AN ABSTRACTION FRAMEWORK FOR TANGIBLE INTERACTIVE SURFACES

MARTIN KALTENBRUNNER

Cumulative Doctoral Dissertation

Doktor-Ingenieur (Dr.-Ing.)
Fakultät Medien
Bauhaus-Universität Weimar

Supervisor: Prof. Dr. Florian Echtler
Internal Reviewer: Prof. Dr. Eva Hornecker
External Reviewer: Prof. Dr. Johannes Schöning

20. November 2017 – version 1.0.1

Martin Kaltenbrunner:
An Abstraction Framework for Tangible Interactive Surfaces
Cumulative Doctoral Dissertation, Doktor-Ingenieur (Dr.-Ing.),
CC BY-NC-ND 20. November 2017

I don't care to belong to any club
that will have me as a member.

— Groucho Marx

Dedicated to the memory of Engelbert Maria Dorfinger

1954 – 2001

ABSTRACT

This cumulative dissertation discusses - by the example of four subsequent publications - the various layers of a tangible interaction framework, which has been developed in conjunction with an electronic musical instrument with a tabletop tangible user interface. Based on the experiences that have been collected during the design and implementation of that particular musical application, this research mainly concentrates on the definition of a general-purpose abstraction model for the encapsulation of physical interface components that are commonly employed in the context of an interactive surface environment. Along with a detailed description of the underlying abstraction model, this dissertation also describes an actual implementation in the form of a detailed protocol syntax, which constitutes the common element of a distributed architecture for the construction of surface-based tangible user interfaces. The initial implementation of the presented abstraction model within an actual application toolkit is comprised of the *TUIO protocol* and the related computer-vision based object and multi-touch tracking software *reacTIVision*, along with its principal application within the *Reactable* synthesizer. The dissertation concludes with an evaluation and extension of the initial TUIO model, by presenting TUIO2 - a next generation abstraction model designed for a more comprehensive range of tangible interaction platforms and related application scenarios.

ZUSAMMENFASSUNG

Diese kumulative Dissertation diskutiert - am Beispiel vier aufeinanderfolgender Publikationen - die einzelnen Schichten eines gegenständlichen Interaktionsmodells, das im Zusammenhang mit einem elektronischen Musikinstrument mit gegenständlicher Benutzerschnittstelle entwickelt wurde. Basierend auf den Erfahrungen die während der Gestaltung und Implementierung dieser konkreten musikalischen Anwendung gesammelt wurden, konzentriert sich diese Forschung hauptsächlich auf die Definition eines generell einsetzbaren Abstraktionsmodells für die digitale Repräsentation physischer Interfacekomponenten welche üblicherweise im Kontext interaktiver Oberflächen verwendet werden. Gemeinsam mit einer detaillierten Beschreibung des zugrundeliegenden Abstraktionsmodells, behandelt diese Dissertation auch dessen konkrete Implementierung in Form einer detaillierten Protokollsyntax, die das verbindende Element einer verteilten Architektur für die Realisierung von oberflächenbasierten gegenständlichen Benutzerschnittstellen darstellt. Die eigentliche Implementierung des vorgestellten Abstraktionsmodells als konkretes Toolkit besteht aus dem *TUIO Protokoll* und der damit verbundenen Computervision Anwendung für Objekt- und Fingergestentracking *reacTIVision*, gemeinsam mit deren primären Anwendung in der Realisierung des *Reactable* Synthesizers. Die Dissertation schliesst mit einer Evaluierung und Erweiterung des ursprünglichen TUIO Modells, mit der Präsentation von TUIO2 - einem Abstraktionsmodell der nächsten Generation, das für eine umfassendere Reihe von gegenständlichen Interaktionsplattformen und die dazugehörigen Anwendungsszenarien entworfen wurde.

ACKNOWLEDGMENTS

The research results presented within this thesis have been fundamentally motivated by the development of the Reactable, which took place at the Music Technology Group[1] at the University Pompeu Fabra in Barcelona, Spain during the years from 2003 until 2009. The project had been initially conceived by **Sergi Jordà** and has been realized in collaboration with my colleagues **Günter Geiger** and **Marcos Alonso**. The development of the reacTIVision framework would not have been possible without the collaboration of **Ross Bencina** and the initial contribution by **Enrico Costanza**, who both also participated together with **Till Bovermann** in the definition process of the TUIO protocol. I'd like to thank my co-authors for their professional support and friendship during these exciting years of research and development. Furthermore I'd also like to thank **Xavier Serra** for supporting this research as the director of the Music Technology Group as well as **Sile O'Modhrain** for introducing me to the field of Tangible User Interfaces during my research at the Media Lab Europe in Dublin. Finally I'd also like to thank the participants of the TUIO Hackathon at the ITS 2014 conference for their valuable feedback towards the finalization of the TUIO 2.0 specification.

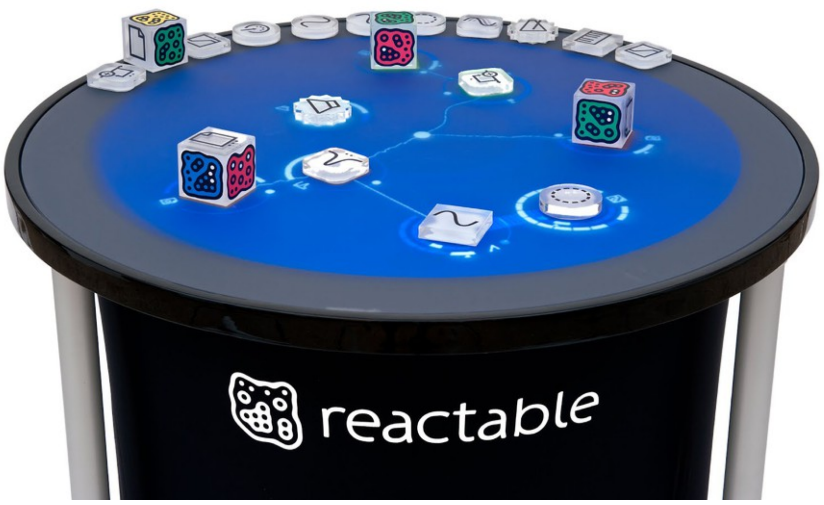

Figure 1: The Reactable by Jordà, Geiger, Alonso and Kaltenbrunner.
Photo: Reactable Systems

I am also grateful to my supervisors **Florian Echtler** and **Eva Hornecker** who supported me during the finalization of my dissertation at the Bauhaus University Weimar.

1 http://mtg.upf.edu/

CONTENTS

Part I

INTRODUCTION

1 THESIS STATEMENT

1.1 RESEARCH THESIS

This cumulative dissertation documents the research process and the related design and development work to support the following thesis:

> *The introduction of an intermediate abstraction layer defining a semantic classification and state encapsulation of interface components that are employed within a tangible interaction environment, is supporting the sensor and application independent design of table-based tangible user interfaces, by providing an infrastructure for the integration of low-level interface technologies with high-level application specific models.*

1.2 RESEARCH CONTEXT

Today the tangible interaction paradigm already represents a relatively well developed research area, where several application models have been established[1] to describe the fundamental design principles that emerged from the research and development practice during the past decades. Most of these models are concentrating on the description of common design patterns or semantic classifications of the function, content or relation of physical design elements at the application layer from a user perspective.

Since tangible user interfaces by definition[2] couple the physical world with digital information, most current approaches therefore also conceptually unify the definition of tangible interface components within the physical domain and their related application logic. Although from a designer perspective these models provide a sound theoretical foundation for tangible interface design through their conceptual embodiment of digital information within physical artifacts, they generally mask the involved technical background, which is necessary for the realization of the complex physical interface hardware.

On the other hand, prospective developers of a tangible user interface can take advantage of a growing pool of openly available hardware and software toolkits, which can be employed for the construction of the manifold physical manifestations of tangible interface components. These tools include computer-vision software, sensor and actuator devices, micro-controller boards and consumer electronics with advanced sensor technologies, which provide the necessary control input to track and describe the state of the physical interface components and the related user interactions. Since these tools and devices mostly provide raw sensor data, from the developer perspective this process involves the additional task of mapping the retrieved raw input data to the according application layer.

This dissertation intends to establish a model for the semantic description of the types, states and relationships of common tangible user interface components, which can be applied to encapsulate the raw input data of arbitrary interface technology employed for the construction of the physical interface. This abstraction layer establishes a generalized interface description and provides the state and attributes of its tangible and intangible components to the application layer, which consequently can build upon the provided semantics for driving its application model. This model therefore serves as an intermediate glue layer for coupling the physical environment with the digital application model, providing a consistent tangible interface representation from a developer, designer and user perspective.

1.3 research contributions

This research has been conducted in various incremental steps, which include the design, implementation, application and final evaluation of an initial model, leading to the definition of the extended abstraction model presented in the final chapter of this dissertation. The following contributions are the result of these research phases:

- The definition of a basic abstraction model providing a semantic description of the physical interaction environment that serves as a mediator between the physical interface components and the digital application logic, defining a sensor and application invariant classification of physical and gestural interface elements in the context a tangible interactive surface.
- The implementation of the above mentioned abstraction model within the open source TUIO protocol definition and the related application programming interface, defining a dedicated syntax and semantic descriptors for the communication of the identified tangible interface component states to an underlying application layer.
- A hard- and software infrastructure provided by computer-vision based reacTIVision reference platform of the proposed physical abstraction model, which is taking advantage of the actual implementation of the previously established protocol infrastructure.
- The interaction design of the table-based tangible musical instrument Reactable, illustrating the practical implications and usage of the proposed model within the application domain for musical interaction, through the manipulation of sound that is conceptually embodied within physical artifacts.
- The refinement of the initial abstraction model based on the analysis of various additional tangible interaction platforms and its subsequent extension and implementation within the TUIO 2.0 protocol in order to reflect the feature space of state-of-the-art table-based tangible interaction models.

1.4 RESEARCH METHODS

Throughout the development of the presented framework concept, its protocol implementation and reference design I consistently applied an *Open Science* methodology by providing a *Public Domain* protocol specification along with an *Open Source* toolkit to the scientific community, also supporting the activities of creative *Open Design* communities. In this context the term *Public Domain* refers to the royalty-free usage of the TUIO protocol, while the *Open Source* concept provides full access to the source code of its implementation. *Open Design* extends this idea by providing further details on the specific hardware design aspects that are relevant for the actual application of these technologies. This in combination with the formal publication of all relevant scientific results and the according documentation constitutes an elementary *Open Science* toolset for the field of Human Computer Interaction.

This open methodology is additionally facilitated by the modular design of the documented framework, which also allowed for the integration of various community contributions into my own research practice. Furthermore an open science approach generates comparable results that are based on a shared research infrastructure, which allows the evaluation and further improvement of individual research aspects within a common ecosystem. The research practice therefore not only consists of the peer-reviewed publication of scientific results, but in addition to that of the peer-improved publication of their according open-source implementation, which I consider the necessary elements for the establishment of a community supported *open-experiment* situation, also including *crowd-sourced* contributions.

The specific application of the presented framework has been evaluated through a *practice-based* research approach within the specific musical application scenario of the Reactable, a table-based modular synthesizer with a tangible user interface. This also included the employment of *artistic-research* methods, through the integration of composers and performers into the design process. The experiences collected from my own performance practice in combination with the feedback from several artistic collaborators, were integrated into an *iterative design* process for the continuous improvement of both the technical foundations and interaction design concepts of this musical instrument.

The final abstraction framework developed within this thesis is based on the comprehensive analysis and classification of various tangible interactive surfaces and their principle physical components and has been evaluated through the exemplary reciprocal representation of several tangible interface platforms and applications within this model.

2 MOTIVATION

The daily practice of an interaction designer, computer scientist and digital artist is primarily defined by the daily work with computers. While these devices represent the most common and versatile tools of our community, the digital user interface today not only dominates the environment of intellectual and creative workers, but has also become the center of many productive and recreational activities of our industrialized society.

Since the industrial revolution of the 19th century, machines not only have increased our productivity, but since then also dominate our everyday life. Early mechanical machine design was generally driven by technical possibilities and requirements and thus often required specific skills and training from their operators. This was also the case after the advent of digital computers in the middle of the 20th century, which were still largely defined by the mechanical constraints of their early interface design. With the increasing digital complexity and the resulting computing capabilities, the machine interface design radically changed towards a more human-centered approach, resulting in the foundations of human-computer interaction research. Based on the early idea of the "electronic brain" and the connotation of an artificial intelligence, this lead to the development of the first programming languages and the according text-based user interfaces, which nevertheless still required significant expertise from their operators.

The following introduction of screen-based user interfaces along with the according direct manipulation devices, eventually led to the introduction the graphical WIMP interface at Xerox with its digital representation of an office desktop metaphor, which today still dominates the design philosophy of most standard computing platforms. The increasing availability of affordable desktop computers in combination with these more intuitive user interfaces, have fundamentally driven the digital revolution of the late 20th century.

While the ongoing digitalization of many professions, such as print production and architecture, now provided tools and products with higher quality along with an increased productivity, this process also resulted in the loss of many cultural techniques that had previously dominated these sectors. Eventually the digitalization also emerged into the private domain, converging many activities such as media consumption, communication and recreational play into single multipurpose computing devices. This of course also resulted into a virtualization of these professional and recreational activities, and the according reduction of interaction potential to screen-based graphical user interfaces. While this clearly demonstrates the universal potential of these devices, there also exist several sectors, which do not only benefit from this tendency towards a standard virtual interface.

Taking the example of musical instrument design, we can observe a similar digitalization tendency during the past decades. The fundamental design of acoustic musical instruments for centuries had been largely defined by the physical principles of sound production in vibrating bodies, along with the resulting design constraints that defined the material, size and shape of the instrument, along with its according interaction possibilities. The advent of electronic musical instruments in the late 19th century eventually broke these physical design constraints, allowing the generation of virtually any possible sound structure, which also paved the way for completely novel user interfaces such as the gesture-controlled Theremin.

The availability of general purpose desktop and mobile computers also resulted in a radical digitalization of musical instrument design. While these powerful computing devices on the one hand provide nearly endless sonic capabilities, the convergence towards the graphical interface on the other hand also led to a limitation of expressive user interactions. Within our own digital instrument design practice, we therefore intended to combine the digital sound production potential of the digital computer with the primarily physical interface aspects of musical interaction design.

While some trends in human-computer interaction such as virtual reality are pushing towards a further digital virtualization, there are also several fields such as ubiquitous computing, physical computing and most importantly tangible computing, which are focussing on the particular physical design aspects of the human-machine interface. Although these research fields generally intend to maintain the benefits of the digital domain, they rather focus on the representation of digital information within physical artifacts or its embodiment into an everyday environment. These approaches therefore stand for the complete opposite to the ongoing virtualization of our life into digital metaphors, and rather intend to augment our physical activities with the potential of digital technologies.

Although the mentioned fields already look back to several decades of research, they still have not reached the technical and conceptual maturity to perpetrate our everyday life in a similar way as the graphical user interface. Apart from an increased technical complexity of a physical interface compared to a mere graphical representation, today's standard computing platforms also come with a mature set of tools that facilitate the design and implementation of state-of-the-art graphical user interfaces with little effort.

Our motivation for this research was therefore not only the development of novel tangible user interface concepts for the design of digital musical instruments. Since the realization of our instrument already required the bottom-up creation of a complete tangible computing environment, my research also focused on the design of a generalized toolset in order to facilitate the creation of general purpose tangible user interfaces. Apart from the according software infrastructure, this toolset also provides the according theoretical foundations for the definition of tangible interactive surfaces.

3 PLATFORM DEFINITION

This brief overview about the early research on tangible interaction models is focusing on the role of physical objects that are commonly employed as the principal interface components. This intends to summarize various object schemes, application models and design paradigms of tangible interfaces with the goal to narrow down our target platform of tangible interactive surfaces and identify common strategies for the description of generic types, attributes and relationships of physical user interface components in this context.

