroid for each touch contact, there is now additional information available that can determine the finger size. Therefore the according width attribute can be either set to the diameter of an average finger relative to the actual sensor resolution, or also set to zero. Although other non-intrusive user identification methods[23] do not require any physical user tagging, they are

equally represented by the simple User ID on the protocol level.

```
/tuio2/ptr s_id tu_id c_id x_pos y_pos angle shear radius press
```

Fig. 10. multi-user identification on the Diamond Touch. Photo: MERL

TUIO1 had no concept of differentiating between users, therefore the only solution to represent full data from the touch sensor would, once again, have involved reassigning a semantically different attribute in an application-specific way.

*6.2.2 Tokens, Pointers, Symbols & Bounds: Surface.* The original Microsoft Surface platform and its successor SUR40 supported multi-touch input as well as tagged and untagged object recognition. For object tagging there were two different fiducial types available, where the identity tag is capable of encoding a 128-bit symbol. The generic Contact component can distinguish between touches, tags and untagged blobs. Applying the TUIO abstraction model to these platform features therefore suggests the definition of Pointers, Tokens, Symbols and Bounds components. The primary Token component refers to any tagged physical object, which can have either an attached ByteTag, which is defined by the Type ID 0 and a range of 256 component IDs, or by an alternative Type ID 1 referring to an IdentityTag, which can be specified in further detail using an additional Symbol message.

Finger touch contacts are defined by the according Pointer message, which can be extended with an additional Bounds message that specifies the contact geometry with the according dimension and rotation attributes. The Bounds message can be also sent individually for all contacts, which have been neither identified as finger touch or tagged object.

```
/tuio2/ptr s_id tu_id c_id x_pos y_pos angle shear radius press
/tuio2/bnd s_id x_pos y_pos angle width height area
/tuio2/tok s_id tu_id c_id x_pos y_pos angle
/tuio2/sym s_id tu_id c_id ms/id 0x00...00
```

With TUIO1, it would neither have been possible to differentiate between the two token types, nor to transport the 128-bit message of the identity token (particularly difficult as all OSC types are only 32 bit wide by default).

*6.2.3 Tokens, Pointers and Geometries: reacTIVision.* The current development version of reacTIVision 1.6 [12] and its according feature set are still based on the current TUIO 1.1 protocol

specification. The reacTIVision software is capable of providing input data from tagged tangible objects as well as fingers and untagged physical objects touching the surface. Following the new TUIO 2.0 platform model these features can be represented employing the Token, Pointer and Bounds components. In addition to the equivalent attributes that are already used within the TUIO 2Dobj, 2Dcur and 2Dblb profiles the platform is also capable of providing the more detailed Contour geometry of the detected component geometries.

For the Pointer component the platform can additionally implement the Finger Type ID as well as the pointer width attribute. The pressure attributes and hovering state are not yet available, although they may be approximated from the finger geometry. Since reacTIVision presently provides the additional Yamaarashi symbols in addition to the three variations of the amoeba symbols with different sizes and ranges, the Type ID can be employed to distinguish these symbol types, which can be consequently used for different tasks.

The Bounds component can be either used to provide additional geometry information about the existing Token and Pointer instances as well as for the encoding of dedicated untagged object components. Since the application internally already maintains the full contour information of each detected contact region, these attributes can be directly encoded within an optional OCG outer contour message. Additionally the full component area can be provided using the according ARG message, since reacTIVision internally uses exactly this span list representation for its region data structure.

```
/tuio2/tok s_id tu_id c_id x_pos y_pos angle
/tuio2/ptr s_id tu_id c_id x_pos y_pos angle shear radius press
/tuio2/bnd s_id x_pos y_pos angle width height area
/tuio2/ocg s_id x_p0 y_p0 ... x_pN y_pN
```

TUIO1 did not offer any encoding for detailed contour data beyond a simple bounding ellipse, and would also not have been capable of differentiating marker types without an ad-hoc mapping of class IDs.


Fig. 11. Two SLAP widgets with integrated controls.
Photos: Malte Weiss et.al.

*6.2.4 Tokens and Controls: Slap Widgets.* Weiss' et.al. SLAP widgets[28] are a well known example of enhanced tabletop tokens, which provide additional interactivity through simple mechanical components. This interactivity requires the association of additional control dimensions to these physical widgets, which can be encoded by the combination of a Token component, with an accordingly formatted Control message. While the most simple physical push button and slider examples can be represented with a CTL message providing a single boolean attribute, more complex control configurations can be realized through any combination of boolean and continuous controls. Even

a full octave of a piano keyboard including key velocity can be represented by control message including an array of twelve floating point attributes, which is associated to a token and/or an according bounds message for its overall position and geometry.

```
/tuio2/tok s_id tu_id c_id x_pos y_pos angle
/tuio2/ctl s_id bool/int
```

Conversely, TUIO1 did not have any concept of control or sensor data at all which could have been associated with a specific tangible.

*6.2.5 Spatial Interaction: TISCH.* Echtler's TISCH [7] provides a versatile environment for finger and hand tracking through different methods. It combines various sensor technologies in order to track a user's hand above and on the surface, by combining an optical shadow tracking technique with standard surface touch tracking. This illustrates the tracking of the same pointer component such as a whole hand and/or its individual fingers from various perspectives. For this purpose Pointer components can be employed for the representation of each hand and finger pointer and combine those through an optional Link Association. The complementary two-dimensional hand shadow from the surface (in the form of its contour geometry) can be associated to the previous hand pointer by using the same Session ID.

```
/tuio2/ptr s_id tu_id c_id x_pos y_pos angle shear radius press
/tuio2/ocg s_id x_p0 y_p0 ... x_pN y_pN
/tuio2/ala s_id0 ... s_idN
/tuio2/lia s_id false s_id0 port ... s_idN port
```

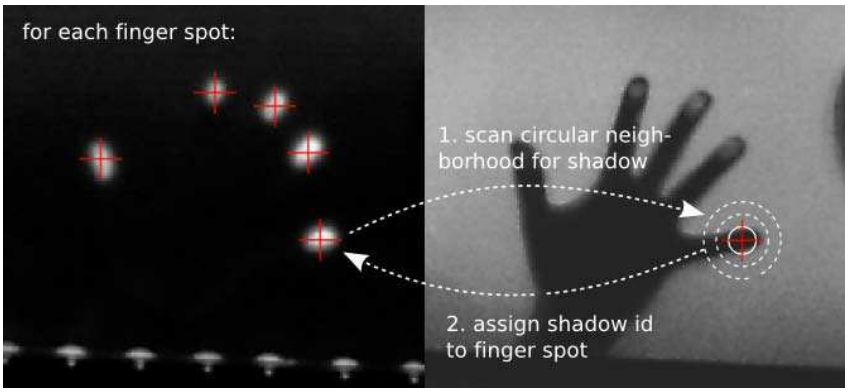


Fig. 12. Touch and shadow image processing in TISCH.
Photos: Florian Echtler et.al.

In TUIO1, particularly the concept of associating various individual components to form a higher-level composite object was not available, leading to use of custom extension in this case once more.

*6.2.6 Actuated Tokens: Pico.* Patten's Pico platform[22] provides object tracking via embedded electromagnetic tags including the possibility of moving the objects using an array of electromagnets. While the combination of two tags within a single object allows the determination of its position

and orientation, the actuator component only allows changing the position of an according object. Although there is no direct implementation available for pointing components such as touch recognition or dedicated pointing devices, the platform can make use of single-tag objects for the realization of pointing or selection tasks. The platform additionally makes extensive use of various untagged physical objects, although these additional components are not actively tracked by the sensor hardware. Nevertheless their position and physical properties have a direct impact on the active tokens, since they impose strong physical constraints for their location and freedom of movement.

Therefore the principal interface component for the encoding of the capabilities of this platform are Tokens with their according Component ID, position and orientation. The actuator on the other hand can be actually driven by according Token messages sent from the application layer back to the hardware platform, in order to update their software defined positions.

outbound: **/tuio2/tok** s_id tu_id c_id x_pos y_pos angle
inbound: **/tuio2/tok** s_id tu_id c_id x_pos y_pos angle

Fig. 13. Physical constraints on the Pico platform. Photos: James Patten

Additional Geometry components could be employed to describe the outline of the objects used as physical constraints, but the Pico platform makes use of the physical properties without sending an actual digital representation to the application layer. Also the dedicated pointing components are rather determined at the application layer than within the actual sensor hardware, therefore they can be either encoded with an according TOK message or alternatively with a dedicated PTR message.

**/tuio2/tok** s_id tu_id c_id x_pos y_pos angle
**/tuio2/ptr** s_id tu_id c_id x_pos y_pos angle shear radius press
**/tuio2/ocg** s_id x_p0 y_p0 ... x_pN y_pN

Although TUIO1 would also have been capable of being used bidirectionally (with PTR messages being sent in both directions), it would not have allowed rich representations of geometry of passive objects on the surface.

### 6.3 Standalone Tangibles

In this section, we discuss systems which are not bound to a specific surface area, but rather use computer vision or relative position sensing to allow interaction with tangible objects in arbitrary environments.

*6.3.1 Symbols and Bounds: Papier Maché.* Klemmer's Papier Maché[18] toolkit combines various tangible input technologies within a single application model. Under the overall concept of Phobs it integrates several computer vision and physical computing elements. VisionPhobs are describing the boundaries of untagged physical objects, while TagPhobs can represent barcode and RFID tags. Within our abstraction model the properties of a VisionPhob can be encoded by a Bounds component, which provides all the necessary attributes. TagPhobs on the other hand can be encoded by an according Symbol component, which can additionally distinguish between the two tag types that are available within this toolkit.

```
/tuio2/bnd s_id x_pos y_pos angle width height area
/tuio2/sym s_id tu_id c_id ean/13 5901234123457
/tuio2/sym s_id tu_id c_id rfid 0x04c5aa51962280
```

The major limitation of TUIO1 in this case would have been the inability to differentiate token types, and also to transmit the actual symbol contents without custom ad-hoc protocol modifications.

*6.3.2 Physical Associations: Triangles.* Gorbet's Triangles[10] are a generic hardware platform for the creation of constructive assemblies using triangular components, which can be connected to each other on either side. The resulting topologies are the basis for various application scenarios, which accordingly need to have access to the current configuration and connection events of the full construction. In order to represent the components and properties within our model, a combination of Tokens and Link Associations can be employed. The TOK message represents each individual instance of a triangle and only serves as a reference for all currently present triangles. Since neither the position or orientation of these objects are available, none of these attributes are necessary for the representation of these components. A series of LIA messages encodes the topology of the full construction by specifying the links between individual objects. The Session ID and the connecting side of each triangle are specified in the according component list of the Link Association message.

```
/tuio2/tok s_id tu_id c_id x_pos y_pos angle
/tuio2/ala s_id0 ... s_idN
/tuio2/lia s_id true s_id0 port ... s_idN port
```

As TUIO1 has no concept of inter-object links, the representation of such a structure would have been difficult, especially as no absolute coordinates within the TOK message would have been available.

*6.3.3 Logical Associations: Siftables.* Merrill's Siftables[21] (later commercialized as Sifteo Cubes) represent an autonomous tangible interaction platform based on gestural control and the establishment of logical links between objects. Although the absolute spatial position is not relevant in this scenario the individual objects can provide their three-dimensional orientation. Therefore a 3D Token component can be used for the representation of the individual blocks, while only encoding the Component ID and 3D angle attributes. The logical object relationships are represented by the according combination of ALA and LIA Link Association messages, where the physical connection

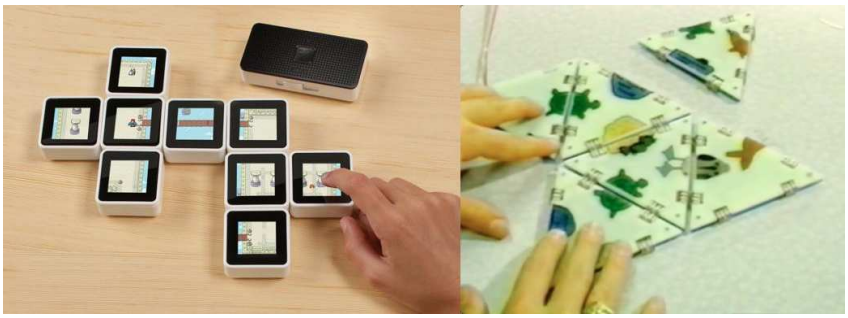

Fig. 14. a) Sifteo Cubes and b) Triangles
Photos: David Merrill and Matthew Gorbet

attribute is set to false.

```
/tuio2/t3d s_id tu_id c_id x_pos y_pos z_pos angle x_ax y_ax z_ax
/tuio2/ala s_id0 ... s_idN
/tuio2/lia s_id false s_id0 port ... s_idN port
```

Similar to the Triangles example above, TUIO1 would have been unable to provide a clear way to describe inter-object links (although the 3D orientation would have been representable using the `3Dobj` token).

### 6.4 Musical Interfaces

Due to TUIO's provenance from the application scenario of electronic music creation, we also discuss two non-academic projects which focus on novel forms of interaction with synthesizers.

*6.4.1 Signals: Tangible Sequencer.* Jeffrey Traer's tangible sequencer[3] is comprised of a set of coloured cubes with an illuminated push button, which can send optical trigger signals to the next object that is within their vicinity. This configuration and its according behaviour can be implemented by using a Token component in conjunction with an associated Signal message. The tokens are actually only needed to provide a reference for each tangible object, although their spatial position and orientation attributes are not relevant in this context. Therefore an alternative symbol message, which for example encodes the actual colour of an individual cube could be used. Since the signals sent between objects represent a simple trigger event, the according Signal ID attribute can be simple set to zero.

```
/tuio2/tok s_id0 tu_id c_id x_pos y_pos angle
/tuio2/tok s_id1 tu_id c_id x_pos y_pos angle
/tuio2/sig s_id0 0 s_id1
```

TUIO1 does not encode the concept of inter-object messages, therefore a representation of this system would only be possible through ad-hoc extensions.