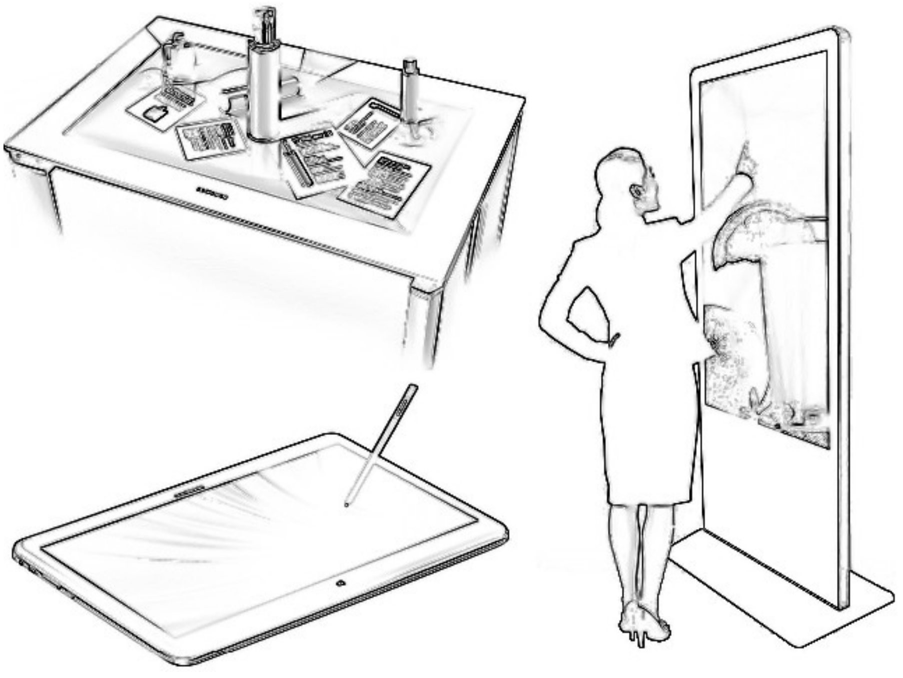

Figure 2: Examples of Interactive Surfaces: horizontal, vertical and mobile

3.1 POST-WIMP INTERFACES

Starting with the first experiments on Graspable User Interfaces by Fitzmaurice et al.[3], the following research lead to the development of several models, which intended to identify the various classes, functions and relationships of tangible objects as physical representations of digital application models. Initially Fitzmaurice introduced graspable design elements, which allowed the user to interact more directly with a virtual interface environment, extending Shneiderman's principle of Direct Manipulation[4] to the physical domain, allowing for direct physical manipulation by using both hands for working with several physical artifacts at once. The design of the physical interface elements within the Bricks interface intended to translate the metaphors from Graphical User Interfaces (GUI) to the physical domain. Physical handles replaced Mouse pointers and the palette tool found its way back into the physical desktop. While the color palette commonly employed in GUI drawing applications already borrowed the tool metaphor from its real world equivalent, the physical drawing palette in the Bricks interface augmented a replication of the original tool with digital means. Ishii's subsequent work on the metaDesk platform[5] continued the direct translation of interface components previously known from the desktop and WIMP metaphor to the physical domain, by creating an interactive desktop environment providing a platform for various physical design elements, such as the passive lens as physical representation of a window, as well as a collection of tangible artifacts named phicons serving as symbolic physical representations of digital data. Furthermore a set of physical tools allowed the direct manipulation of the underlying digital application models. This early work establishes several classes of physical objects that are employed as pointers (handles), symbolic tokens (phicons) and physical tools (color palette).

3.2 TANGIBLE USER INTERFACES

Ishii et al.[6] later established the term Tangible User Interfaces (TUI), which was supported by the fundamental idea of coupling the physicality of tangible objects with digital information. By connecting atoms and bits, symbolizing the physical and digital domain, TUIs thereafter concentrate on the interaction with physical objects supported with digital media. The meaning of the term tangible as derived from the Latin verb *tangere - to touch* does not primarily refer to the haptic aspects of the interface, but rather the interaction with its physicality. This actual meaning is also well expressed within the German definition[7] *Gegenständliche Schnittstellen*, which emphasizes the central role of the manipulated physical object as *Gegenstand*.

The ongoing development of the now established research area within the Tangible Media Group, yielded several rather distinct projects that evolved the idea of tangible interaction away from the desktop metaphor. While projects such as the URP[8] platform still organized the physical interface components within the spatial context of an augmented surface environment, further projects such as Ullmer's MediaBlocks[9] concentrated on the relationship of physical objects and their role as containers for digital media. Furthermore Gorbet's Triangles[10] represent constructive assemblies, allowing the construction of physical structures based on mechanical connections without explicit spatial reference. Ullmer et al. summarize[11] the various approaches of organizing physical objects within spatial systems defined by Cartesian positions, angles and distances, relational systems defined by their adjacencies and content and constructive assemblies defined by their physical topologies. I intend to add here a fourth category of autonomous artifacts, which are representing self-contained physical objects, such as Ishii's et al. Music Bottles for example.[12]

3.3 TANGIBLE OBJECT MODELS

Ullmer et al. later define the fundamental Control-Representation Model, which emphasizes the primary function of physical objects as control and representation of the underlying digital application model, which can additionally be supported by ambient digital media. This conceptually defines the physical object as input device providing control data to an application model, which it physically embodies. Fishkin[13] establishes a high-level taxonomy of Embodiment and Metaphor, such as elementary grammars defining nouns as symbolic objects or verbs as functional objects for example, as well as various levels of distant, nearby or full embodiment reflecting the "immersion" of digital content into a physical container. In addition to defining the general design paradigms and the organization of physical objects that are employed for the construction of tangible user interfaces, the subsequent research concentrated on the semantic definition of the object content, function and their relationship to each other. First of all several object types needed to be identified, yielding terms such as phicons, containers, tools or props, which concentrated on the application specific content or function of each individual object. Ullmer introduced the concept of Token-Constraint Systems, which defined the function and behaviour of basic interface elements as elementary tokens within the context of enclosing physical constraints.[14] Shaer et al. elaborate this concept with the establishment of design patterns based on the physical association of several tokens in the form of dedicated Token and Constraint Systems (TaC).[15] This work leads to the specification of the Tangible User Interface Modeling Language (TUIML)[16] which formalizes the TUI design process at the application level. TUIML has been implemented within the Tangible User Interface Management System TUIMS, which also defines additional lexical handlers for the various input technologies. Nevertheless we can observe that most recent models concentrate on the establishment of design patterns within the application domain, and are therefore implicitly masking the physical domain and the necessary sensory data acquisition methods for the description of the physical environment and object states.

3.4 TANGIBLE INTERFACE TOOLKITS

Compared to the top-down approach of the application oriented object models most software and hardware toolkits on the other hand provide a bottom-up abstraction of the physical input layer. Although there exists a myriad of advanced sensor hardware, computer vision libraries and physical computing platforms which generally facilitate the design and development of tangible user interfaces, these tools often require substantial customization and extension to be suitable for a specific application scenario. Nevertheless there also exist some toolkits and platforms, which not only provide the necessary hard-

and software infrastructure, but also define a set of elements that encapsulate the common tangible interface components.

Today many external sensing systems employ generic computer vision libraries such as OpenCV[1], which provides the basic functionality to detect physical objects in an environment, by analyzing their colour and shape for example. More specific applications, such as the popular ARToolkit[17] or its various descendants provide the tracking of visual markers, which can be also attached to physical objects. This not only allows the identification of marked physical tokens, but also the precise tracking of their position and orientation. Apart from standard digital cameras, there also exist specific optical sensing devices such as the Leap Motion or Microsoft Kinect. Although these sensors had been primarily designed for gestural interaction, they are also commonly employed for the analysis of physical environments and object tracking and recognition tasks.[18] While the individual technology and algorithms of these external sensing approaches may differ, from a tangible interface perspective they provide generic components for *Tokens* (tangible objects), *Symbols* (markers) and *Geometries* (descriptors) as well as *Pointers* (gestures).

Physical computing platforms, such as the Phidget Toolkit[19] provide a modular approach for the construction of self-contained physical objects with embedded sensors and actuators. While today there exist many hardware environments that provide a low entry level for the assembly and programming of complex electronics, specific tangible interface toolkits such as Bdeir's Little Bits[20] also provide substantial abstraction in the material domain, allowing for a direct physical programming paradigm. Generally we can observe that most of these tangible platforms, also including Merrill's [21] Sifteo Cubes can be defined through physical *Tokens* with associated *Controls*.

Klemmer's early multi-sensor toolkit Papier-Maché[22] defines *Phobs* that represent physical objects, marked with an RFID tag or other symbols. *VisionPhobs* on the other hand encapsulate computer vision objects, including the image bitmap for further processing. Papier-Maché therefore handles the basic classes of *Tokens*, specified by *Symbols* and simple *Geometries*. The toolkit also implements a fundamental event model for physical object handling: *addPhob*, *updatePhob* and *removePhob* provide the interaction data according to the general object manipulation events.

Interactive tabletop systems today are primarily focused on gestural multi-touch interaction with digital content. Only few commercially available platform such as the Microsoft SUR40, also provide the interaction with physical interface components to the application developer. In addition to generic multi-touch input, the platform is capable of tracking visual markers, tagged objects as well as untagged objects or blobs, resulting in a component structure of *Symbols*, *Tokens*, *Pointers* and *Geometries*. The platform SDK additionally provides a set of high level multi-touch gestures based on the finger input in relation to the surface plane and the tangible objects.

1 http://opencv.org/

3.5 TANGIBLE INTERACTIVE SURFACES

As shown above, tangible user interfaces have been implemented in a variety of form factors, which commonly share the design paradigm of the direct manipulation of physical artifacts and gestural input based on the analysis of body, hand or finger movements or the direct tracking of physical objects with embedded or external sensor systems. While there exist a variety of implementations where the tangible artifacts are self-contained and can be manipulated independently, many tangible interfaces place these physical objects into the context of a certain spatial environment or physical reference frame. This environment can be either a dedicated physical container object, which constrains the behavior of simple physical tokens, where in other cases this environmental context can be an entire room, a wall or simply a table surface. I will concentrate on the description of the specific case of interactive surfaces, such as a vertical blackboard or wall, or a horizontal table or floor.

These interactive surface environments generally provide a versatile and augmentable (e.g. through digital sound and image) reference frame for manifold application scenarios, where tangible tokens can be placed, moved and manipulated in direct contact with the surface or within the space that extends above or in front of that surface. Users can directly interact with the physical objects, or directly with the surface and within the whole spatial environment, which usually incorporates a hardware specific sensor system, capable of tracking the current state of all involved interface components and the gestures performed by the user on the surface itself or through the physical manipulation of objects.

Interactive surfaces usually provide additional visual feedback using common screen or projection technologies. In many cases the visual or physical surface design is also partially predefined with graphically printed or physically crafted design elements, where in some cases any computationally generated visual feedback may be even fully omitted. Since tangible interfaces also can incorporate multimodal feedback, the environment, the surface structure and even the tangible objects themselves can also embed additional digitally controllable acoustic or visual feedback. In addition to the passive haptic quality of the physical interface components, these can also be equipped active haptic feedback using tactile or kinetic actuators.

This thesis will concentrate on the comprehensive description of such a tangible surface environment, by defining an abstraction model, which encapsulates a continuously updated state description of an interactive surface including all its physical interface components. This model is represented through a specific protocol syntax, which communicates the semantic abstraction to an underlying digital application layer. This allows the implementation of generally platform-independent tangible user interfaces, taking advantage of its generalized multi-platform and application-invariant interface abstraction.

Part II

SELECTED PUBLICATIONS

The following four first-author papers, which were published during the years from 2005 until 2009, form the central part of this cumulative dissertation. According to Google Scholar this research as of today has accumulated more than 1100 citations since its publication.[2]

[1] Martin Kaltenbrunner, Till Bovermann, Ross Bencina, and Enrico Costanza. "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, 2005.

[2] Martin Kaltenbrunner, Sergi Jordà, Günter Geiger, and Marcos Alonso. "The reacTable: A Collaborative Musical Instrument." In: *Proceedings of the Workshop on Tangible Interaction in Collaborative Environments (TICE)*. Manchester, U.K., 2006.

[3] Martin Kaltenbrunner and Ross Bencina. "reacTIVision: A Computer-Vision Framework for Table-Based Tangible Interaction." In: *Proceedings of the first international conference on Tangible and Embedded Interaction (TEI07)*. Baton Rouge, Louisiana, 2007.

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

These core publications constitute the model definition within the TUIO protocol, its implementation within the reacTIVision framework, an application within the Reactable and the final evaluation of the proposed abstraction model for tangible interactive surfaces through a subsequent evolution of the reacTIVision and TUIO framework. Each publication chapter is preceded by an initial clarification of my contributions in the particular research context and is followed by a concluding analysis and further clarifications that have become relevant since the initial publication of each paper. The comment section of the final chapter in this section provides a deeper strength and weakness analysis in the context of all four publications and also documents several improvements to the reacTIVision and TUIO framework since their original release.

Disclaimer: Due to the separate publication dates of these papers, this cumulative dissertation may include some redundant our superseded information, which will be clarified from a historical perspective in the comment section after each individual publication.

2 http://scholar.google.com/citations?user=G7rN7JUAAAAJ

These additional eight, and only partially first-authored publications are also relevant in the context of this dissertation, but have been omitted in order to avoid redundancies.

[1] Martin Kaltenbrunner, Günter Geiger, and Sergi Jordà. "Dynamic Patches for Live Musical Performance." In: *Proceedings of the 4th Conference on New Interfaces for Musical Expression (NIME04)*. Hamamatsu, Japan, 2004.

[2] Martin Kaltenbrunner, Sile O'Modhrain, and Enrico Costanza. "Object Design Considerations for Tangible Musical Interfaces." In: *Proceedings of the COST287-ConGAS Symposium on Gesture Interfaces for Multimedia Systems*. Leeds, UK, 2004.

[3] Ross Bencina and Martin Kaltenbrunner. "The Design and Evolution of Fiducials for the reacTIVision System." In: *Proceedings of the 3rd International Conference on Generative Systems in the Electronic Arts (3rd Iteration 2005)*. Melbourne, Australia, 2005.

[4] Ross Bencina, Martin Kaltenbrunner, and Sergi Jordà. "Improved Topological Fiducial Tracking in the reacTIVision System." In: *Proceedings of the IEEE International Workshop on Projector-Camera Systems (Procams 2005)*. San Diego, USA, 2005.

[5] Sergi Jordà, Martin Kaltenbrunner, Günter Geiger, and Ross Bencina. "The reacTable*." In: *Proceedings of the International Computer Music Conference (ICMC 2005)*. Barcelona, Spain, 2005.

[6] Sergi Jordà, Günter Geiger, Marcos Alonso, and Martin Kaltenbrunner. "The reacTable: Exploring the Synergy between Live Music Performance and Tabletop Tangible Interfaces." In: *Proceedings of the first international conference on Tangible and Embedded Interaction (TEI07)*. Baton Rouge, Louisiana, 2007.

[7] Martin Kaltenbrunner and Florian Echtler. "TUIO Hackathon." In: *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces (ITS2014)*. Dresden, Germany, 2014.

[8] Florian Echtler and Martin Kaltenbrunner. "SUR40 Linux: Reanimating an Obsolete Tangible Interaction Platform." In: *Proceedings of the ACM International Conference on Interactive Surfaces and Spaces (ISS2016)*. Niagara Falls, Canada, 2016.