*6.4.2 Tokens and Geometries: Scrapple.* Golan Levin's Scrapple sequencer[4] is based on detection and analysis of the arbitrary shape of physical objects, which is sonified with varying pitch, timbre

[3] http://murderandcreate.com/tangiblesequencer/
[4] http://www.flong.com/projects/scrapple/

Fig. 15. a) Scrapple installation and b) Tangible Sequencer
Photos: Golan Levin and Jeffrey Traer

and duration depending on the position, size and extension of the physical object. Therefore a simple combination of a Bounds component together with an OCG message, which is providing the sufficient outer contour information for each object placed onto the table. The system could be eventually fully realized by providing the Contour information only, but since the Bounds component already represents a convenient simplification of the object, it can be included as well to avoid further processing at the application layer.

```
/tuio2/bnd s_id x_pos y_pos angle width height area
/tuio2/ocg s_id x_p0 y_p0 ... x_pN y_pN
```

TUIO1 would have reduced all shapes in this scenario to bounding ellipses, which would not be sufficient to cover all possible configurations that users may want to construct.

## 7 CONCLUSION

We presented an update to the widely used TUIO protocol with the aim to correct numerous shortcomings and to address necessary ad-hoc modifications that had been introduced by other researchers over the last years. An online version of the protocol specification is available at https://www.tuio.org/?tuio20 (see also appendix). A reference implementation of the protocol in C++ is available at https://github.com/mkalten/TUIO20_CPP.

The **original abstraction model** had defined the three basic interface components of **Tokens** (as objects), **Pointers** (as cursors) and **Bounds** (as blobs), which represent the fundamental elements of tangible interactive surfaces. Based on this model we defined and published a now **de-facto standard protocol** that encapsulates these component states and attributes. The protocol abstraction provided the necessary semantic description of tangible interface components independently of the actual hardware capabilities and sensor technology, allowing for the development of platform-independent applications.

An analysis of various community contributions and third-party tangible interaction platforms, eventually led to the definition of an **extended abstraction model**. While **Tokens**, **Pointers** & **Bounds** still represent the core components of this new abstraction model with an extended set of attributes, an additional intangible **Symbol** component has been introduced in addition to a set of **Geometries** describing the physical appearance. Furthermore this extended model also allows the encapsulation of additional component **Controls** as well as the description of physical or logical component **Associations**

The provided example encodings have shown, that a large number of tangible interaction platforms and applications can be easily represented with the components and attributes provided by our extended abstraction model and can subsequently be encoded within a TUIO 2.0 protocol syntax. This on the one hand allows the abstraction of currently available available hardware platforms and interactive devices, and on the other hand the realization of versatile tangible application scenarios based on its extended feature set.

While it is still unlikely to provide a single platform that implements the full feature set of all available components and attributes of the TUIO 2.0 protocol capabilities, its definition, implementation and availability to the public domain may motivate the further development and improvement of existing hard- and software toolkits in order to extend their capabilities according to this model.

## REFERENCES


[1] Lisa Anthony and Jacob O. Wobbrock. 2012. $N-protractor: A Fast and Accurate Multistroke Recognizer. In *Proceedings of Graphics Interface 2012 (GI '12)*. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 117–120. http://dl.acm.org/citation.cfm?id=2305276.2305296

[2] Sandra Baldassarri, Eva Cerezo, and José Ramón Beltrán. 2017. Immertable: A Configurable and Customizable Tangible Tabletop for Audiovisual and Musical Control. In *Proceedings of the XVIII International Conference on Human Computer Interaction (Interaccion '17)*. ACM, New York, NY, USA, Article 32, 8 pages. https://doi.org/10.1145/3123818.3123842

[3] Michel Beaudouin-Lafon. 2004. Designing Interaction, Not Interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI '04)*. ACM, New York, NY, USA, 15–22. https://doi.org/10.1145/989863.989865

[4] S. Mealla Cincuegrani, S. Jordà, and A. Väljamäe. 2016. Physiopucks: Increasing User Motivation by Combining Tangible and Implicit Physiological Interaction. *ACM Trans. Comput.-Hum. Interact.* 23, 1, Article 4 (Feb. 2016), 22 pages. https://doi.org/10.1145/2838732

[5] Paul Dietz and Darren Leigh. 2001. DiamondTouch: A Multi-user Touch Technology. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology*. ACM. https://doi.org/10.1145/502348.502389

[6] Florian Echtler and Andreas Butz. 2012. GISpL: Gestures Made Easy. In *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction*. ACM. https://doi.org/10.1145/2148131.2148181

[7] Florian Echtler, Manuel Huber, and Gudrun Klinker. 2008. Shadow Tracking on Multi-touch Tables. In *Proceedings of the Working Conference on Advanced Visual Interfaces*. ACM. https://doi.org/10.1145/1385569.1385640

[8] Florian Echtler and Gudrun Klinker. 2008. A Multitouch Software Architecture. In *Proceedings of the 5th Nordic Conference on Human-computer Interaction: Building Bridges*. ACM, New York, NY. https://doi.org/10.1145/1463160.1463220

[9] Philipp Ewerling, Alexander Kulik, and Bernd Froehlich. 2012. Finger and Hand Detection for Multi-touch Interfaces Based on Maximally Stable Extremal Regions. In *Proceedings of the 2012 ACM International Conference on Interactive Tabletops and Surfaces (ITS '12)*. ACM, New York, NY, USA, 173–182. https://doi.org/10.1145/2396636.2396663

[10] Matthew G. Gorbet, Maggie Orth, and Hiroshi Ishii. 1998. Triangles: Tangible Interface for Manipulation and Exploration of Digital Information Topography. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. https://doi.org/10.1145/274644.274652

[11] Jefferson Y. Han. 2005. Multi-touch Sensing Through Frustrated Total Internal Reflection. In *ACM SIGGRAPH 2005 Sketches (SIGGRAPH '05)*. ACM. https://doi.org/10.1145/1187112.1187287

[12] Martin Kaltenbrunner. 2009. reacTIVision and TUIO: A Tangible Tabletop Toolkit. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces (ITS2009)*. ACM, Banff, Canada.

[13] Martin Kaltenbrunner. 2018. *An Abstraction Framework for Tangible Interactive Surfaces*. Doctoral Thesis. Bauhaus-Universität Weimar. http://nbn-resolving.de/urn:nbn:de:gbv:wim2-20180205-37178

[14] Martin Kaltenbrunner, Till Bovermann, Ross Bencina, and Enrico Costanza. 2005. TUIO - A Protocol for Table Based Tangible User Interfaces. In *Proceedings of the 6th International Workshop on Gesture in Human-Computer Interaction and Simulation (GW 2005)*. Vannes, France.

[15] Martin Kaltenbrunner and Florian Echtler. 2014. TUIO Hackathon. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces (ITS2014)*. Dresden, Germany. https://doi.org/10.1145/2669485.2669631

[16] Dietrich Kammer, Jan Wojdziak, Mandy Keck, Rainer Groh, and Severin Taranko. 2010. Towards a Formalization of Multi-Touch Gestures. In *ACM International Conference on Interactive Tabletops and Surfaces (ITS '10)*. ACM, New York, NY, USA, 49–58. https://doi.org/10.1145/1936652.1936662

[17] Kenrick Kin, Björn Hartmann, Tony DeRose, and Maneesh Agrawala. 2012. Proton++: A Customizable Declarative Multitouch Framework. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*

*(UIST '12)*. ACM, New York, NY, USA, 477–486. https://doi.org/10.1145/2380116.2380176

[18] Scott R. Klemmer, Jack Li, James Lin, and James A. Landay. 2004. Papier-Mache: Toolkit Support for Tangible Input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. https://doi.org/10.1145/985692.985743

[19] David Ledo, Steven Houben, Jo Vermeulen, Nicolai Marquardt, Lora Oehlberg, and Saul Greenberg. 2018. Evaluation Strategies for HCI Toolkit Research. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, Article 36, 17 pages. https://doi.org/10.1145/3173574.3173610

[20] Jessica Lo and Audrey Girouard. 2017. Bendy: Exploring Mobile Gaming with Flexible Devices. In *Proceedings of the Eleventh International Conference on Tangible, Embedded, and Embodied Interaction (TEI '17)*. ACM, New York, NY, USA, 163–172. https://doi.org/10.1145/3024969.3024970

[21] David Merrill, Jeevan Kalanithi, and Pattie Maes. 2007. Siftables: Towards Sensor Network User Interfaces. In *Proceedings of the 1st International Conference on Tangible and Embedded Interaction*. ACM. https://doi.org/10.1145/1226969.1226984

[22] James Patten and Hiroshi Ishii. 2007. Mechanical Constraints As Computational Constraints in Tabletop Tangible Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. https://doi.org/10.1145/1240624.1240746

[23] Raf Ramakers, Davy Vanacken, Kris Luyten, Karin Coninx, and Johannes Schöning. 2012. Carpus: A Non-intrusive User Identification Technique for Interactive Surfaces. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*. ACM. https://doi.org/10.1145/2380116.2380123

[24] Christophe Scholliers, Lode Hoste, Beat Signer, and Wolfgang De Meuter. 2011. Midas: A Declarative Multi-Touch Interaction Framework. In *Proceedings of the Fifth International Conference on Tangible, Embedded, and Embodied Interaction (TEI '11)*. ACM, New York, NY, USA, 49–56. https://doi.org/10.1145/1935701.1935712

[25] Brygg Ullmer and Hiroshi Ishii. 1999. mediaBlocks: Tangible Interfaces for Online Media. In *CHI '99 Extended Abstracts on Human Factors in Computing Systems*. ACM. https://doi.org/10.1145/632716.632739

[26] Brygg Ullmer and Hiroshi Ishii. 2000. Emerging frameworks for tangible user interfaces. *IBM systems journal* (2000).

[27] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O. Wobbrock. 2012. Gestures As Point Clouds: A $P Recognizer for User Interface Prototypes. In *Proceedings of the 14th ACM International Conference on Multimodal Interaction (ICMI '12)*. ACM, New York, NY, USA, 273–280. https://doi.org/10.1145/2388676.2388732

[28] Malte Weiss, Julie Wagner, Yvonne Jansen, Roger Jennings, Ramsin Khoshabeh, James D. Hollan, and Jan Borchers. 2009. SLAP Widgets: Bridging the Gap Between Virtual and Physical Controls on Tabletops. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. https://doi.org/10.1145/1518701.1518779

[29] Jacob O. Wobbrock, Andrew D. Wilson, and Yang Li. 2007. Gestures Without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology (UIST '07)*. ACM, New York, NY, USA, 159–168. https://doi.org/10.1145/1294211.1294238

Received February 2018; revised April 2018; accepted June 2018.

## A TUIO 2.0 PROTOCOL SPECIFICATION

This appendix presents the detailed specification and individual messages which compose the TUIO 2.0 protocol. For a high-level comparison to TUIO1, please refer to section 5. We assume some familiarity of the reader with the previously defined TUIO1.1 specification (see also https://tuio.org/?specification).

### A.1 Message Structure

TUIO 2.0 defines a unified profile for all previously covered tangible object types, such as tokens (tagged objects), pointers and geometries (for untagged generic objects). The idea of *ALIVE messages* and *FSEQ messages* of the original TUIO specification was generally maintained within `/tuio2/frm` and `/tuio2/alv` messages, while the *SET messages* of the previous 2Dobj, 2Dcur and 2Dblb profiles, were mapped to individual messages within the same `/tuio2/*` name space. Therefore the OSC name space was reduced to an abbreviated, but hopefully still human-readable structure and was also adapted to a more OSC compliant message and bundle style.

As within TUIO 1.1, a Session ID is an unique identifier for each interface component, which is maintained over its life-time, starting with its appearance, and maintained with every update until its removal. This not only allows the explicit distinction of otherwise untagged pointers for example, but also the multiply use of tokens that are tagged with the same symbol type. Furthermore the unified TUIO 2.0 profile structure, now also allows the cross referencing of various components through their common Session ID. This allows the association of a control to a pointer for example, as well as adding additional geometries to a token, including an incremental increase of detail if necessary.

The distribution of SET messages over different profiles such as 2Dobj, 2Dcur and 2Dblb, has been reduced to dedicated TOK, PTR and BND messages that transmit the status updates for tokens (objects), pointers (cursors) and bounds (blobs). The component geometry can be described in greater detail with additional geometry messages such as OCG (contour) and SKG (skeleton). The new SYM messages allow the transmission of symbol content, and CTL messages allow the association of additional control dimensions to existing components. A set of associative messages allow the encoding of container relationships (COA) as well as physical or logical links (LIA) between tangible objects. Custom messages that meet the requirements of yet undefined trackers now also can be realized within the same name space. This allows the usage of the same session ID across the same surface profile for alternative token, pointer or geometry references to the same tangible object. The specification primarily defines a profile for two-dimensional surfaces, which is partially extended to the 3rd dimension by complementary 3D component messages for tokens, pointers and bounds. Therefore TUIO was designed as a semantic description of tangible interfaces components within the confines of an interactive surface environment and the space expanding above that surface.

**Please note:** The following textual representation illustrates the syntax of the individual TUIO messages. An actual OSC implementation encodes these messages using binary data types, as specified in the attribute type table further below. You can also refer to the OSC specification for detailed information regarding the basic data types and overall message encoding.