4 TUIO: A PROTOCOL FOR TABLE-TOP TANGIBLE USER INTERFACES

4.1 INTRODUCTION

The first publication of this cumulative thesis includes the original specification of the TUIO protocol in its initial version 1.0, representing the **definition of the initial Tangible Abstraction Framework**. The following paper has been reformatted and corrected without any changes to its original content.

4.1.1 Co-Authors

In addition to my core contribution of the TUIO abstraction model and its draft protocol implementation as first author, the following co-authors collaborated in the inter-operability aspects with their tangible interaction platforms, and the related design issues:
Till Bovermann lfsaw@lfsaw.de
then: Neuroinformatics Group, Bielefeld University, Germany.
now: Time-based Media, Berlin University of the Arts, Germany.
Ross Bencina rossb@audiomulch.com
then: Music Technology Group, University Pompeu Fabra, Spain.
now: Sonic Fritter Pty Ltd, Melbourne, Australia.
Enrico Costanza e.costanza@ucl.ac.uk
then: Liminal Devices Group, Media Lab Europe, Dublin, Ireland.
now: Interaction Centre, University College London, UK.

4.1.2 Context

Together with Till Bovermann, this publication has been presented as a poster at the *6th International Workshop on Gesture in Human-Computer Interaction and Simulation*[1] which took place from May 18-20 2005 at the Berder Island in France. Although this poster today represents the most cited publication from the proceedings of this workshop, it unfortunately had not been included in the final "revised selected papers" book, which was published in the *Lecture Notes in Computer Science* series by Springer later in 2005[23].

This publication and the final TUIO specification are the result of an intensive workshop organized upon my initiative, which took place during the first week of November 2004 at the Music Technology group in Barcelona, Spain. The participation of two external co-authors was supported through a Short Term Scientific Mission by the Cost287-ConGAS[2] program on *Gesture Controlled Audio Systems*.

1 http://www-valoria.univ-ubs.fr/gw2005
2 http://www.cost.eu/COST_Actions/ict/287

4.2 ORIGINAL PUBLICATION

4.2.1 Abstract

In this article we present *TUIO*, a simple yet versatile protocol designed specifically to meet the requirements of table-top tangible user interfaces. Inspired by the idea of interconnecting various existing table interfaces such as the *reacTable* [24], being developed in Barcelona and the *tDesk* [25] from Bielefeld, this protocol defines common properties of controller objects on the table surface as well as of finger and hand gestures performed by the user. Currently this protocol has been implemented within a fiducial marker-based computer vision engine developed for the *reacTable* project. This fast and robust computer vision engine is based on the original *d-touch* concept [26], which is also included as an alternative to the newer fiducial tracking engine. The computer vision framework has been implemented on various standard platforms and can be extended with additional sensor components. We are currently working on the tracking of fingertips for gestural control within the table interface. The *TUIO* protocol has been implemented using *OpenSound Control* [27] and is therefore usable on any platform supporting this protocol. At the moment we have working implementations for Java, C++, PureData, Max/MSP, SuperCollider and Flash.

4.2.2 General Observations

This protocol definition is an attempt to provide a general and versatile communication interface between tangible table-top controller interfaces and underlying application layers. It was designed to meet the needs of table-top interactive surfaces, where the user is able to manipulate a set of objects. These objects are tracked by a sensor system and can be identified and located in position and orientation on the table surface. Additionally we defined a special cursor object, which doesn't have a unique ID and doesn't provide rotation information. The protocol's flexible design offers methods for selecting which information will be sent. This flexibility is provided without affecting existing interfaces, or requiring re-implementation to maintain compatibility.

4.2.3 Implementation Details

The *TUIO* protocol defines two main classes of messages: *set* messages and *alive* messages. *Set* messages are used to communicate information about an object's state such as position, orientation, and other recognized states. *Alive* messages indicate the current set of objects present on the surface using a list of unique session IDs. To avoid possible errors evolving out of packet loss, no explicit add or remove messages are included in the *TUIO* protocol. The receiver deduces object lifetimes by examining the difference between sequential *alive* messages. In addition to *set* and *alive* messages, *fseq* messages are defined to uniquely tag each update step with a unique frame sequence ID. To summarize:

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

4.2.3.1 *Efficiency & Reliability*

In order to provide low latency communication our implementation of the *TUIO* protocol uses UDP transport. When using UDP the possibility exists that some packets will be lost. Therefore, our implementation of the *TUIO* protocol includes redundant information to correct possible lost packets, while maintaining an efficient usage of the channel. An alternative TCP connection would assure the secure transport but at the cost of higher latency. For efficiency reasons *set* messages are packed into a bundle to completely use the space provided by a UDP packet. Each bundle also includes a redundant *alive* message to allow for the possibility of packet loss. For larger object sets a series of packets, each including an *alive* message are transmitted. When the surface is quiescent, *alive* messages are sent at a fixed rate dependent on the channel quality, for example once every second, to ensure that the receiver eventually acquires a consistent view of the set of alive objects. The state of each alive but unchanged object is periodically resent with additional *set* messages. This redundant information is resent at a lower rate, and includes only a subset of the unchanged objects at each update. The subset is continuously cycled so that each object is periodically addressed. Finally, each packet is marked with a frame sequence ID (*fseq*) message: an increasing number which is the same for all packets containing data acquired at the same time. This allows the client to maintain consistency by identifying and dropping out-of-order packets. To summarize:

– *set* messages are bundled to fully utilize UDP packets
– each bundle of *set* messages includes an *alive* message containing the session IDs of all currently alive tangible objects
– when the surface is quiescent the *alive* message is resent periodically
– the state of a cycling subset of alive but unchanged objects is continuously resent via redundant *set* messages
– each bundle contains a frame sequence (*fseq*) message

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

4.2.3.2 *Message Format*

Since *TUIO* is implemented using Open Sound Control (OSC) [27] follows its general syntax. An implementation therefore has to use an appropriate OSC library such as *oscpack* [28] and has to listen to the following message types:

```
/tuio/[profileName] set sessionID [parameterList]
/tuio/[profileName] alive [list of active sessionIDs]
/tuio/[profileName] fseq (int32)
```

4.2.3.3 *Parameters*

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

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

Table 1: semantic types of set messages

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

4.2.3.4 *Profiles*

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

2D interactive surface

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

2.5D interactive surface

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

3D interactive surface

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

raw profile

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

custom profile

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

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

4.2.4 Conclusion

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

4.2.5 Acknowledgments

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

4.3 remarks & analysis

4.3.1 Distributed Architecture

The original motivation for the development of the TUIO protocol was largely based on performance limitations, which didn't allow for the concurrent execution of a then still demanding computer-vision algorithm along with real-time sound-synthesis and complex graphics on a single CPU. Therefore the first experimental implementations of the Reactable had to be based on a distributed architecture, separating the computer-vision sensor component from the actual instrument component including the synthesizer and visual feedback. These components were initially running on two separate network-connected machines, but soon replaced by a dual-cpu desktop machine and later a dual-core laptop. Nevertheless this modular approach through an internal messaging infrastructure between the various system components proved to be quite practical for the further development process, since it allowed for the easier evaluation and integration of alternative sensor and synthesizer components.

4.3.2 Model Robustness

After initial implementations of a custom UDP based messaging infrastructure, we chose the then emerging Open Sound Control encoding, mostly due to its relation to our musical application scenario, and its easy integration into Pure Data, which was used for the first implementation of the Reactable synthesizer. OSC generally prefers UDP transport over TCP in order to minimize the overall latency, which is critical for an interactive sonic system. Later tests have shown that on a local system the accumulated latency of the network layer is neglectable when compared to the input and output delays from the complete camera-projector system.

Early event-based experiments, sending explicit messages for each *add*, *update* and *remove* event with every component update, had proven rather error-prone due to the very likely loss of UDP packets on noisy WIFI networks. While a lost *add* event still could be reconstructed from a following *update* event, a lost *remove* event would inevitably lead to an inconsistency on the client side. Therefore we chose a state-based model for the TUIO protocol, where in addition to *update* events through the TUIO *set* message, the actual *add* and *remove* events have to be implicitly derived from the ID changes in the TUIO *alive* message, which is sent along with each TUIO bundle. This model allows for a consistent surface component representation on the client side, while an eventual packet loss only leads to a lower update rate in a worst case scenario. Closing a TUIO bundle with an additional *fseq* message including an incremental frame ID, not only marks the end of each sensor update cycle, but allow assures further consistency in case of an out-of-order arrival.

4.3.3 Tangible User Interface Objects

As the TUIO acronym - *Tangible User Interface Objects* - suggests, the main focus of the TUIO protocol actually lies in the physical domain of tangible interactive surfaces. This most importantly includes the manipulation of physical objects in direct contact or above an interactive surface. Touch interaction as such therefore was originally intended to provide a complementary interaction modality, by supporting the primary object manipulation gestures with additional touch gestures performed around these physical objects. TUIO objects in the original conception refer to formless physical tokens, which can be located on the surface through normalized cartesian coordinates and their absolute orientation, as well as identified through an arbitrary marker. While our implementation was based on computer vision and fiducial markers attached to the tokens, the actual object ID can also be determined from color or shape as well as alternative markers such as RFID tags. Apart from that TUIO 1.0 did not yet provide any further information about the actual object size and geometry, which therefore had to be handled entirely at the application layer.

4.3.4 Multi-Touch Interaction

Through its definition of an additional cursor profile, TUIO 1.0 also allows multi-touch interaction on platforms without any object-tracking capabilities, which in fact turned out to be the more attractive feature provided by the protocol. By the time of its publication mainstream operating systems and mobile computing platforms did not yet provide any kind of multi-touch hardware nor an official application programming interface. With the advent of smart-phone platforms such as the Apple iPhone as well as Jeff Han's iconic FTIR based [29] implementation, multi-touch interaction concepts gained a widespread popularity, although its conceptual foundations and basic gestural ideas had already been explored [30] earlier in the late 20th century. Due to the initial lack of commercially available hardware platforms, researchers[31] and other professionals engaged in an open source DIY community, implementing custom made hardware and software platforms, mostly based on computer vision in camera-projector configurations. This also led to the first third-party TUIO tracker implementations, such as touchlib and community core vision (then tBeta), taking advantage of the various TUIO client implementations that we had provided in conjunction with reacTIVision. Although nowadays many mobile platforms such as iOS and Android, as well as mainstream operating systems such as Windows and Linux already provide integrated multi-touch support, mostly in conjunction with capacitive touch displays, TUIO still plays a relevant role for system integrators of large-scale multi-touch platforms, which are primarily used in museum exhibitions and the advertisement industry. Most commercial manufacturers of capacitive touch-foils and optical touch-frames also provide native TUIO support in addition to a standard HID interface. While commercial camera-based systems[3] have become less common than capacitive touch-only displays, they still offer the advantage of additional marker-based object tracking capabilities. In addition to that, some manufacturers have also started to implement capacitive object tracking capabilities[4] into their interactive display solutions. Hence the lack of alternative representations of the tangible object layer in mainstream operating-systems, has led to the full implementation of the TUIO touch and object profiles in these commercial products. Other proprietary tangible interaction platforms, such as the now abandoned Microsoft Surface also have been extended with third-party TUIO support, taking advantage of the wide range of TUIO-enabled programming environments.

4.3.5 Surface Interaction

The original TUIO specification has a primary focus on tangible objects and touch gestures in direct contact with the tangible interactive surface, which are represented in the two primary `/tuio/2Dobj` and `/tuio/2Dcur` profiles. Nevertheless the overall TUIO concept not only defines on-surface interaction, but also includes above-surface interaction through its fish-tank metaphor. In order to avoid the transmission of excess 3D attributes in on-surface configurations and in order to also allow the explicit distinction of on-surface and above-surface interactions, the TUIO protocol also defines additional 2.5D and 3D profiles that include the necessary additional attributes to encode the distance and orientation for tangible objects. Our TUIO reference implementations primarily focus on the 2D profiles though, also due to reduced availability of tangible interaction platforms with three-dimensional sensing capabilities. Nevertheless there also exist some experimental platforms with above surface tracking capabilities[32], which also implement the TUIO 3D profiles[33] necessary in such as scenario. Other platforms incorporate 3D pointing devices such as the Nintendo WiiMote through the 2.5D cursor profiles, as well as additional spatial object tracking capabilities with the Microsoft Kinect depth sensor through the 3D object profiles.

4.3.6 Crowd-Sourced Development

Due to the open availability of the protocol specification and its distributed architecture, today there exists a vast variety of community supported TUIO client and server implementations in addition to our own open-source reference implementations. An updated list of these third-party contributions, including many open-source and commercial software applications and libraries as well as hardware platforms, is maintained on the TUIO website, which was established in 2009.[5]

3 http://www.multitaction.com/hardware
4 http://www.interactive-scape.com
5 http://www.tuio.org

5 REACTIVISION: A COMPUTER-VISION FRAMEWORK FOR TABLE-BASED TANGIBLE INTERACTION

5.1 INTRODUCTION

The second paper of this cumulative thesis includes the first comprehensive publication of the reacTIVision framework in its version 1.3, representing the **implementation of the initial Tangible Abstraction Framework**. The following paper has been reformatted and corrected without any changes to its original content.

5.1.1 Co-Author

Ross Bencina rossb@audiomulch.com
then: Music Technology Group, University Pompeu Fabra, Spain.
now: Sonic Fritter Pty Ltd, Melbourne, Australia.

5.1.2 Context

This paper has been selected for publication at the *First International Conference on Tangible Interaction*[1] which took place from February 15-17 2005 in Baton Rouge, Louisiana. Together with its companion paper on the Reactable[34] it represents one of the most cited publications of this conference series and has been awarded with the *Lasting Impact Student Award*[2] at the 10th edition of the TEI conference.

Following two initial publications which primarily focused on the definition[35] and generation[36] of the fiducial tracking engine, this publication discusses the overall design of the reacTIVision framework itself as a feature complete reference implementation of the initial TUIO specification.

While the included amoeba fiducial design and libfidtrack implementation by Ross Bencina represent an improved implementation of the initially employed d-touch[26] concept by Enrico Costanza, the core design of the reacTIVision framework is generally invariant of the actual fiducial tracking method. Therefore reacTIVision is described as an extensible computer vision framework for various marker-based object and finger tracking methods, which primarily focuses on the representation of abstract tangible interface components on interactive surfaces. In addition to the standalone application, this also constitutes an initial reference implementation of the complete TUIO framework through a series of client implementations for several programming languages and environments.

1 https://tei.acm.org/2007/
2 https://tei.acm.org/2016/

5.2 ORIGINAL PUBLICATION

5.2.1 Abstract

This article provides an introductory overview to first-time users of the reacTIVision framework – an open-source cross-platform computer-vision framework primarily designed for the construction of table-based tangible user interfaces. The central component of the framework is a standalone application for fast and robust tracking of fiducial markers in a real-time video stream. The framework also defines a transport protocol for efficient and reliable transmission of object states via a local or wide area network. In addition, the distribution includes a collection of client example projects for various programming environments that allow the rapid development of unique tangible user interfaces. This article also provides a discussion of key points relevant to the construction of the necessary table hardware and surveys some projects that have been based on this technology.

5.2.2 Introduction

The reacTIVision framework has been developed as the primary sensor component for the reacTable [37], a tangible electro-acoustic musical instrument. It uses specially designed visual markers (fiducial symbols see Fig.4) that can be attached to physical objects. The markers are recognized and tracked by a computer vision algorithm optimized for the specific marker design [36] improving the overall speed and robustness of the recognition process. These fiducial marker symbols allow hundreds of unique marker identities to be distinguished as well as supporting the precise calculation of marker position and angle of rotation on a 2D plane.

reacTIVision and its components have been made available under a combination of open source software licenses (GPL, LGPL, BSD) and can be obtained both as ready to use executable binaries and as source code from a public SourceForge site. This document describes the features of reacTIVision 1.3 which has been released in conjunction with the publication of this article. The reacTable software website[3] provides further information about the project.