### A.2 Global Messages

#### *A.2.1 FRM (frame message).*

```
/tuio2/frm f_id time dim source
/tuio2/frm int32 ttag int32 string
```

The FRM message is a unique identifier for an individual frame of sensor measurements, and therefore has to be included at the beginning of each TUIO bundle. Each frame is identified by a 32bit unsigned integer value that represents an incrementing frame ID. This allows dropping messages that arrive late with an eventually lower frame ID, the frame ID 0 is reserved to forced state updates. The following time stamp is represented as an OSC 64bit time tag. Although the key component messages may include redundant speed and acceleration attributes that compensate for possibly lost packages, the availability of dedicated timing information is essential for many gesture-based interfaces. The dimension attribute encodes the sensor dimension with two

16bit unsigned integer values embedded into a 32bit integer value. The first two bytes represent the sensor width, while the final two bytes represent the sensor height. This allows to encode a sensor dimension up to 65535x65535 and implicitly also describes the surface ratio as well as its relative resolution. (In an optional 3D fishtank scenario - or the pointer hovering state - the optional Z-axis values are relative to the sensor height.)

The source string attribute provides additional information about the origin of the TUIO message bundle. This string intends to provide a unique identification string for each TUIO source, which follows the following format convention:

`src_name:src_instance@src_origin`, where the `src_name` is provided as a unique and reasonably short identifier string for a TUIO server application, the `src_instance` numbers the various possible instances, while the `src_origin` encodes the machine address in HEX format, depending on its origin. So a single reacTIVision instance running on localhost could be identified by a source string in the form of "REAC", "REAC:0" or `REAC:0@0x7F000001` depending on its context. The TUIO2 client implementation needs to consider the correct decoding of the source string detail level.

#### *A.2.2 ALV (alive message).*

```
/tuio2/alv s_id0 ... s_idN
/tuio2/alv int32... int32
```

The end of each bundle is marked with the ALV message containing a list of all active session IDs, which allows the robust reconstruction of added or removed TUIO components. This is more robust than the possible usage of dedicated ADD or DEL messages, which can cause inconsistencies when lost on a transport channel such as UDP. Added objects can also be derived from their first appearance in a TOK, PTR or BND message. The session ID attribute is encoded using a 32bit unsigned integer value allowing a possible value range between 0 ... 4.294.967.295 until overflow, which should not cause any negative effects in a typical session. Since OSC only defines a default 32bit signed integer field, a TUIO implementation needs to cast the s_ID attribute to uint32 during the encoding/decoding step.

### A.3 Component Messages

#### *A.3.1 TOK (token message).*

```
/tuio2/tok s_id tu_id c_id x_pos y_pos angle [x_vel y_vel a_vel m_acc
    r_acc]
/tuio2/tok int32 int32 int32 float float float [float float float float
    float]
```

The TOK message is the equivalent to the 2Dobj SET message of the TUIO 1.* specification, which encodes the common attributes of tagged physical objects. Tokens are generally identified through their Component ID, which corresponds to a visual marker or embedded RFID tag for example. Several Tokens can have the same Component ID, and can be only distinguished through their unique Session ID, which is only assigned throughout its current appearance within a session. The Session ID (s_id) and Component ID (c_id) as well as the general X & Y position and angle attributes remain unchanged, while a combined Type/User ID (tu_id) allows the multiplexing of various symbol types within the same session as well as the association of an additional user ID. The first two bytes of the type/user attribute are therefore encoding the User ID, while the second half of the attribute encode the actual Type ID resulting in two 16bit unsigned integer values. This allows a possible range of 65535 Type and User IDs. The User ID can be used to determine if a token is currently being held by a user, therefore the ID 0 is reserved for the "no user" state. A TUIO implementation has to consider this special usage of the int32 tu_id attribute with an according encoding/decoding step. Speed and acceleration parameters are optional and the client implementation has to consider the two possible message lengths. There is yet no predefined list of fixed Type IDs for Tokens, but its details can be specified through an additional Symbol message.

#### *A.3.2 PTR (pointer message).*

```
/tuio2/ptr s_id tu_id c_id x_pos y_pos angle shear radius press [x_vel
    y_vel p_vel m_acc p_acc]
/tuio2/ptr int32 int32 int32 float float float float float [float float
    float float float]
```

The PTR (pointer) message is equivalent to the 2Dcur SET message of the TUIO 1.* specification, which encodes the common attributes of pointing gestures. The message syntax changed significantly compared to the original profile, in addition to the Session ID and its X & Y position, it also defines a Component ID that allows the distinction or tagging of individual pointer components (e.g. fingers). The provided angle attributes specify the Pointer's rotation angle as well as the shear angle relative to the horizontal surface plane. An additional BND message can be used to specify a more detailed geometry of the pointer area.

The radius attribute indicates a pointer's "region of influence" by specifying its action radius (encoded normalized to the sensor height). An additional pressure value in the range from 0..1 was added for the encoding of discrete or continuous surface pressure. Additionally a negative pressure value can be used to indicate a pointer that is not in touch with the surface, and therefore in a hovering state.

The Type ID attribute allows the distinction of different pointer types (eg. fingers, mouse or pen) and is also used to encode the associated User ID. The first two bytes of the type attribute are therefore reserved for the User ID, while the second half of the attribute encode the actual Type ID resulting in two 16bit unsigned integer values. This allows a possible range of 65535 user IDs and type IDs. A TUIO implementation has to consider this special usage of the int32 tu_id attribute with an according encoding/decoding step. TUIO2 defines a list of default Pointer Type IDs, where the ID 0 stands for an undefined or unknown pointer. The IDs 1-5 define fingers of the right hand starting with the index finger (index, middle, ring, little, thumb) followed by same sequence from ID 6-10 for the left hand. The default ID for an unknown finger is the right index finger ID 1. The ID range from 11-20 defines a small selection of common pointer devices (11 stylus, 12 laser pointer, 13 mouse, 14 trackball, 15 joystick, 16 remote). The ID range from 21-30 defines various body parts (21 right hand pointing, 22 right hand open, 23 right hand closed, 24 left hand pointing, 25 left hand open, 26 left hand closed, 27 right foot, 28 left foot, 29 head, 30 person). Any Type ID starting from 64 and above can be freely associated by the tracker implementation. Speed and acceleration parameters are optional and the client implementation has to consider the two possible message lengths. Please note that the pointer also encodes the pressure velocity and acceleration, which can be used for dynamic input.

#### *A.3.3 BND (bounds message).*

```
/tuio2/bnd s_id x_pos y_pos angle width height area [x_vel y_vel a_vel
    m_acc r_acc]
/tuio2/bnd int32 float float float float float float [float float float
    float float]
```

The BND message is the equivalent to the 2Dblb SET message of the TUIO 1.1 specification, which encodes the basic geometry information of untagged generic objects (blobs). The message format describes the inner ellipse of an oriented bounding box, with its centre point, the angle of the major axis, the dimensions of the major and minor axis as well as the region area. Therefore this compact format carries information about the approximate elliptical region enclosure, but also allows the reconstruction of the oriented bounding box. The region area is normalized in pixels/width*height, providing quick access to the overall region size.

The BND message usually identifies the boundaries of any generic untagged physical object, and can be also used to transmit the basic geometry information such as the angle and dimensions of finger blobs or physical tokens that have been already identified by a previous PTR or TOK message. The session ID has to be equal in both messages in order to match the component with the corresponding bounds.