5.2.3 Architecture

reacTIVision has been designed as a distributed application framework rather than an object code library. Each component of the system is implemented as a separate executable process. Communication between components is achieved using a published protocol. This design simplifies use for novice programmers and facilitates integration with popular programming environments such as Processing and Pure Data. The architecture also allows the execution of framework

3 http://reactivision.sourceforge.net/

components on different machines, which can be useful in certain installation contexts.

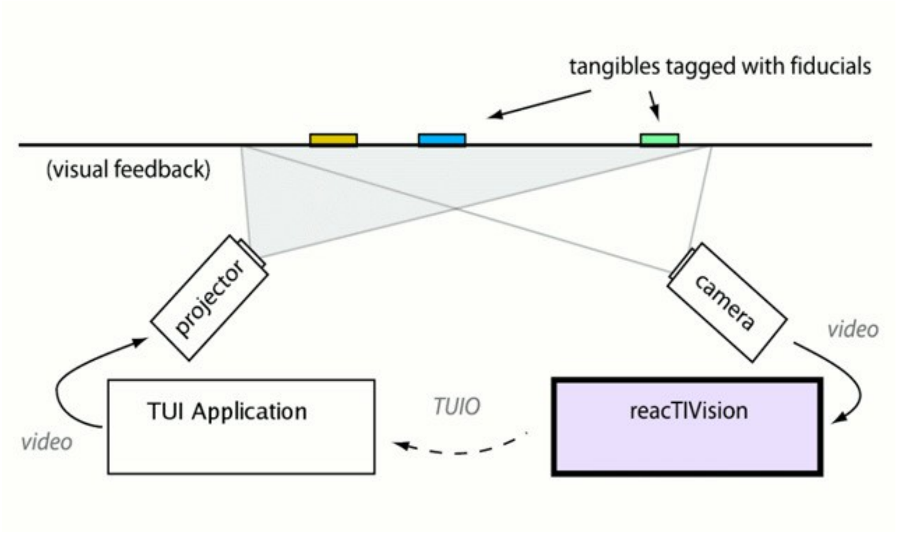

Figure 3: reacTIVision diagram

This article provides an introductory overview to first-time users of the reacTIVision framework – an open-source cross-platform computer-vision framework primarily designed for the construction of table-based tangible user interfaces. The central component of the framework is a standalone application for fast and robust tracking of fiducial markers in a real-time video stream. The framework also defines a transport protocol for efficient and reliable transmission of object states via a local or wide area network. In addition, the distribution includes a collection of client example projects for various programming environments that allow the rapid development of unique tangible user interfaces. This article also provides a discussion of key points relevant to the construction of the necessary table hardware and surveys some projects that have been based on this technology.

Recognition Component The reacTIVision application acquires images from the camera, searches the video stream frame by frame for fiducial symbols and sends data about all identified symbols via a network socket to a listening application. The reacTIVision application has been designed in a modular way, making it easy to add new image recognition and frame processing components. The code base is cross-platform with builds for all three major operating systems, Windows, Mac OS X and Linux. It has been written in portable C++ code, combined with platform-dependent frame acquisition components. The video acquisition framework is also available separately as open source software under the name PortVideo.[4]

5.2.3.1 *Communication Component*

ReacTIVision defines its own communication protocol TUIO [38] that was specifically designed for the needs of tabletop tangible user interfaces: encoding and transmitting the attributes of tangible artifacts

4 http://portvideo.sourceforge.net/

that are found on a table surface. In order to provide fast and reliable communication with local and remote client applications the protocol layers a redundant messaging structure over UDP transport. TUIO defines a set of Open Sound Control [27] protocol messages. These messages constantly transmit the presence, position and angle of all found symbols along with further derived parameters. On the client side these redundant messages are then decoded to generic add, update and remove events corresponding to the physical actions that have been applied to each tangible object. In order to achieve maximum compatibility with existing musical application environments reacTIVision can alternatively send MIDI [39] control messages that can be individually configured for each fiducial symbol. However, due to the various limitations of MIDI, such as bandwidth and data resolution, TUIO is the recommended and default transport layer.

5.2.3.2 *Client Components*

In order to facilitate the development of tangible interface applications the reacTIVision framework provides a large collection of example clients for a variety of programming languages including C++, C#, Java, Processing and Pure Data. Example clients provide a full TUIO client implementation that decodes the messages to generic interface events and draws the results into a graphical window or simply prints them to the console. Additional unsupported example projects are available for SuperCollider, Max/MSP and Flash. The TUIO simulator written in platform independent Java can be used to simulate a table environment during the initial development phase.

5.2.4 Fiducial Engines

This section gives some background regarding the history, design, evolution, and capabilities of the marker tracking implementations employed by reacTIVision. After some initial experiments with publicly available marker systems such as ARToolkit [17], the first reacTable prototype made use of E. Costanza's original D-touch [26] code, which was kindly provided by its author. Further development of the reacTable generated requirements for more compact symbol sizes as well as improved processing speed for real time musical interaction. This first lead to a reimplementation of the d-touch tracking algorithm with significant performance gains. Subsequently the fiducial marker geometry was redesigned to take advantage of a genetic algorithm, which minimized marker size and facilitated a more efficient tracking algorithm. All three fiducial recognition engines (d-touch, classic and amoeba) are available within the reacTIVision application, with the most recent and reliable amoeba engine as the default. In all three fiducial engines the source image frame is first converted to a black & white image with an adaptive threshold algorithm. This image is then segmented into a region adjacency graph reflecting the containment structure of alternating black and white regions. This graph is searched for unique tree structures, which are encoded into the fiducial symbols. Finally the identified trees are matched to a dictionary to retrieve unique marker ID numbers.

5.2.4.1 *Amoeba Engine*

The highly compact geometry of the amoeba fiducials was obtained by a genetic algorithm. This GA optimized the fiducial appearance using a set of fitness functions targeting shape, footprint size, center point and rotation angle accuracy. The current set distributed with reacTIVision contains 90 different symbols that have been chosen from a pool of 128 with certain tree structure constraints. In this case all symbols have 19 leaf nodes and a maximum tree depth of 2. The limitation to specific tree structure constraints allows the exclusion of other structures found in noise, providing higher robustness of the algorithm by avoiding the detection of false positives. The position of an amoeba symbol is calculated as the centroid of all found leaf nodes (small circular monochrome blobs), which provides sub-pixel accuracy. The orientation of the marker is calculated as the vector from the marker centroid to the centroid of all black leafs which are distributed in the upper part of the symbol. A second fiducial set used internally for reacTable installations provides roughly 300 extra symbols that are usually printed onto business cards and handed out to the public. Just as with the standard symbol set, unique fiducial IDs are derived by comparing the detected tree structure to a dictionary of known trees.

5.2.4.2 *Finger Tracking*

As an initial solution for the tracking of fingertips in a multi-touch surface the simplest amoeba fiducial, with a single tree branch (see Fig.4d) can be used as a small finger sticker. While this method is not as elegant as other optical finger tracking methods [40] it has proven to be simple and robust without any additional computational overhead since it can be detected using the existing fiducial tracking algorithm. Due to the minimal nature of the symbol, no rotation angle can be calculated from its structure, although in the case of tracking the finger as a simple pointer the position information alone is sufficient. One drawback of this symbol's simple tree structure is the possibility of finding false positives in noise. In most cases false-positives can be filtered by taking into account the presence and trajectory of potential finger markers in past frames and neglecting the appearance of false positives in isolated frames. Recent reacTIVision development builds contain an improved plain finger tracking component, without the need of the described finger symbol sticker. This layer is fully taking advantage of information already provided by the segmenter in order to identify and track fingertips that are touching the table surface, at no significant additional computational cost. This additional plain object tracking also provided a method of double-checking objects that have been lost by the fiducial tracking core, which significantly improved the overall tracking robustness. Due to the relatively recent addition to the code base, this feature along with formal comparative results on its performance will be made available together with a future reacTIVision release.

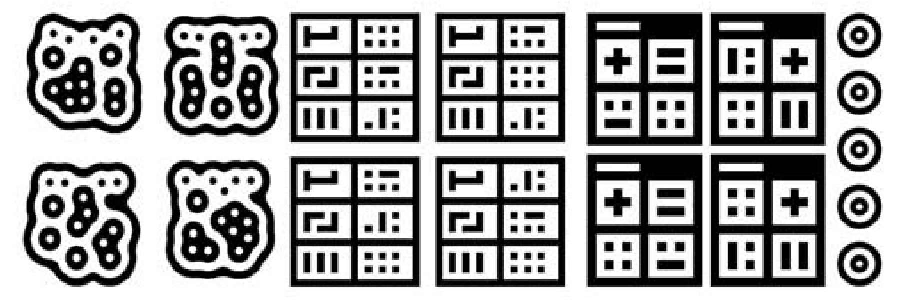

Figure 4: symbols a) amoeba b) classic c) d-touch d) finger

5.2.4.3 *Classic Engine*

The "classic" fiducial tracking engine uses the original d-touch fiducial set (figure 2b) and geometry evaluation scheme while its codebase has been re-implemented from scratch. The dice shaped fiducial symbols can represent 120 different identities that are obtained by permutations of the positions of regions with two to six sub-regions. The primary region with a single sub region is used for the determination of the rotation angle and is therefore always placed in the upper left corner of the symbol. As already mentioned above, the dice symbols do not optimally use the available space and the calculation of the fiducial center point and rotation angle is not as accurate as with the amoeba set.

5.2.4.4 *D-Touch Engine*

The original d-touch code was eventually released publicly under the GPL license and has since been integrated into reacTIVision. Although D-Touch can use a variety of different topology based symbol sets, including the original dice set used by the classic fiducial tracking engine, the implementation embedded in reacTIVision uses a reduced subset of the dice style symbols with 24 permutations of regions with one to four sub-regions. The extra region needed for angle calculation is a single empty box on top of the symbol, which occupies less space then the main code regions.

5.2.5 How to Build a Table Interface

5.2.5.1 *Table Construction*

The design of a table depends on both general application requirements and the installation environment. For a musical instrument the table needs to be mobile and easy to assemble and disassemble, for public installation the table needs to be robust and accessible. In many cases a standard glass table might be sufficient for building a first prototype. Apart from the general structure, the table's most important component is its surface. Whether used with or without projection, it is recommended that the table's surface be semitransparent, such as sanded glass or Plexiglas with a blurring coating. One simple way to achieve a blurring surface is to place a sheet of ordinary tracing paper on the table. The reason a blurring surface is desirable is that on transparent surfaces objects can be tracked above the table until the image loses focus, sometimes leading to unpredictable detection results. It is usually desirable that the objects are detected only when they are in contact with the table's surface, such that they disappear from the camera's view immediately when lifted. In addition to improving sensor behavior a semitransparent surface serves as an ideal projection screen for projected visual feedback, which in many cases is needed for table-based tangible user interfaces.

5.2.5.2 *Camera & Lens*

reacTIVision in general will work with any type of camera and lens. Most of the better quality USB or FireWire webcams with a resolution of 640x480 at 30fps will be sufficient. For larger tables, industrial grade USB2 or FireWire cameras provide higher resolutions and frame rates. If DV or video cameras are to be used, they need to support full frame mode, since an interlaced video signal completely destroys the structure of fiducial symbols in motion. When working with any computer vision system, the overall recognition performance is strongly dependent on the source image quality. Image quality results from a combination of various factors, which include the camera sensor, the lens quality, the illumination and other important camera and lens settings. In general we have found that cameras with CCD sensors provide much better overall image quality than CMOS sensors. Cameras with an exchangeable lens mount are to be preferred. To decrease the minimum distance to a sufficiently large surface the system needs to use wide-angle lenses. The necessary focal length of the lens can be calculated as a function of the sensor size, the distance to the surface and the diameter of the viewable area of the surface. Be aware that some consumer grade "wide-angle" lenses may not focus consistently across the full viewing area which can have detrimental effects on tracking performance. To set up the best image quality obviously requires that the lens is focused. A simple focusing procedure is to fully open the iris and then try to achieve the best possible focus. After that the iris can be slowly closed until a perfectly sharp image is achieved. In addition to focus, the camera's shutter speed needs to be fast enough to avoid motion blur, since long exposure times will cause blurry images of moving fiducial symbols, making them more difficult or impossible to recognize. Both narrower iris and faster shutter speeds result in less light reaching the sensor, which needs to be compensated by stronger illumination. Low lighting levels can also be corrected slightly by increasing the sensor gain, although too much gain will decrease the image quality by introducing grainy noise.

5.2.5.3 *Illumination*

In a camera-projector system the two visual components need to operate in different spectral bands so they do not interfere with each other. Since the projector obviously needs to operate in the visible range, the camera has to work in the infrared (IR) spectrum only. CCD camera sensors are perfectly sensitive to infrared light, but most of the time are protected with an IR filter which needs to be removed from the sensor housing or lens. At the same time the table setup needs to be illuminated with strong and diffuse IR light, which is completely invisible to the eye and therefore does not interfere with the table projection. Suitable light sources are IR LED arrays which are available in different intensities, alternatively one could use halogen lights, which produce a lot of IR but need to be equipped with IR pass filters which can be purchased in any photography shop. These IR pass filters also need to be applied to the camera in order to filter all visible light, most importantly from the projection, since the projected image would otherwise overlay and interfere with the fiducial symbols. In the case where no projection is required, the setup can operate in the visible spectrum, significantly simplifying the illumination process.

5.2.5.4 *Mirrors and lens distortion*

If a camera or projector does not have a sufficiently wide-angle lens, placing a mirror into the table helps to achieve a larger active surface while maintaining a relatively low table height. Unfortunately mirrors as well as wide-angle lenses produce distorted images both for the projection and the camera image.4 reacTIVision comes with a built-in calibration component which offers a simple mechanism to correct these distortion errors. In the case of projection the image needs to be pre-distorted by applying the image as a texture onto a virtual surface in order to again appear straight on the table surface. The TUIO distortion example code provides a simple graphical feedback component with built-in distortion engine. Both distortion components, the reacTIVision sensor application as well as the application providing the visual feedback, need to be calibrated in order to match the physical object position with the virtual projection position. See the usage section below for more details on the calibration process.

5.2.5.5 *Computer Hardware*

The rest of the hardware can be built from standard off-the-shelf components. In many cases a modern dual-core computer will be more than sufficient to handle both the computer vision component along with the actual tangible interface application. For self-contained table setups a laptop or small shuttle PC might be the right choice if everything needs to fit inside the table. The projector usually resides underneath the projection surface pointing at a mirror on the table's bottom edge, therefore a small form-factor combined with a strong lamp and an appropriate wide-angle lens are its most important features. Since projectors can produce a considerable amount of heat, appropriate ventilation must be assured to avoid overheating within the table.

5.2.5.6 *Tangibles*

Almost any object, including simple wooden or plastic geometric shapes, everyday objects or artifacts, and even food or vegetables can be turned into an easily trackable tangible interface component by attaching a fiducial marker to it. Ideally the symbol has to be attached on the bottom side of the object in order to hide it from the user's attention and also to avoid possible hand occlusion problems. The fiducial symbol set can be printed with a laser printer onto ordinary white office paper. Gray recycled paper is less desirable as it tends to degrade symbol contrast. Some ink-jet inks are invisible in the infrared domain and therefore unusable for IR illuminated setups, although such ink can be used to add additional user-readable color codes to the symbols that stay invisible to the computer vision component. In order to protect the symbols from scratches, and color loss, the printed paper surface can be coated with transparent adhesive foil, which also simplifies cleaning of the symbol's surface from dirty spots that can degrade recognition.

5.2.6 Framework Usage

5.2.6.1 *reacTIVision application handling*

The main reacTIVision application only provides a very simple GUI showing the actual camera image and some visual feedback on the fiducial detection performance. It is generally configured by calling the application with various command line options at startup and can be controlled with some key commands during execution. Startup options include the configuration of the following features. See the documentation that comes with the application for more details.

- Distortion mode and calibration file
- Fiducial engine alternatives
- TUIO host name and port number
- Optional MIDI transport and configuration file
- Parameter inversion (when using mirrors)

During runtime the following features of the reacTIVision application can be controlled using simple key commands.

- Switch to calibration mode
- Change the display type
- Verbose on screen and console feedback
- Application control: pause and exit

5.2.6.2 *Distortion calibration procedure*

This section briefly explains the calibration procedure using the reacTIVision sensor application in conjunction with the TUIO distortion example, made available by *Marcos Alonso* as part of the reacTIVision framework. This example application can be extended to take advantage of its distortion correction functionality. In the calibration folder that comes with the application package there are two calibration sheet examples for rectangular and square table setups, which can also be used for round tables. Print the desired document scaled to match the size of your visible table surface and place the sheet onto the table with the calibration grid facing downwards. Start the TUIO_Distort application and switch to calibration mode by hitting the 'c' key. Using the keys 'a,w,d,x' adjust each vertex on the projected grid to match the vertices on the sheet. You can navigate between vertices using the cursor keys. After finishing this first calibration step you can switch the TUIO_Distort application to normal mode by hitting the 'c' key again while leaving the calibration sheet untouched in its positions on the table. Now start the reacTIVision application in distortion mode by providing a grid file with the '-g' option. Once started, switch into calibration mode by hitting the 'c'. In the same ways as for the calibration procedure of the projected graphics, you now need to adjust each vertex to match the grid on the sheet by using the keys mentioned above. After finishing this second calibration step and exiting the calibration mode by hitting the 'c' key, both applications will be synchronized and the projected visual object feedback should exactly match the physical object positions. You can see a preview the resulting image distortion within reacTIVision by hitting the 'g' key.

5.2.6.3 *Application programming*

All of the TUIO client examples for standard object oriented programming languages, such as C++, C#, Java and Processing implement an event based callback mechanism that notifies registered classes when objects are added, moved or removed from the table. The same events are generated for (finger tracking) cursor operations. In general, application logic has to implement the *TuioListener* interface, which defines various callback methods such as `addTuioObj()`, `updateTuioObj()` and `removeTuioObj()`. These methods are called by the framework-supplied *TuioClient* class, which derives events from the continuous stream of status information received from the sensor application. The *TuioClient* class has to be instantiated and started using the `connect()` method at the beginning of the session. It is also necessary to register all *TuioListener* classes that need to be notified by the *TuioClient* using the `addTuioListener()` method. The *TuioClient* operates in its own thread in the background until it is terminated using the `disconnect()` method. For environments such as PureData or Max/MSP, TUIO client objects are provided that decode events from the TUIO protocol stream and provide them to the environment via appropriate messages.

5.2.7 Example Projects Based on reacTIVision

5.2.7.1 *reacTable*

This table-based instrument has been the driving force for the development of the reacTIVision framework, since the reacTable's real-time music interaction and expressivity demand very high performance and recognition robustness from the sensor component. The physical artifacts on the reacTable surface allow the construction of different audio topologies in a kind of tangible modular synthesizer or graspable flow-controlled programming language. Several users can simultaneously manipulate various objects and control additional features with finger gestures. The reacTable web documents the various instrument features in greater detail.[5]

5.2.7.2 *recipe-table*

This project, which was shown during the *Ars Electronica Festival* 2005, has been developed by a group of students within the *Interface Culture Lab* at the University of Art and Industrial design in Linz. The recipe table is a fully working prototype of a future kitchen environment, where food and food products placed onto an interactive surface are detected and the system suggests a series of possible recipes that can be cooked with those ingredients. Changing the ingredients position in relation to each other allows the user to navigate within the possible recipes according to his or her personal preferences. reacTIVision has been used to identify and track the labeled products, simulating a barcode tracking system. In the near future such an environment could identify and track RFID labels that will soon be incorporated into standard consumer products. Further information about this intelligent environment and its creators can be found on the project web page.[6]

5.2.7.3 *Blinks & Buttons*

Blinks is a table-top interactive installation by the German artist *Sascha Pohflepp*, where projected photos are distributed on an interactive surface. Moving a glass prism over a photo causes it to refract the light to the sides of the table. This light contains projections of other photos taken at exactly the same moment in other locations. The user can browse the image collection over time. reacTIVision has been used to track the prism controller in conjunction with the Processing application. You can find more information about this installation at the project's web site.[7]

5 `http://mtg.upf.edu/project/reactable`
6 `http://www.recipe-table.net/`
7 `http://blinksandbuttons.net`

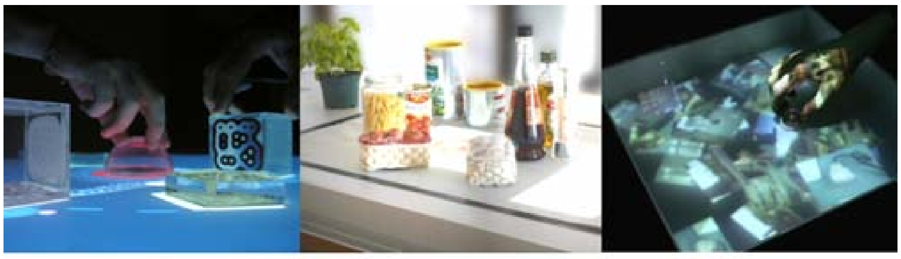

Figure 5: examples a) reacTable b) recipe-table c) blinks

5.2.8 Future Work

The reacTIVision framework is still being actively developed and the existing code-base will be improved and new features added. An important improvement in the next release will be the inclusion of the plain finger tracking layer, which doesn't require fiducial stickers on the fingertips. We are also planning to include additional fiducial engines such as *ARToolkit*, Barcodes and Semacode decoding into reacTIVision. Video acquisition under Linux needs to be extended to support a wider range of cameras, eventually by incorporating the promising unicap[8] library, which provides a uniform camera access method for Linux operating systems.

5.2.8.1 *Acknowledgments*

The authors would like to thank the Music Technology Group at the Universitat Pompeu Fabra for supporting the development of this publicly available software, as well as the rest of the reacTable team, *Sergi Jordà*, *Günter Geiger* and especially *Marcos Alonso* who support and contribute to this framework in various ways. We would also like to thank the numerous reacTIVision users for their suggestions and encouragement. Last but not least we are grateful for the initial support of *Enrico Costanza* by making the development of this framework possible with his earlier *D-Touch* contribution.

8 http://unicap-imaging.org/

5.3 remarks & analysis

This paper describes the feature set and architecture of reacTIVision in its historical version 1.3, which was released in November 2006. Constituting a feature complete reference implementation of a TUIO 1.0 tracker, it serves as a standalone computer-vision based application framework, providing generic object and finger tracking functionality, which is then encoded and transmitted through the `/tuio/2Dobj` and `/tuio/2Dcur` profiles.

5.3.1 Modular Architecture

While Bencina's libfidtrack implementation provides the core fiducial tracking functionality in addition to Costanza's original d-touch tracker, reacTIVision fundamentally serves as a generic and modular framework, which was designed for the integration of arbitrary computer-vision based tracking tasks not only limited to visual markers attached to physical tokens, and has as such been designed independently from any specific symbol types. It therefore establishes a modular architecture, which consists of several layers:

- *PortVideo* constitutes an underlying cross-platform camera acquisition layer, which provides a simple and uniform application programming interface for various digital cameras. In addition to the initial camera configuration and setup, it delivers an 8bit greyscale (or alternatively RGB color) frame buffer (including its efficient conversion from native camera formats) to the upper layers.
- The core image processing layer allows the integration of various *FrameProcessors*, which perform various tasks such as image calibration, binary thresholding, image segmentation as well as the integration of the actual fiducial tracking algorithms. These FrameProcessors provide basic buffer in- and output interfaces, as well interactive controls for the initial and online configuration of internal parameters.
- The *FiducialTracker* module retrieves the raw fiducial symbol data from each individual tracking library, and intends to maintain a robust global representation of all presently available symbols. This also includes additional position and angle filtering methods, as well as the application of a general heuristic model in order to improve the overall tracking stability.
- After the the analysis of each camera frame, the final *TuioServer* module encodes all currently present tangible object states through the `/tuio/2Dobj` profile attributes, which are transmitted to the TUIO client for further processing at the application layer.
- A global *VisionEngine* module manages the camera frame acquisition and the processing order of the following FrameProcessor queue. In addition to a simple XML configuration, it also provides a graphical user interface for the interactive application control and visualization.

5.3.2 Diffuse Illumination Surface

Due to the rather experimental nature of our early Reactable hardware and the resulting performance limitations such image distortion (caused by the use a convex mirror instead of a wide-angle lens) and limited contrast (caused by the use of tracing paper as projection surface), a robust native finger tracking implementation was virtually impossible to achieve. As stated in the paper, the Reactable is based on diffused infrared illumination, which at that time also resulted in additional hot-spot reflections disturbing the overall image quality. This problem could have been partially resolved by the integration of an additional FTIR based illumination system, and thus increasing the contrast of the finger blobs, although this would have affected the overall system complexity and cost.

Eventually our solution was the engineering of a dedicated table surface, which was optimized for the use in a camera-projection system with diffuse infrared illumination. On the one hand this acrylic surface was internally pigmented in order to achieve a proper projection quality, and also had a matte finish on the lower side in order to avoid the previously disturbing reflections. A properly adjusted diffused top surface also maintained one of the major advantages of the tracing paper, which provided a clear image for fiducial symbols in contact with the surface, but on the other hand blurred all symbols when lifted. The additional use of high quality wide-angle lenses in conjunction with an industrial camera system finally resulted in an adequate image quality and contrast, which allowed a sufficiently robust fiducial tracking and finger identification.

5.3.3 Integrated Finger Tracking

After these improvements to the Reactable hardware, our initial hack of using a set of small fiducial symbols as finger stickers, could be replaced with a more natural native finger tracking method. On the other hand, one of the (few) benefits of the fiducial-sticker workaround was to avoid an additional image processing step for the finger tracking. In order to maintain this significant performance advantage, we were actually employing the existing image segmentation data from the fiducial tracking engine, by retrieving possible candidate blobs within a certain size range without the need for any additional expensive image processing. These candidate regions were then simply analyzed by calculating the deviation error from the typical circular and/or elliptic finger shape, when in contact with the surface.

While this method proved to be sufficiently robust, when used in conjunction with the above surface configuration, many casual reacTIVision users found it difficult to achieve a stable finger tracking performance. On the one hand this was mainly caused by hardware factors mentioned above, such as reflection problems, limited image quality and contrast as well as poor infrared illumination and environmental light. On the other hand, this was also partially caused by our *Locally Adaptive Thresholding* algorithm, which had been optimized to achieve an optimal binary image for the primary fiducial tracking task. In some circumstances this method unfortunately produces image artifacts, which may negatively affect the quality of our finger tracking method. Since to date we couldn't find an improved thresholding algorithm with appropriate results for both tasks, we resolved these issues by improving the overall image contrast, using stronger infrared illuminators, high quality cameras and lenses, band-pass filters and the above mentioned surface structure.

5.3.4 Reactivision Projects

In addition to the Reactable, this paper also lists a few example projects which at the time of writing had been realized by using the reacTIVision and TUIO framework. Since apart from its principal research focus, this application framework also had been developed for teaching purposes in the field of Tangible User Interfaces, some of these assignments had been developed by our own students at the Interface Culture Lab in Linz, the Pompeu Fabra University in Barcelona and Catholic University of Porto as well as many other universities worldwide. After this publication, and partially also due to the growing popularity of the Reactable, this software has been employed for the realization of numerous scientific, artistic and educational interactive table projects from an active community. Apparently this was also supported by its ease of use, free availability, and the open-source nature of the whole framework, which also provided a collection of example projects to start with.

Although reacTIVision primarily had been designed for the particular use case of horizontal tabletops, it had been interesting to observe alternative usage scenarios of its tangible token tracking technology. While vertically oriented surface environments are usually designed for touch-only interaction due to to gravitational limitations, the *Roy Block* game by Sebastian Schmieg[9] incorporates the difficulties of vertical tangible interaction design into the actual game mechanics. Another noteworthy example is the expansion of the interaction space to the dimensions of a whole room, incorporating large-scale tangibles such as in the BuildaSound[41] application by Monika Rikić. Finally Tristan Hohne resolved the challenge of the seamless integration of fiducial tracking into the graphic design of his experimental interactive book project Hirngespinster[10] with highly aesthetic results.

As of today this individual publication has accumulated more than 420 citations according to Google Scholar, which are primarily from researchers that employ the reacTIVision platform for tangible interface research. But since outside the scientific community there are hundreds of otherwise unreferenced reacTIVision projects, we are maintaining a continuously updated list of selected examples on a dedicated Vimeo channel.[11]

9 http://file.org.br/file_prix_lux/sebastian-schmieg
10 http://vimeo.com/6329622
11 http://vimeo.com/channels/reactivision

6 THE REACTABLE*: A COLLABORATIVE MUSICAL INSTRUMENT

6.1 INTRODUCTION

The third publication of this cumulative thesis discusses the collaborative aspects of the tangible modular synthesizer Reactable, representing an **application of the initial Tangible Abstraction Framework**. The following paper has been reformatted and corrected without any changes to its original content.

6.1.1 Co-Authors

The following co-creators of the Reactable contributed to the overall musical instrument design, sound synthesizer implementation and visual feedback design, while my role as first author concentrates on the interaction design aspects and the tangible platform development.

Sergi Jordà sergi@reactable.com
then: Music Technology Group, University Pompeu Fabra, Spain.
now: Reactable Systems SL, Barcelona, Spain.
Günter Geiger gunter@reactable.com
then: Music Technology Group, University Pompeu Fabra, Spain.
now: Julius Blum GmbH, Höchst, Austria.
Marcos Alonso marcos@reactable.com
then: Music Technology Group, University Pompeu Fabra, Spain.
now: Apple Inc., Cupertino, California.

6.1.2 Context

This publication has been presented at the *15th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*[1] which took place from 26-28 June 2006 in Manchester, UK.

While there exist several relevant previous[37] and posterior[34] publications by the authors' collective on the Reactable, including two first-authored publications on the interaction design[24] and the object design[42] of this tangible musical instrument, this publication discusses the more relevant collaborative design aspects in the context of a distributed tangible interaction framework.

Although the Reactable is not in the main focus of this dissertation, its feature requirements were the motivation for the development of the fundamental tangible interaction technologies provided through the TUIO and reacTIVision framework. Thus the Reactable represents a comprehensive application scenario, showcasing the tangible object representation and multi-user interaction capabilities of the platform.

1 http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4092163

6.2 ORIGINAL PUBLICATION

6.2.1 Abstract

The reacTable* is a novel multi-user electro-acoustic musical instrument with a tabletop tangible user interface. In this paper we will focus on the various collaborative aspects of this new instrument as well as on some of the related technical details such as the networking infrastructure. The instrument can be played both in local and remote collaborative scenarios and was designed from the very beginning to serve as a musical instrument for several simultaneous players.

6.2.2 The reacTable*

The reacTable*, is a novel multi-user electro-acoustic musical instrument with a tabletop tangible user interface. Several simultaneous performers share complete control over the instrument by moving physical artefacts on the table surface while constructing different audio topologies in a kind of tangible modular synthesizer or graspable flow-controlled programming language.