#### *A.3.4 SYM (symbol message).*

```
/tuio2/sym s_id tu_id c_id group data
/tuio2/sym int32 int32 int32 string string
```

The SYM message allows the transmission of the type and data contents of a marker symbol. Since this information can be redundant, and does not necessarily apply to all symbol types, it is represented by a dedicated message, which can be omitted or sent at a lower rate if desired. The Session ID, Type/User ID and Component ID are identical to the values used in the corresponding TOK message. Therefore the actual symbol code and the meta-information about the marker type and symbol description only needs to be received once by the client. The group attribute is a string describing the symbol type, such as fiducial markers, barcodes, or RFID tags. The code attribute is alternatively an OSC string or an OSC blob data field that transmits the symbol code or contents: such as the libfidtrack left heavy depth sequence, an EAN barcode number, or an RFID UID. Since the possibly symbol space may often exceed the range of component IDs, a TUIO implementation needs to maintain its internal mapping of Symbols to Component IDs. In case a TUIO tracker such as an RFID reader, is not capable to determine the symbol position or orientation, the SYM message can be sent individually without any association to a previous TOK component.

| /tuio2/sym s_id **tu_id c_id grp dat** | description |
|---|---|
| 0 2 fidtrk/18 0122212221221221111 | libfidtrack 18-node |
| 1 8 fidtrk/12 0122121211111 | libfidtrack 12-node |
| 2 0 mifare/ul 0x04c5aa51962280 | mifare ultralight RFID |
| 3 1 mifare/1k 0x0af55f2a | mifare classic 1K RFID |
| 4 0 qr/url http://www.tuio.org/ | URL QR-code |
| 5 4 ean/13 5901234123457 | EAN bar-code |
| 6 18 ms/byte 0x12 | MS byte tag |
| 7 255 color/rgb 0x0000FF | RGB color tag (blue) |

#### *A.3.5 T3D (token 3D message).*

```
/tuio2/t3d s_id tu_id c_id x_pos y_pos z_pos angle x_ax y_ax z_ax [x_vel
    y_vel z_vel r_vel m_acc r_acc]
/tuio2/t3d int32 int32 int32 float float float float float float float [
    float float float float float float]
```

The T3D message encodes an alternative 3D representation for tokens that are used within the space that extends above the surface. The message includes an additional Z coordinate as well as the rotation axis and angle. The optional velocity attributes also include these additional dimensions.

#### *A.3.6 P3D (pointer 3D message).*

```
/tuio2/p3d s_id tu_id c_id x_pos y_pos z_pos x_ax y_ax z_ax radius [x_vel
     y_vel z_vel r_vel m_acc r_acc]
/tuio2/p3d int32 int32 int32 float float float float float float [float
    float float float float float]
```

The P3D message encodes an alternative 3D representation for pointers that are used within the space that extends above the surface. The message includes an additional Z coordinate as well as vector of the pointing direction. The radius attribute refers to the spherical region of influence of the 3D pointer (encoded normalized to the sensor height).

#### *A.3.7 B3D (bounds 3D message).*

```
/tuio2/b3d s_id x_pos y_pos z_pos angle x_ax y_ax z_ax width height depth
     volume [x_vel y_vel z_vel r_vel m_acc r_acc]
/tuio2/b3d int32 float float float float float float float float float float
    float float [float float float float float float float]
```

The B3D message encodes an alternative 3D representation for untagged components that are used within the space that extends above the surface. The message includes an additional Z coordinate and the according depth attribute as well as the full 3D orientation axis and angle. The optional velocity attributes also include these additional dimensions.

### A.4 Geometry Messages

The following list of CHG, OCG, ICG, SKG, SVG and ARG messages are optional descriptors of the component geometry, which can be incrementally describe the contour, skeleton and full area of the referenced component in various levels of detail. The RAW message allows in conjunction with an ARG message the full reconstruction of a bitmap that corresponds to the raw sensor data.

#### *A.4.1 CHG (convex hull geometry).*

```
/tuio2/chg s_id x_p0 y_p0 ... x_pN y_pN
```

The CHG message is a list of points that define the simplified convex hull of the blob. This means that the number of points has to be reduced to a reasonable amount, which represents the original hull with a minimum error. The client implementations have to take the variable length of this message into account.

#### *A.4.2 OCG (outer contour geometry).*

```
/tuio2/ocg s_id x_p0 y_p0 ... x_pN y_pN
```

The OCG message is a list of points that define the simplified outer contour of the blob. This means that the number of points has to be reduced to a reasonable amount, which represents the original contour with a minimum error. The client implementations have to take the variable length of this message into account.

#### *A.4.3 ICG (inner contour geometry).*

```
/tuio2/icg s_id x_p0 y_p0 ... x_pN y_pN true x_p0 x_p0 ... x_pN y_pN
```

The ICG message additionally defines the points of interior contour lines. According to the OCG message, a ring is only described as a disk, while the ICG message encodes the additional inner contour needed to reconstruct its full shape. The inner contour is a list of points that uses the Boolean value TRUE to indicate the beginning of a separate contour. A graphical representation of the number eight therefore would contain two inner contour sequences for example.

#### *A.4.4 SKG (skeleton geometry).*

```
/tuio2/skg s_id x_p0 y_p0 x_p1 y_p1 node ... x_pN y_pN
```

The SKG message represents the skeleton structure of a blob. In contrary to the list of contour points this needs to be represented as a tree structure. After the session ID the message begins with an arbitrary leaf of that tree structure and continues the point list until it reaches the next leaf point. The integer node number directs the tree back to the last node point.

*A.4.5 S3D (skeleton 3D geometry).*

```
/tuio2/s3d s_id x_p0 y_p0 z_p0 x_p1 y_p1 z_p1 node ... x_pN y_pN z_pN
```

The S3D message represents the three dimensional skeleton structure of a blob. Apart from an additional Z-coordinate for each node, this message follows the same syntax as the standard skeleton geometry message.

*A.4.6 SVG (skeleton volume geometry).*

```
/tuio2/svg s_id r0 ... rN
```

The SVG message adds the radius to each skeleton point as defined by the SKG message. This allows an approximate reconstruction of the blob volume (encoded normalized to the sensor height) based on the skeleton, without the need of the more detailed contour description. This message is based on information from the SKG message and can therefore only be used meaningfully after decoding a previous SKG or S3D message.

*A.4.7 ARG (area geometry).*

```
/tuio2/arg s_id x0 y0 w0 ... xN yN wN
```

The ARG message is the most detailed shape description message and describes the full blob area as a list of spans. This is basically a run-length encoding with an initial span point and the following span length. The span list allows the complete reconstruction of the region area and its exact contour information.