The instrument hardware is based on a translucent round table. A video camera situated beneath, continuously analyzes the table surface, tracking the nature, position and orientation of the objects that are distributed on its surface. The tangible objects, which are physical representations of the components of a classic modular synthesizer, are passive, without any sensors or actuators; users interact by moving them, changing their position, their orientation or their faces. These actions directly control the topological structure and parameters of the sound synthesizer. A projector, also from underneath the table, draws dynamic animations on its surface, providing a visual feedback of the state, the activity and the main characteristics of the sounds produced by the audio synthesizer. The idea of creating and manipulating data flows is well acquainted in several fields, such as electronics, modular sound synthesis or visual programming, but the reacTable* is probably the first system that deals with this connectivity paradigm automatically, by introducing Dynamic Patching [24] where connections depend on the type of objects involved and on the proximity between them. By moving these objects on the table surface and bringing them into proximity with each other, performers construct and play the instrument at the same time, while spinning them as rotary knobs allows controlling their internal parameters.

6.2.2.1 *Current State*

The reacTable* structure and components have been discussed in detail in some earlier publications.[37] Since then, apart from general refinements of the synthesizer and the general system stability, the most significant improvements have been made to the table hardware itself and to the computer vision sensor component, which we have published recently as an open source software framework. The reacTIVision application along with example projects for various programming environments is available online.[2]

The reacTable* currently exists in two variations: the concert table, which sports a highly sophisticated and precisely controllable synthesizer for the professional musician. This table setup was used for the first reacTable* concert. The second version has been configured for public installations, with a more playful and popular sounding synthesizer, which was mostly designed for entertainment and educational purposes. This configuration has been shown at the AES and ICMC conferences in Barcelona, ICHIM in Paris and the Ars Electronica Festival in Linz and was received very positively by the audience. The graphics synthesizer also is flexibly configurable through XML configuration files, which allow the simple adaptation of the visual appearance for various installation contexts. As a showcase demo we developed a commercial advertisement installation for a popular brown soft-drink manufacturer.

6.2.2.2 *Learning from Musical Control and Performance*

Various reasons turn real-time computer music performance into an ideal field for the experimental exploration of novel forms of human-computer-interaction:

- It is an environment that combines outstandingly, expression and creativity with entertainment; freedom with precision, rigor and efficiency [43]
- Users are required to have an open but precise and rather complex control over multi-parametric processes in real-time.
- Playing and creating music with the help of digital tools can be a social and collective experience that integrates both collaboration and competition. Moreover, this experience can also be addressed to children.
- Music performance provides an ideal test bed for studying and comparing use and interaction by both dilettantes and experts, both children and adults.

Early and definite examples of this music-HCI synergy can be found for example, in the research and development taken by William Buxton during the 70s and 80s (e.g [44] [45]). We believe that music performance and control (both traditional and computer supported) can constitute an ideal source of inspiration and test bed for exploring novel ways of interaction, specially in highly complex, multidimensional and continuous interaction spaces such as the ones present when browsing huge multimedia databases. In these types of fuzzy interaction environments, exploration can follow infinite paths, results can hardly be totally right or wrong, and the interaction processes involved could be better compared with playing a violin that being reduced to the six generic virtual input devices that constitute the GKS standard (locator, stroke, valuator, pick, string and choice).

2 http://reactivision.sourceforge.net/

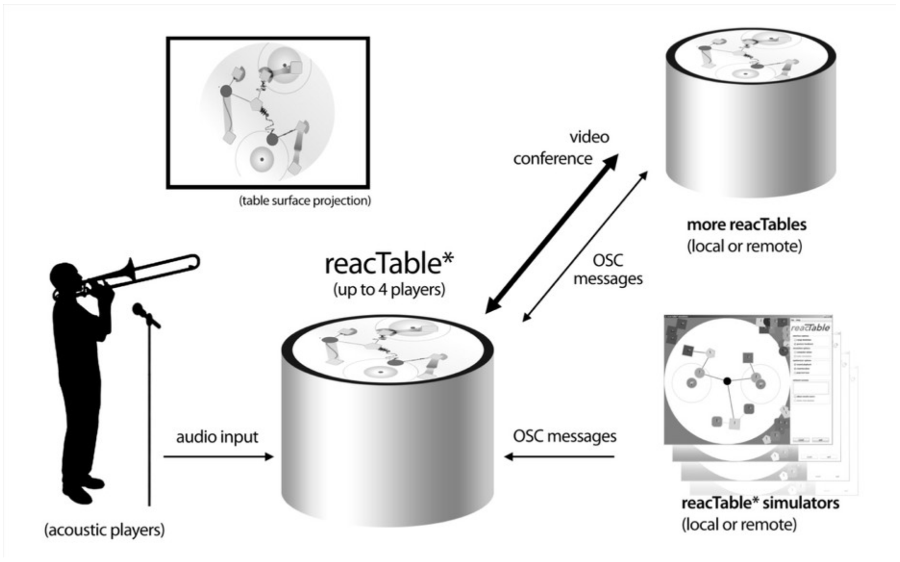

Figure 6: reacTable* collaboration scenarios

6.2.3 Collaborative Music Models

6.2.3.1 *Quick Overview*

There are almost no traditional and just a few contemporary digital collaborative instruments available at the moment. Some traditional Instruments like the piano can be played by four hands although they were not primarily designed for that task. In recent years many musical instrument designers came up with the idea of creating instruments specifically for collaborative music.[46] An early example closely related to the reacTable* is Blaine's Jam-O-Drum [47], a musical installation which encourages visitors to collaboration. In extreme cases, such as the Tooka [48], the instrument only works at all when played by trained and synchronized players.

An illustrative example of a collaborative sound installation is the Public Sound Object [49], which tries to explore several models of remote collaboration in the context of musical practice. The PSO allows network musicians to join a virtual jam session in an abstract musical space, where proxy bouncing ball objects can be remote controlled via a web application. The actual sound synthesis is performed on the installation site and streamed back to the players. Although the PSO architecture has to deal with a significant amount of network latency (time between control action and acoustic result) it overcomes this latency by integrating it as an essential part of the installation. For a more detailed review of collaborative music concept see Barbosa's survey on "Displaced Soundscapes" [50]

6.2.3.2 *Collaborative reacTable* Models*

The reacTable* already had been planned as a collaborative instrument from the very beginning. A table can be considered to be an already culturally defined collaborative space. Tables are places where various people can meet and discuss and where people together can develop their ideas and work on joint projects. Architects work over their plans and models, managers develop their project plans and generals used to move their troops on strategic table models.

These concepts have been widely used and translated to the digital domain by the introduction of Tangible User Interfaces (TUI) [2], where a large group of projects and interfaces as well have been implemented using table interfaces just because of their collaborative nature. Physical objects on a table surface, especially on a round table set-up, are equally accessible for direct manipulation for any participant at the same time.

Figure 6 shows a summary of some possible collaboration models. This includes local collaboration on the same table, and remote collaboration using distant physical or virtual table setups. Additional musical instruments or the audience can collaborate with the reacTable* players on stage.

Local Collaboration

The current reacTable* is a round table with a diameter of exactly one meter providing an active surface with a diameter of 80cm. With this size the ideal amount of simultaneous players ranges from two to four players at the same time. Of course one could imagine even more players being involved but due to the spatial limitations of a table of the current size, the surface of a quarter of a table represents the bare minimum for reasonable playing. For local players two collaboration styles have emerged so far:

- Spatial separation:
  An additive collaboration [46] style, where each player plays in a dedicated zone of his choice, rather defending the territory than collaborating directly with the other players. Each player builds and plays personal instrument patches aside with the other players. The interaction between players basically is similar to that of the members of a band or an orchestra.

- Shared space:
  An multiplicative collaboration [46] scenario, where the players are building and playing their instrument patches together in a real collaborative process. One player can construct a basic instrument, while the second interferes by adding, removing or manipulating additional objects of the patch. Due to the very volatile nature of the dynamic patching paradigm this collaboration has to be planned and performed very carefully in order to maintain a constructive nature.

Additional musical instruments on stage, such as a trombone or violoncello for example, can participate in a reacTable* session by representing their musical output as a sound source on the table. In the most simple scenario the reacTable* players alone have the full control over the further processing and mixing of such a sound source as physically available object on the table. One could also imagine though, that the instrument players themselves can control the position of a virtually present (projection only) sound source on the table, as well as controlling some other control parameters by moving on stage and other additional gestural control performed by the player.

Remote Collaboration

The second reacTable* collaboration scenario involves the connection of two or more table instruments placed at distant locations. In the concert configuration two reacTables one in Austria and one Spain were connected via a conventional internet connection, though technically the maximum number of interconnected tables can be ex-tended easily. Conceptually the two physical table surfaces virtually melt into a single table surface, where players at both locations are playing within the same logical table space. This extends the properties of the local collaboration scenarios we mentioned above to a global context.

Physical objects that are moved on the local table in Barcelona appear as a virtual (projected) object on the remote table in Linz. Both local and remote objects behave the same way, the only difference is that local players of course can not touch or move the virtual objects from the distant table. In a typical collaborative instrument building scenario, a player in Barcelona can place a sound generator object on his table. The object is recognized and located on the surface by the sensor component and a graphical representation of the object is projected underneath the physically present object. At the same time this object's properties data is transmitted to the remote table in Linz, where the same graphical representation is projected at exactly the same position as on the table in Barcelona. Then the player in Linz can place a sound effect on the table surface and after the same process of recognition, transmission and display, the sound effect appears projected on the table in Barcelona. As we will explain in the technical section below, the tables just interchange low-bandwidth control data and no audio data at all. The actual resolving of the dynamic patches and the final sound synthesis are performed independently on both installation sites. Preliminary tests during the development phase showed that under normal conditions with deterministic synthesis objects the resulting musical output was virtually the same on both locations, with minimal differences in the time of appearance and removal of synthesizer modules. Some modules causing a nonlinear behavior such as a feedback object could temporarily lead to significantly different sonic results.

In a concert situation the players themselves are quite aware of the fact that those spooky projected objects are moved by some real human players at a distant location. For the audience though this might not be that clear at all. Hence in order to improve the sense of presence an additional projection of a live video transmission showing the table and the players performing at the remote location proved to be a rewarding addition to the overall experience of this remote collaborative performance.

Regarding the maximum amount of players in a networked session we found that the same rule of thumb as for the local collaborations scenario can be applied: Due to the spatial limitations of the table surface four players are a reasonable maximum. Eventually even four tables with one to two players each would be possible, although not all players are should be active at the same time. The players during the TeleSon concert entered and left at predefined points of the piece, while only during the finale all four players were present at the same time.

Remote Participation

During the early development phase of the reacTable* prototype, the whole dynamic patching and synthesizer infrastructure was designed without an actual physical reacTable*. The use of a software simulator of the complete physical table and its sensor component allowed the rapid development of the basic instrument features without the need of caring too much about possible real world limitations. This software simulator proved also to be quite useful for the composer of the inauguration piece, because it provided a much more convenient infrastructure for experimentation and rehearsal. This software simulator, which actually also includes all the necessary networking infrastructure, has been written in the platform-independent Java programming language as well as the synthesizers also are implemented in cross-platform environments such as *Pure Data* (PD) [51]. This portable design allows an easy distribution and installation of the client software to remote machines. Since the development of the Pure Data browser plug-in [52] by the reacTable* team, even a distribution of an embedded web-application has become possible.

In a typical remote participation scenario, the software simulator clients can join an existing remote collaboration session of one or more physical reacTables. The software simulator fully simulates a complete reacTable* setup and therefore shows exactly the same behavior as a real table, although this simulations cannot provide the interaction flexibility of a tangible interface. Simulator objects equally appear as projected virtual objects on the remote tables; remote objects equally appear in the simulator, but cannot be moved by the users. Again the dynamic patch generation and the actual sound synthesis are fully performed within the local simulator software.

In an alternative participation scenario, some members of the local audience at a reacTable* concert who are equipped with a PDA or modern smart-phone could download a stripped down version of the Java table simulator and then control a few virtual synthesis objects on stage via a wireless network or Bluetooth connection. We have developed a preliminary prototype for an off-the-shelf PocketPC handheld computer by porting the existing software to this platform and adapting the interface to the limitations of a small touch-screen interface.

We are also currently working on another setup where a second tangible interface is sending pitch or temporal control to a connected reacTable*. The scoreTable* basically is a sequencer, where objects placed on the table trigger musical events when they are detected by a radar-style sweep. Additional players can compose musical or rhythmical patterns that directly interact with the synthesizer objects on the reacTable*.

6.2.4 TeleSon Invention #8

During the preparations for the *International Computer Music Conference* 2005, which took place in Barcelona and was organized amongst others by the *Music Technology Group* it was decided to commission the composition of a piece for the reacTable* from a professional composer. As the result of an official competition *Chris Brown* was chosen to write and perform this piece for the inauguration concert of the ICMC 2005. Brown has been closely involved in the final development of the reacTable* and his constant feedback during the development of the piece and the synthesizer provided valuable input for the final instrument mappings itself.

The resulting piece TeleSon, a composition for two reacTables and four players was finally performed twice in a networked performance between Austria and Spain. *Chris Brown* and *Günter Geiger* were performing in Barcelona, while *Martin Kaltenbrunner* and *Marcos Alonso* were playing in Linz. The first concert was the actual ICMC inauguration concert, which took place on Sunday, September 4th in the premises of the SGAE in Barcelona and at the *Interface Culture* exhibition in Linz. The second concert was performed the following Monday, September 5th between the *Galeria Metronom* in Barcelona and the *Ars Electronica Centre* Sky Media Loft, and was attended in sum by around 600 persons at both locations.

6.2.5 Networking Infrastructure

Networked reacTables interchange their objects' ID, location and orientation by transmitting UDP packages via a conventional IP network using the TUIO [38] protocol which is based on Open Sound Control (OSC) [27] messages. UDP assures the fastest transport and the lowest latency method, while TUIO provides the necessary redundancy to guarantee a stable and robust communication. Connected tables just pass on their raw control data, which they receive from the sensor component, without transmitting any audio data at all. The resolving of the synthesizer patches and the actual sound synthesis is done locally at each installation site, which reduces the impact of possible latency problems to a minimum. Each client just treats the control data from objects of a remote table the same way as from the ones on the local table. We assign for example the IDs `1` to `n` to a set of `n` tangible objects on our local table. Any table in the network is expected to use the same set of objects with the same functional properties. As a consequence we can define the total set of objects in this network session by multiplying number of local objects by the number of ta-

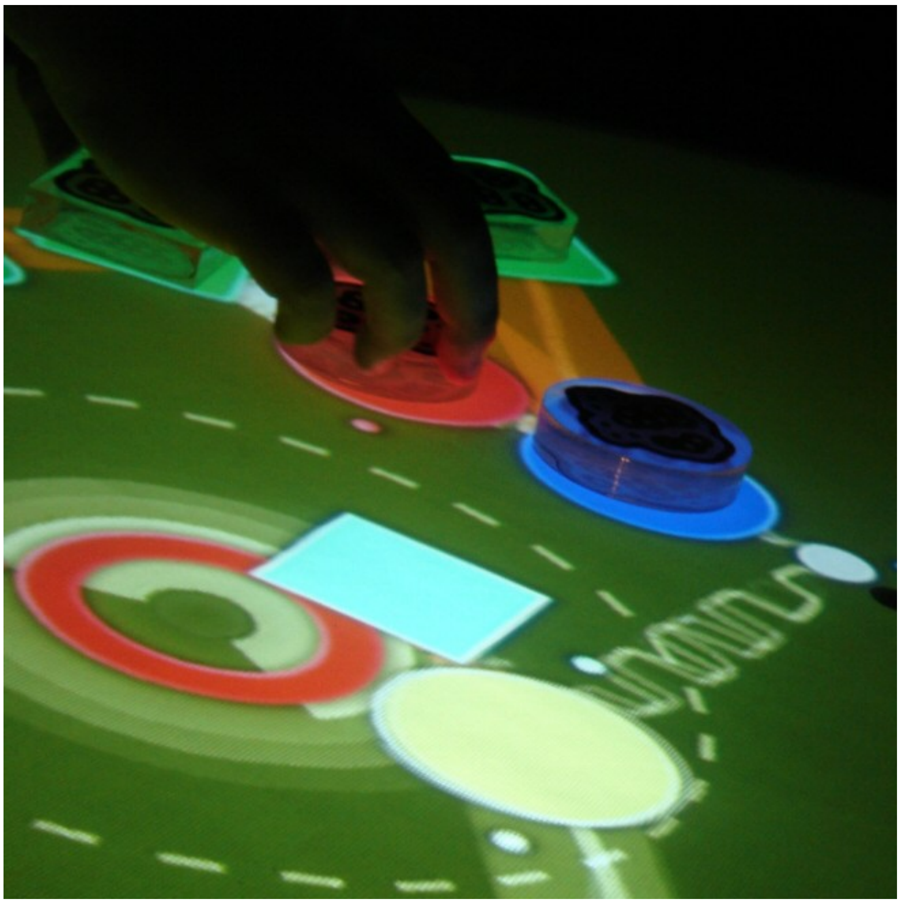

Figure 7: Local and remote synth objects

bles in a network session. A second remote table then for example appears with IDs from `n+1` to `2*n` for its set of tangibles. Adding another remote table to the session just increments the number of total objects by n in our example.