*A.4.8 RAW (raw message).*

```
/tuio2/raw s_id width data
```

This final RAW region descriptor provides additional 8bits of data resolution for each point of the region as referenced by a previous ARG (area) message. The data attribute is an OSC blob field with a length according to the amount of region points, where the individual samples are represented by an 8bit unsigned integer value. The actual decoding of this message has to follow the order of the previous span list, which therefore only covers the discrete region point samples. The width attribute specifies the normalized distance between two points in order to correctly reconstruct the actual number of samples between the initial and final span point. Although this value can be also retrieved from the surface dimension attribute (1/samplewidth) of the preceding FRM message, it is included here for the convenience of rapid access. The RAW message allows for example the transmission of gray-scale image data from a computer vision sensor or pressure maps that have been retrieved from an according pressure sensitive device.

### A.5 Content Messages

*A.5.1 CTL (control message).*

```
/tuio2/ctl s_id c0 ... cN
/tuio2/ctl int32 bool/float ... bool/float
```

The CTL message can be used to transmit additional control dimensions that can be associated to an existing component instance, such as a token with an integrated temperature sensor. This (open length) list of variable float or boolean values, encodes each individual control dimension as discrete 0/1 boolean or continuous floats in the normalized range from -1.0f ... 1.0f. A simple 3-button wheel mouse for example could be encoded using a CTL message with three boolean values for the discrete buttons and one additional float value for the continuous wheel after the initial session ID.

An array of 12 float attributes can for example encode the keys of a full octave in a small piano keyboard including key velocity. The association of the according CTL message to a previous TKN consequently allows the identification and localization of that physical keyboard component.

*A.5.2 DAT (data message).*

```
/tuio2/dat s_id mime data
/tuio2/dat s_id string string/blob
```

The DAT message allows the association of arbitrary data content to any present TUIO component. Apart from the common session ID, this message only contains an initial OSC string that defines the MIME type of the following data attribute, which can be either transmitted using an OSC string or OSC blob data type. Therefore this message is capable of encoding and transmitting textural or binary data such as business cards, XML data, images or sounds etc. Being a simplified version of Symbols, a DAT message can be for example also used to transmit the actual data content of an RFID tag that has been referenced within a previous SYM message. Due to the likely limited bandwidth resources of the used OSC channel, this infrastructure is not suitable for the transmission of larger data sets. In this case the use of alternative transport methods is recommended.

| /tuio2/dat s_id **mime data** | content | description |
|---|---|---|
| text/x-vcard OSC_string | ASCII vcard | business card |
| text/html OSC_string | HTML code | HTML text |
| image/x-icon OSC_blob | icon data | windows icon |

*A.5.3 SIG (signal message).*

```
/tuio2/sig s_id c_id s_id0 ... s_idN
```

The SIG message allows the transmission of a trigger signal from a reference component to one or more TUIO components which are specified in the following list of session IDs. The Component ID specifies the type of signal allowing a possible range of 4.294.967.295 signal variations.

### A.6 Association Messages

The following association messages reflect the established links within constructive assemblies as well as container associations within the physical domain. They provide a description of the resulting component relationships, which can be employed to describe mechanical or logical connections as well as the placement of a token within the confines of another physical object.

*A.6.1 ALA (alive associations).*

```
/tuio2/ala s_id0 ... s_idN
```

The initial ALA message lists all active components that are currently in an associated state. This is providing a mechanism in analogy to the ALV message structure, and allows the robust reconstruction of connection and disconnection events.

*A.6.2 COA (container association).*

```
/tuio2/coa s_id slot s_id0 ... s_idN
```

The COA message allows associating one or more components such as several tokens to be contained within another physical component such as an object described by its geometry. Container associations are established by providing lists of one or more associated objects, which also allow the reconstruction of individual association events. The first session ID specifies the container object with a following variable length list of the associated object session IDs. It also allows nested container relationships, which can be encoded using various subsequent container messages. The provided slot attribute determines the entry point of the associated components within the container.

#### *A.6.3 LIA (link association).*

```
/tuio2/lia s_id bool s_id0 l_id0 ... s_idN l_idN
```

The LIA message is used for describing the topologies of constructive assemblies comprised of physical objects that allow the establishment of direct mechanical connections between them. The explicit declaration of physical object connections can for example be employed for environments, which are not capable of reporting spatial object relations. Additionally these connector associations can be used to encode collisions of physical objects, without the need for the additional transmission of the detailed object geometries for later collision detection.The initial session ID specifies the reference object with a following variable length list of a ID tuple that lists all component session IDs that are connected to the reference component as well as a coupled Link ID which identifies the input and output ports. This link attribute is comprised of two 16bit unsigned integer values embedded into a single 32bit integer value, which specify the output port within the initial two bytes and the input port of the connected component within the last two bytes. Alternatively the link association can also be used to establish logical connections between individual components, the provided boolean value determines if the association is physical (true) or logical (false).

#### *A.6.4 LLA (linked list association), LTA (linked tree association).*

```
/tuio2/lla s_id type s_id0 l_id0 ... s_idN l_idN
/tuio2/lla int32 bool int32 int32 ... int32 int32

/tuio2/lta s_id type s_id0 l_id0 s_id1 l_id1 node ... s_idN l_idN
/tuio2/lta s_id bool int32 int32 int32 int32 float ... int32 int32
```

These two additional messages allow the encoding of connections between several interface components using linked lists, tree structures or individual connection lists for the encoding of more complex topologies. The LLA (linked list association) message encodes consecutive chains of connected objects, while the LTA (linked tree association) message encodes tree structures in a format similar to the SKG (skeleton geometry) described above. The initial boolean type value determines if the association is physical (true) or logical (false). The Session ID and Link ID tuples are structured as defined in the LIA (link association) message specified above. The LTA node jump simply uses the float data type in order to allow its distinction from the related Session ID references.

### A.7 Custom Messages

```
/tuio2/_[attr] s_id [list of attributes]
/tuio2/_sxyPP s_id x_pos y_pos int float
```

The custom profile allows the transmission of custom shaped messages that can change the position, omit or add attributes as desired. The message name is composed out of an initial underscore character that indicates a custom message. The following characters can be freely chosen from the list of know attributes as shown in the table below. Additional undefined parameters can be added using wildcard P character in the profile name. The actual parameter type is defined by the OSC message itself. The purpose of the custom message within the TUIO specification is to allow the TUIO clients at least a partial decoding of custom messages. Since TUIO is based on OSC a TUIO source implementation also can choose to add undocumented freely named and formatted messages, which then of course cannot be decoded by any standard TUIO client. Therefore the custom message should be chosen if any known attributes are transmitted, be it only the common session ID. This at least allows the client to associate the unknown message to a known symbol.

```
/tuiox/[ext] s_id [list of attributes]
```

It is important that all implementations within the dedicated `/tuio2/` name space follow the exact message syntax and attribute format as defined in this specification. Any modification of the existing message syntax could lead to a fragmentation of the protocol specification and will also cause unstable results with standard

client implementations. Although TUIO 2.0 intends to provide a versatile mechanism for the encoding of a large variety of tangible interaction platforms, it is also probable that some individual application scenarios or hardware platforms might requite an extension to the protocol. In case the custom message structure described above might not meet the semantic needs of such an extension, the usage of a separate `/tuiox/` name space is suggested for the definition of message extensions that are following the general TUIO paradigm. This most importantly includes the shared reference to a common Session ID. Custom client implementations can therefore be configured to listen to one or more custom message extensions, while standard client implementations remain unharmed.

Message extensions could for example encode the attributes of dedicated controller devices such as the Wiimote (e.g. /tuiox/wii) or the complementary description of the user's hand (e.g. /tuiox/hnd). TUIO 2.0 does not define the three-dimensional geometry of interface components, which also could be implemented within several /tuiox/ messages if required. This extension space can also serve as a staging platform for the future inclusion into subsequent versions of the standard protocol specification. For all other custom messages that are not following the general structure of the TUIO protocol, the usage of a completely separate OSC name space is recommended though.

### A.8 Timing Model

As mentioned above, the TUIO 1.* specification originally did not contain any explicit timing information. On the contrary TUIO 2.0 transmits a time code for each bundle at a resolution smaller than microseconds by including a dedicated OSC time tag attribute with each FRM (frame) message. This level of time resolution provides enough information for time based gesture analysis on the client side.

Because of the possible packet loss, the key component messages TOK, PTR and BND still define the optional speed and acceleration attributes. In analogy to the normalized coordinate system, TUIO2 defines a velocity vector and a simple method how to calculate the resulting velocity and acceleration values correctly. The movement velocity unit is defined by the displacement over the full length of the axis (also normalized to 0-1) per second. As an example, moving a finger horizontally across the whole surface within one second, results in a movement velocity of (1.0 0.0) The rotation velocity unit is defined as one full rotation per second. Therefore as an example, performing one object rotation within two seconds, results in a rotation velocity of 0.5. The acceleration values are then simply calculated as a function of speed changes over time (in seconds).

### A.9 Bundle Structure

While the bundle structure in the original TUIO definition was mostly used to take most advantage of the usually used UDP packet size, TUIO2 proposes a more structured use of OSC bundles in order to allow more TUIO2 client implementations on generic OSC platforms.

The previous TUIO 1.1 message structure used a single profile name space for all messages within a bundle, with only the first attribute as command, which eventually caused filtering problems with some OSC implementations. TUIO2 already takes this into account with its newly defined message structure, where the unified name space already includes the command and it also defines some simple rules for the internal order of each OSC bundle.

Each bundle needs to contain at least a FRM and a ALV message, where the FRM message is the first message in a bundle, while the ALV message concludes the bundle. As proposed in the original TUIO 1.0 specification, an implementation should take advantage of the full UDP packet size by creating message bundles that fit into the available packet space. Eventually free packet space can be filled with redundant object messages that can be periodically resent. It is up to the client implementation to identify and filter redundant messages.

The following sequence of TUIO 2.0 messages illustrates an example bundle encoding two active interface components including a tangible token with an associated symbol and a finger pointer with an associated basic geometry. The full bundle is embedded within an initial frame message and the concluding alive message. Please note that the alive message also contains an additional reference to another active component (with session ID 11) that has not been updated within this frame.

**initial frame message**

```
/tuio2/frm 1236 {OSC time tag} {640x480} REAC:0@0x7F000001
```

**component messages**

```
/tuio2/tok 10 0 4 0.460938 0.3375 1.57
/tuio2/sym 10 0 4 fidtrk/18 0122222212221211111
/tuio2/ptr 12 1 0 0.525 0.3 0.05 0.0 0.1 1.0
/tuio2/bnd 12 0.525 0.3 0.05 0.1 0.15 0.015
```

**concluding alive message**

```
/tuio2/alv 10 11 12
```

### A.10 Server & Client implementations

A TUIO server (sometimes also referred as TUIO tracker or TUIO source) is a device or application that encodes and sends TUIO messages based on the OSC format, while a TUIO client is an application or library that receives and decodes these messages, providing the basic infrastructure for an actual interactive application. This is actually the exact opposite of the OSC naming convention, where an OSC client is sending its messages to an OSC server, which usually means that a controller device (client) is attached to a synthesizer (server). Although this difference might cause some confusion, the present definition of TUIO servers and clients is more adequate for describing the direction of the data flow from the tangible interaction hardware to the application layer.

For speed and simplicity reasons, the TUIO protocol is generally unidirectional, which means that there is currently no dedicated communication channel from the client to the server necessary. Using the UDP transport for example, a TUIO server usually sends its messages to a single TUIO client, which can be running on the same platform as the server (localhost) as well as on a local IP network (LAN) or even at a distant location via the Internet (WAN). Nevertheless the present TUIO protocol could be equally implemented in a bi-directional configuration, where the application layer is sending standard TUIO messages to a tracker platform that is equipped with actuated components. In such a configuration TOK messages could be used for example to move physical objects or to drive actuated elements such as motors or lights with a sequence of according CTL messages.

A TUIO Server will usually encode and send messages for TUIO components that correspond to its general capabilities, therefore a server implementation can also choose to support only a subset of the possible TUIO components. Apart from the compulsory FRM and ALV messages, which comprise the basic body of a TUIO bundle, it depends on the server capabilities or configuration, which types of component messages are actually chosen to be sent. On the other hand a typical TUIO client implementation, especially if designed as a library component, should be ideally capable of decoding the full set of interface component messages as defined in this specification. TUIO 2.0 server and client reference implementations will be provided for the most common programming languages, such as C++, Java and C#. Beta versions are already available on Github at anonymous. These examples can be directly used as a library for the realization of TUIO enabled applications as well as a reference for the implementation for further programming environments. And since TUIO is based upon the OSC specification any platform that already provides an OSC infrastructure, is consequentially also able to send and receive TUIO messages.

### A.11 Transport method

The default transport method for the TUIO protocol is the encapsulation of the binary OSC bundle data within UDP packets sent to the default port 3333. This default transport method is usually referred as TUIO/UDP, and most implementations are based on this method due to its simplicity and speed when sent over a network. Since OSC is not directly bound to a dedicated transport method, alternative transport channels such as TCP can be employed to transmit the OSC encoded TUIO data. As introduced with the TUIO 1.1 implementations, there are already several alternative transport methods available, such as TUIO/TCP and TUIO/FLC (flash local connection via shared memory) to interface with Adobe Flash applications. Since Flash is approaching the end of its life cycle, an alternative TUIO/WEB transport option has been introduced, which establishes a standard

Websocket for the realization of native HTML5 applications. Due to the OSC encoding, TUIO messages can be basically transmitted through any channel that is supported by an actual OSC implementation.

A UDP packet can carry a maximum of 64kb and a minimum of 576 bytes, which usually provides enough capacity to transmit a typical TUIO bundle within a single UDP packet. When sent over a local or wide area network it is also advisable to limit the UDP packet size to the MTU size (usually about 1500 bytes) in order to avoid packet fragmentation. Therefore a TUIO server implementation has to consider that bundles containing larger component sets can eventually exceed the UDP packet capacity, and consequently need to distribute the component messages over several OSC bundles containing the same initial FRM message, while only the last bundle is concluded with a final ALV message.