Basically tables are connected in a peer-to-peer network topology: Each table sends its control data to any table that has been configured to participate in a network session. Yet to facilitate the connection procedure between tables and to overcome potential connection problems caused by routers or firewall configurations, the reacTable* networking infrastructure is using Ross Bencina's *oscgroups*.[3] This software package consists of a server and client component. The server component informs any connected client that is forming part of a predefined group about the connection details of all other clients that belong to the same group. The oscgroups client component then broadcasts any outgoing OSC message to the remote clients of all currently known group members, and passes all incoming OSC messages to the local OSC application. Oscgroups is very portable and was tested under Win32, MacOS X and Linux operating systems and was recently ported to the Windows Mobile platform as well.

The reacTable* management component in general just reflects the incoming OSC messages from the reacTIVision vision engine component, but adds a supplementary tag that identifies the source with a

3 http://www.rossbencina.com/code/oscgroups

sufficiently unique string. This message is an actual extension to the TUIO protocol which does not appear in the original paper. It became necessary because OSC itself does not provide any information about the source of the OSC messages and additionally the oscgroup client appears as a local source anyway.

Therefore we add a new message to each bundle which follows the format `/tuio/2dobj source username@host`. This message allows the reacTable* client software to assign each OSC bundle to the correct origin.

6.2.6 Acknowledgments

Many people and institutions have been involved in the preparations of the reacTable* concert. First of all we would like to thank *Chris Brown* for the composition of this exciting musical piece and his valuable input during the reacTable* development. Without any particular order we would also like to thank the *Phonos Foundation*, the *Galeria Metronom* and the *SGAE* in Barcelona as well as the *Ars Electronica Center* and *Christa Sommerer* from the *Interface Culture Lab* in Linz for their great support.

The authors specially would like to thank the former team member *Ross Bencina*, who made several crucial contributions to the final reacTable*. Without Ross' brilliant implementation of the computer vision component and his contribution to the OSC infrastructure the reacTable* would not be the robust instrument it is today. We also would like to thank the former interns *Ignasi Casasnovas* and *Gerda Strobl* for their valuable work for this project.

6.3 remarks & analysis

This publication is documenting the particular collaborative design aspects of the Reactable, and the experiences gathered from the first public performances with our initial research prototype, which took place in September 2005 after roughly three years of research and development. In this and other historical publications the instrument is commonly referred to as *reacTable**, although this peculiar camel-case naming scheme today has been abandoned since the commercialization of the Reactable in 2009.[4]

6.3.1 Contemporary Music Practice

The principal design objective for the Reactable was the creation of a novel musical instrument, which allowed electronic music performances overcoming the interaction deficits of traditional synthesizer hardware or complex music software, while maintaining the feature set of a professional instrument. During the past decades the contemporary electronic music practice has largely converged to using laptop-based software interfaces in live performance. While these devices offer near endless possibilities of musical expression, they also provide a rather limited interaction bandwidth for expressive performance. Therefore we adopted the tangible interaction paradigm in order to leverage the full potential of a human performer when interacting with physical representations instead of a virtual interface.

6.3.2 Designers, Composers & Performers

As a team of *Digital Luthiers* (electronic instrument designers) our goal was not only to design a device for mere research purposes, but to create an actual musical instrument that can be used live on stage. While the first development phase until 2005 was largely dedicated to the design and implementation of the core instrument functionality, we spent the years after our first public concert refining the instrument literally on stage.

> According to Buxton[53] "*there are three levels of design: standard spec., military spec., and artist spec.: Most significantly, I learned that the third was the hardest (and most important), but if you could nail it, then everything else was easy.*"

While gathering our own experiences from performing our instrument on stage, we also decided to collaborate with other professional artists in order to gather external feedback in addition to our own technical and artistic viewpoints. This was also the main reason for our collaboration with Chris Brown, who was commissioned to compose the piece, which was then performed at our first concert. This *designer - composer - performer* relationship was crucial for the success of the Reactable, which culminated in our collaboration with Björk[54].

4 `http://www.reactable.com`

6.3.3 Physical Embodiment

The core concept of the Reactable is a translation of the modular synthesizer paradigm to the tangible domain. Visual signal-flow programming languages such as Pure Data already introduced virtualized graphical representations of the various sound generators, filters and controllers found in a typical modular synthesizer. The Reactable basically brings this virtualization back to the tangible domain through the physical embodiment of these synthesizer components into simple tangible objects. Therefore the Reactable can be conceptually also considered as tangible sound programming language, which allows to establish and manipulate the sound-flow through the relation and orientation of graspable objects that are distributed on top of its interactive table surface. Although the actual sound is synthesized in a computer, this conceptual notion of sound, which appears to be embodied into physical objects, is further emphasized by the visualization of the sound-flow between the connected synthesizer objects. Visual programming languages on a virtual graphical user interface also have certain advantages, such as the instant loading of preset configurations or the creation and manipulation of sub-patches, such functionality is limited in the physical domain due to the persistence of tangible objects, which actually depend on the manipulation by the user. We resolved some of these limitations through the constitution of a *Dynamic Patching Paradigm*[24], which basically defines a simple ruleset for the connection of our tangible synthesizer objects, based on type, distance and availability. This also determines that the instrument patches on the Reactable have to be built and played at the same time.

6.3.4 Multi-User Interaction

Although the Reactable is often referenced as a *multi-touch instrument*, is has to be emphasized that the actual iteration of the instrument, which is presented in this paper didn't even implement any touch functionality at this point. The version we presented during the first public concert was purely based on tangible object manipulation without any interaction with the surface itself, which only provided the sound visualization and connection feedback. Touch functionality was of course added soon after, since it allowed the control of further sound parameters around each object in addition to the mere manipulation of object position and orientation. Although the introduction of touch-interaction then also allowed additional gestural control of the individual sound dynamics or the cutting of sound flow with a simple stroke gesture, up to today the Reactable does not implement any multi-touch or multi-stroke gesture at all. Therefore the possibility of concurrent object manipulation and multi-touch interaction primarily facilitates the collaborative multi-user interaction scenarios, which we discussed in this paper. This parallel access to all available resources also constitutes a *Direct Physical Manipulation* paradigm.

6.3.5 Networked Performance

Apart from the various local collaboration scenarios described in this publication, one of its most significant contributions was the design and implementation of a networked performance configuration, which was largely designed around the UDP based remote communication capabilities of the TUIO protocol.

Considerable network latencies are a major obstacle for the realization of remote musical performances and improvisations, rendering it difficult or rather impossible to establish a functional musical collaboration by transmitting real-time audio streams. Therefore we chose to transmit only the interaction data, which (although it suffers from the same network delays) has been applied to control a more consistent and synchronous local synthesis on both sides. For this configuration the reacTIVision instance running locally underneath each of the two Reactables, transmits all of its detected object states, positions and orientations not only locally but at the same time to the second distant instrument. Since both tables share the same interactive object configuration, but only have half of the tangibles physically available on-site, this configuration can be conceptually considered as a single Reactable with two control interfaces. As mentioned earlier, the advantages of TUIO/UDP protocol are its low bandwidth requirement and fast transmission latency, while its major disadvantage is the required compensation method for possible (and in our performance also certain) packet loss. From our experience in this critical performance situation, we can now say that the required protocol robustness in order to achieve a consistent model view on both sides has been met entirely. Preparing this performance, we also defined the mentioned TUIO source extension, in order to allow for a proper distinction between the local and remote instances. This extension has been later included in the official TUIO 1.1 revision, which is documented in the following paper.

6.3.6 Restaging Teleson Invention #8

After a meeting with Chris Brown past summer, we are now after 12 years considering restaging a reconstruction of his original composition. Since the Reactable hardware platform and its original software platform has been redesigned and significantly improved since, this will also require an adaptation of the piece. While first the original Reactable prototype from Barcelona is unfortunately lost, we already refurbished its second iteration, which was created for the performance in Linz for this purpose. Eventually the piece should be performed again in its networked configuration connecting Linz and San Francisco, which had not yet been repeated since its original performance between Barcelona and Linz in 2005.

6.3.7 Musical Application

Since its initial conception in 2003 the Reactable synthesizer had been primarily designed as a feature complete tangible application for live musical performance. Most of its underlying hard- and software technology therefore had been developed for this demanding musical application scenario with its specific requirements for performance and stability. While other tangible interaction platforms mostly represent a technical exercise which usually include simple demo applications showcasing its capabilities, the Reactable today still represents one of the few working examples of a comprehensive real-world tangible-tabletop application.

The public interest that was generated after our collaboration with several artists and the following Ars Electronica[5] award, also resulted in a growing demand for an actual commercialization of the instrument. After the further improvement of the Reactable hardware for third-party usage and its installation at various science centers and digital art museums around the world, it eventually became impossible to meet this demand with the infrastructure of a small university research group. Therefore in 2009 our team of four Reactable creators finally decided to establish a spin-off company with the support of the Pompeu Fabra University.

This company Reactable Systems[6] since then has been dedicated to the further hard- and software refinement of the Reactable, leading to the design and implementation of the Reactable Live system, which today can be considered as version 2.0 of the original Reactable. This instrument provides a more mobile hardware configuration and more versatile software features, and has has been sold more than a hundred times until today. In order to reach a larger audience our company recently also diversified the synthesizer concept of the Reactable into a mobile application, which can be controlled through simple capacitive tokens on a tablet surface.

6.3.8 Tangible Platform

Apart from its principal musical application scenario the Reactable hardware obviously can be also employed as a generic tangible interaction platform, where the reacTIVision and TUIO framework provide its basic software development kit for the design and implementation of custom applications. Although the Reactable hardware provides a distinctive round form factor and an outstanding mobility (the instrument can be packed into two travel bags), our company has decided to concentrate on the development of the actual musical instrument rather than providing a generic tangible hardware platform only. This decision had been also supported by the massive intrusion of major companies such as Microsoft into the market of interactive tabletops, which left little room for smaller start-ups in that area.

5 http://archive.aec.at/prix/#12462
6 http://reactable.com/

7 REACTIVISION AND TUIO: A TANGIBLE TABLETOP TOOLKIT

7.1 INTRODUCTION

The forth publication of this cumulative thesis discusses various improvements to the reacTIVision framework as well as an extended version 1.1 of the TUIO framework, and is also representing an **evaluation of the initial Tangible Abstraction Framework**. The following paper has been reformatted and corrected without any changes to its original content.

7.1.1 Context

This publication has been presented at the ITS2009 *International Conference on Interactive Tabletops and Surfaces*[1] which took place from 23-25 November 2009 in Banff, Canada.

Following the two initial separate publications on the TUIO protocol and the reacTIVision application, this paper represents a first reflection on the initial iteration of the tangible interaction framework in a combined perspective. Apart from the documentation of the various improvements to the reacTIVision engine, including a more robust heuristic fiducial tracking approach as well as a proper multi-touch implementation it also provides a detailed performance evaluation of the framework.

After an analysis of the emerging third-party implementations of the TUIO protocol, the paper also proposes an extension to the original specification, which adds a third *blob* profile, providing an additional geometry description of the tangible interface components. Following the experiences gathered from the previous networked performances, an additional *source* message was added in order to allow the multiplexing of various TUIO trackers.

Furthermore this paper also identifies the several shortcomings of the original TUIO specification, which leads to the rudimentary proposal of the next generation TUIO 2.0 protocol, intending to provide a more comprehensive feature set for the representation of state-of-the-art tangible interaction platforms.

The discussion section after this paper will present the more recent development of the reacTIVision framework, which in its current version 1.6 finally implements the TUIO 1.1 specification presented in this paper. The actual TUIO 2.0 specification and the according definition of an extended tangible interaction framework is then discussed in the following final chapter of this dissertation.

1 http://www.acm.org/its/2009

7.2 ORIGINAL PUBLICATION

7.2.1 Abstract

This article presents the recent updates and an evaluation of reacTIVision, a computer vision toolkit for fiducial marker tracking and multi-touch interaction. It also discusses the current and future development of the TUIO protocol and framework, which has been primarily designed as an abstraction layer for the description and transmission of pointers and tangible object states in the context of interactive tabletop surfaces. The initial protocol definition proved to be rather robust due to the simple and straightforward implementation approach, which also supported its widespread adoption within the open source community. This article also discusses the current limitations of this simplistic approach and provides an outlook towards a next generation protocol definition, which will address the need for additional descriptors and the protocol's general extensibility.

7.2.2 Introduction

The TUIO protocol and reacTIVision framework comprise a toolkit for the rapid development of tabletop tangible user interfaces and multi-touch surfaces. Both components have been initially developed for musical applications in the context of the reacTable [37] project, a tangible modular synthesizer based on an interactive table surface. After the presentation of this instrument and also Jeff Han's multi-touch demos based on FTIR [40] had created considerable public interest in gesture-controlled surfaces, the TUIO protocol was eventually adopted by several open source initiatives with the goal to reverse engineer large multi-touch surfaces. Access to such a variety of freely available tools based on a shared protocol supported the democratization of the emerging tangible and multi-touch user interface technology. Since their initial publication and release in 2005, TUIO [38] and the reacTIVision toolkit have been successfully used for the design and implementation of numerous research, commercial and hobbyist projects, supporting the widespread adoption of the tangible interaction paradigm.

7.2.3 Tangible Surface Abstraction

The initial goal of the TUIO protocol definition was to provide a simple description of pointer and token states in the context of a two dimensional table surface, where pointers are defined as untagged points with normalized Cartesian coordinates, while tangible tokens provide an additional identification tag and rotation angle. Although this is a very simplified view of an interactive surface context, this description provides a basic solution for the implementation of multi-touch surfaces and the tracking of tagged physical objects. Such a basic model has of course its limitations, which became even more evident with its adoption within other application areas as well as with the further development of the reacTIVision engine itself. We will discuss these limitations and the consequent future extensions to this model further below.

After evaluating existing alternatives for the controller context [55], the TUIO protocol was based on Open Sound Control (OSC) [27], which has been widely adopted for the encoding of control data from musical instruments and general-purpose interactive devices. OSC successfully intends to overcome the performance limitations of the musical standard MIDI protocol, specifically regarding its bandwidth and data resolution, hence allowing for a more fine-grained control of advanced musical instrument designs. On the other hand the open approach of OSC compared to MIDI, makes it more difficult to interconnect arbitrary controller systems, therefore OSC based protocols such as TUIO need to define a clear semantics of the specific usage scenario within a separate message name space. TUIO in this case defines a range of profiles for the description of token and pointer state changes. Although OSC itself does not specify a default transport layer, most implementations including TUIO, are currently based on the delivery of UDP packets, which allow the necessary low latency delivery over commonly available local, wired or wireless IP networking infrastructure.

The initial application scenario for the TUIO protocol was defined by the interchange of control data between two or more table interfaces, which had been constructed for the first series of reacTable concerts, where four players were performing on two instruments located in different cities. Therefore the protocol design needed to be fast and robust enough for a musical performance over a standard Internet connection. Since the transmission of natural events – such as adding, moving and removing objects – could cause inconsistencies when certain events, most importantly the remove messages, are lost during transport, the protocol structure was specifically designed to stay consistent even when used on a fast but error prone UDP channel. Hence TUIO implements a state model instead of transmitting events, all currently active token and pointer identifiers are transmitted within each message bundle, which allows the continuous reconstruction of add and remove events on the receiving side by comparing the local and received identifiers.

The specification of such a descriptive network based protocol suggests the design of a distributed architecture, separating the tracking sensor component from the actual user application. This distributed approach enables the interoperability of various sensor technologies, platforms and programming environments. Apart from earlier considerations regarding the limited processing power of a single CPU system, which nowadays have become less important with the advent of powerful multi-core processors, another motivation for choosing this architecture was the use of the framework for teaching purposes. Dealing with students from different backgrounds and with varying technical skills, ranging from engineers to artists, providing a collection of TUIO client implementations for programming languages such as C++. Java and C# and more importantly multimedia authoring tools such as Processing, Pure Data, Max/MSP, Quartz Composer and Flash, allowed the involved students to concentrate on the actual interface design task using the most appropriate tool.

7.2.4 The Reactivision Engine

Since its last open source software release[2] and the previous publication of its general functionality [56], the reacTIVision engine has undergone major feature and performance improvements. In addition to the significant improvement of the overall symbol tracking robustness, the recently published public version 1.4 also supports basic multi-touch finger tracking. While the initial versions of reacTIVision only performed the direct tracking of amoeba style fiducial symbols, which have been specifically developed in conjunction with the fiducial tracking core libfidtrack, the latest release introduces various tracking layers, which significantly enhance the symbol tracking performance. This is especially important in conditions with fast moving objects due to expressive gestures in musical performance.

7.2.4.1 *Fiducial Tracking*

The principal fiducial tracking method used within reacTIVision is based on the analysis of region adjacency graphs, originally derived from Costanza's d-touch concept [26]. After applying a local adaptive threshold to the original camera image, the resulting binary image is then segmented into a graph of adjacent black and white regions. Hence the identification of the amoeba symbols is based on a dictionary search of previously defined tree structures that are encoded into the marker topology, and the actual symbol layout carries additional information, which allows the precise calculation of the symbol centre point and its rotation angle [35].

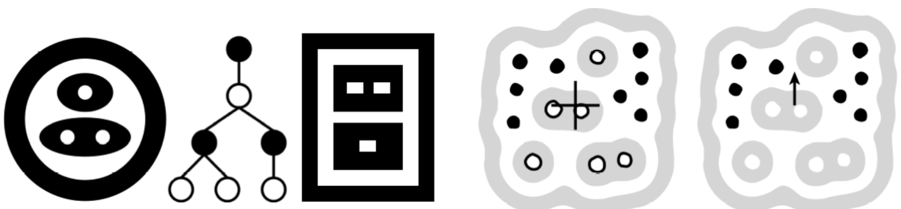

Figure 8: a) fiducial tree and two possible representations
b) encoded center point and angle information

Since the symbol structure allows an almost arbitrary representation of the actual geometry, we used a genetic algorithm [36] for the automatic generation of optimally shaped fiducial symbols, which eventually resulted in the organic amoeba appearance of the presently used fiducial marker collection. This genetic algorithm is driven by a fitness function that selects the generated symbols based on their size, symmetry and position and angle precision. The symbol position is

2 http://reactivision.sourceforge.net/

calculated from the average of all leaf nodes, while the orientation vector points from the center point to the average of all black leaf nodes only. The current default set is for example defined by 18 nodes within a tree with maximum depth of two layers, which results in a possible range of 128 sequences, from which only 108 symbols have been selected to meet the minimum size and precision requirements.

The limitation to dedicated tree spaces, with a clearly defined node count and tree depth ensures the overall tracking robustness, since it is rather improbable to find these complex tree structures within arbitrary image noise, which limits the probability of finding false positives. We also separate the currently used alternative symbol collections by at least three nodes in order to avoid wrong symbol identification due to erroneous image data.

On the other hand this strict analysis is prone to minor changes of the symbol structure, such as addition and loss of individual leaf nodes, which can often appear in noisy or blurred images. While in these cases the algorithm is still capable of identifying the presence of a fiducial symbol in general, the identification of the individual symbol has become impossible, since the actual tree structure has been broken. Nevertheless we use the presence of unknown fiducial symbols within a secondary fuzzy fiducial tracking layer, where we simply assign unidentified erroneous symbol structures to the ones previously tracked nearby, which helps to improve the total symbol recognition rate.

Fast expressive movements, which are very common within musical performance, unveil the limitations of optical tracking methods. Problems such as motion blur can only be partially resolved with shorter camera exposure times and stronger illumination. Since these parameters are limited, very fast object movements yield a blurry fiducial image and hence result in a complete destruction of the fiducial structure, making it impossible for both the standard and fuzzy tracking method to identify an actual symbol. Therefore a third layer is tracking the position of the root node region, the usually white fiducial background. With the knowledge of the previous fiducial position and the displacement of the region centre from the actual symbol centre, the position of fast moving fiducial markers can be updated very accurately using just the root node tracking method. To summarize, the trajectory of fast moving objects, can be tracked accurately with a combination of the three methods outlined above, where the symbol can be tracked in all individual frames without additional filtering methods. Currently we are allowing a single frame without tracking result, which we are using to calculate the correct speed and acceleration updates before the object is finally removed from the list if not found in the following frame. Since the actual position during this single frame is not updated, we are planning to introduce an additional Kalman filter [57] in order to estimate the position of the lost symbol, which then also can be reassigned more easily to a nearby root region.

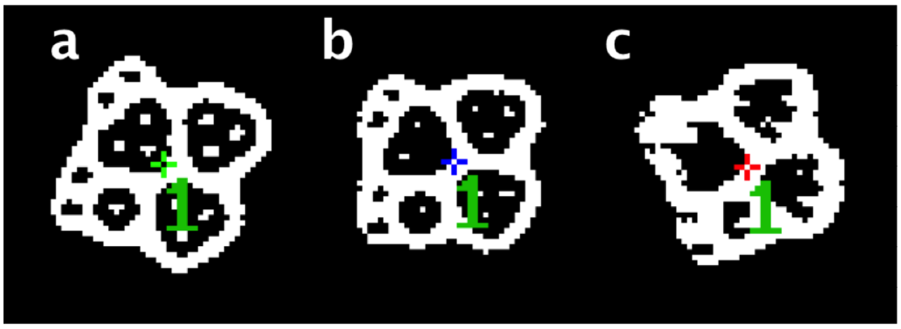

Figure 9: The three tracking modes a) full b) fuzzy and c) root region, depending on the retrieved symbol quality

7.2.4.2 *Finger Tracking*

The complementary multi-touch tracking layer introduced with the latest reacTIVision release takes advantage of the existing image processing infrastructure, without introducing a significant CPU overhead for this additional task. We are simply retrieving all white region candidates with a given size from the available image segmentation data and calculating the error comparing the candidate region to a round region prototype. The average finger size and maximum error can be adjusted within the application, yielding good tracking results in well-adjusted conditions. Compared to sole multi-touch trackers, reacTIVision is required to maintain the full fiducial structure intact, and therefore cannot afford the application of destructive image filters such as Gaussian blur in order to smooth the finger blob contour. Since this approach does not introduce any additional or parallel image filtering in order to enhance the source image, the initial configuration task of the camera settings and illumination environment has to be done more carefully than with comparable multi-touch only solutions. On the other hand this combined method ensures a low latency performance for musical applications, while providing simultaneous fiducial and finger tracking within the same image processing thread. The currently used tiled local adaptive threshold [58] method yields good results, and further improves the performance by neglecting tiles with a gradient below a configurable value. Unfortunately this method introduces square artifacts around low contrast regions, which can degrade the finger tracking accuracy. In order to improve the initial image quality we are currently evaluating alternative local adaptive threshold algorithms though, which should equally meet the requirements of the marker and blob tracking tasks.

7.2.4.3 *Blob Tracking*

With the following release, we introduce an additional generic blob tracking layer, which is also taking advantage of the existing computational infrastructure, by selecting white regions within a given, configurable size range from the available segmentation data structures, while previously detected finger and fiducial root regions are excluded. In order to avoid additional image processing tasks, these

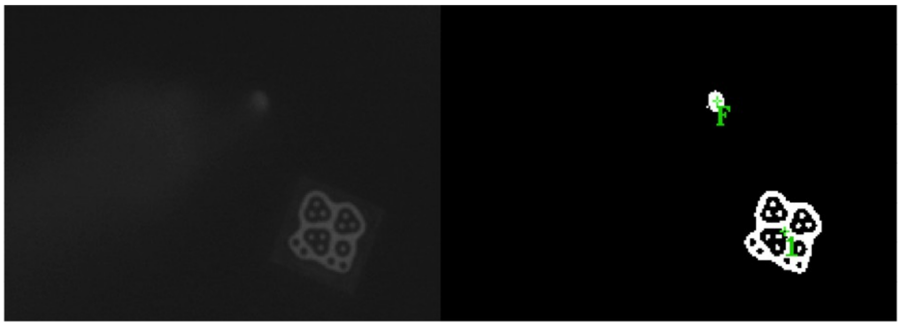
Figure 10: Original camera image and binary threshold image with finger and fiducial tracking feedback

regions are already encoded into a linked list of line spans during the segmentation process, which also annotates the final pixel area of each region. This data representation allows the reconstruction of the region contour and area without additional analysis of the actual source image itself, which again avoids additional processing overhead for this complementary tracking task. The span list implicitly encodes the full blob contour information in a compact format for further processing. The derived list of contour points can be efficiently reduced to the outer (and inner) blob contour, and consequently to a simplified list of contour points, which describes the overall blob geometry in sufficient detail. Finally for each of these retrieved regions, the oriented bounding box is calculated, which is providing an approximate description of its position, size and orientation. Current reacTIVision development builds already implement these basic geometry descriptors for untagged objects, which as a consequence have been also included within a third additional blob profile in an updated revision of the TUIO protocol, which we will describe in more detail below. The additional and more detailed geometry descriptors will be included in a future TUIO 2.0 specification though.

7.2.4.4 *Amoeba Symbols*

In addition to the updates to the core tracking software described above, some significant improvements to the fiducial symbol layout and rendering have been implemented, which enhance the overall tracking performance in boundary conditions such as low camera resolutions, reduced symbol sizes or increased surface distance, all of which result in a smaller size of the symbol in the actual camera image.

The number of symbols provided with the default set has been increased from the original 90 amoeba symbols to a total of 108 usable symbols out of the possible range of 128 within the described tree space. An improved fiducial generation algorithm, which introduces – already during the generation process – the final selection rules based on the symbol size and orientation vector length, yielded 20% more usable symbols that met the imposed criteria. By applying a high penalty to symbols that did not meet the initial size and vector boundaries, the genetic algorithm also converged much faster towards a usable symbol, thus reducing the overall generation time.

The newly created symbol set had its minimum orientation vector length improved by almost 5%, which generally supports the more robust calculation of the symbols' rotation angle. Also the maximum symbol size has been reduced by more than 10%, while as well showing a more uniform and narrow size distribution.

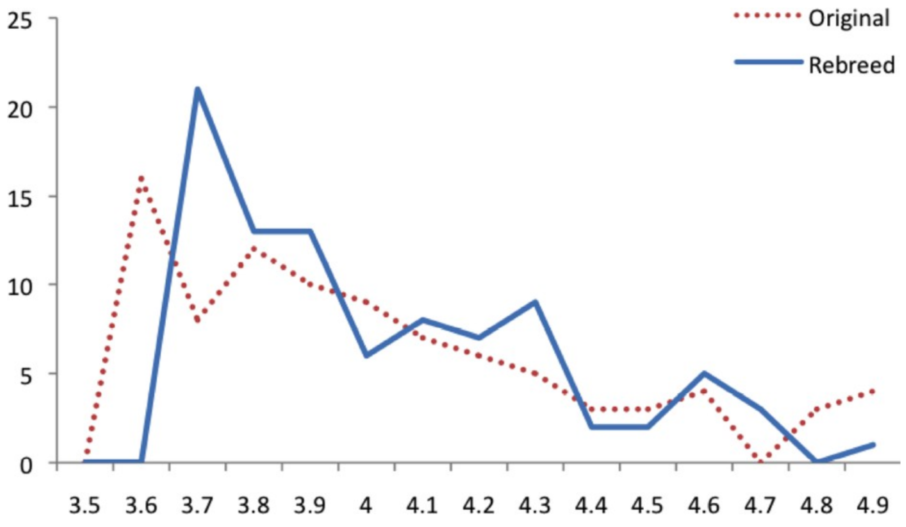

Figure 11: Distribution of the orientation vector length

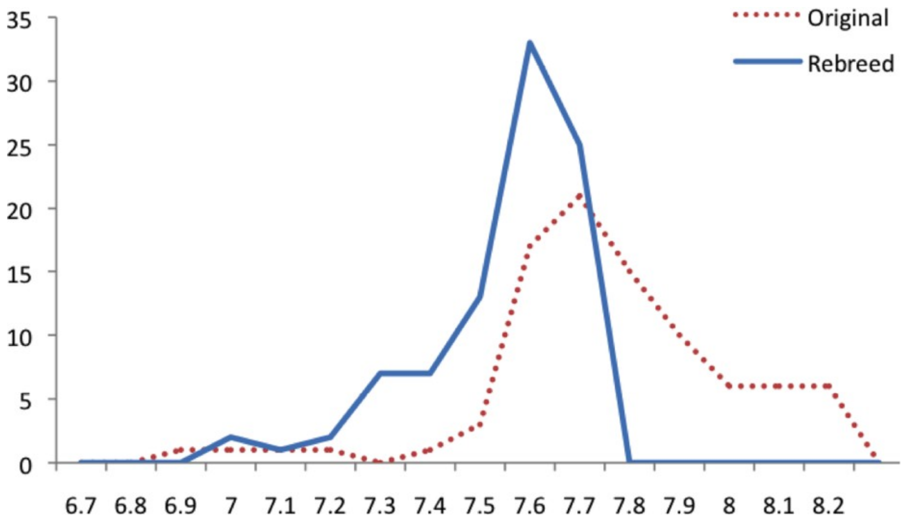

Figure 12: Distribution of the symbol footprint size

We also introduced an alternative set of smaller symbols from a different tree space with 12 nodes only. While this tree space contains 15 possible variations, we selected 12 symbols that met the size and vector length criteria. This additional set can be used for applications that only require a limited set of different symbols IDs, but has the advantage of a reduced maximum symbol size by another 10% compared to the default amoeba set. In general, reacTIVision allows the usage of any arbitrary set of tree sequences, although we currently only provide these two subsets for the moment. Additional symbol collections can be added with a simple configuration file, but we recommend separating the selected tree spaces by at least three tree nodes, in order to avoid wrong symbol identifications. It should be also considered, that smaller symbol sets will not provide sufficiently precise position and rotation information, while these simpler tree configurations are also more likely to be found in arbitrary noise, and are therefore prone to yield false positives compared to the carefully selected standard symbols. Alternatively there are already third party fiducial generators available, which allow the generation of alternative symbols of any desired tree size.

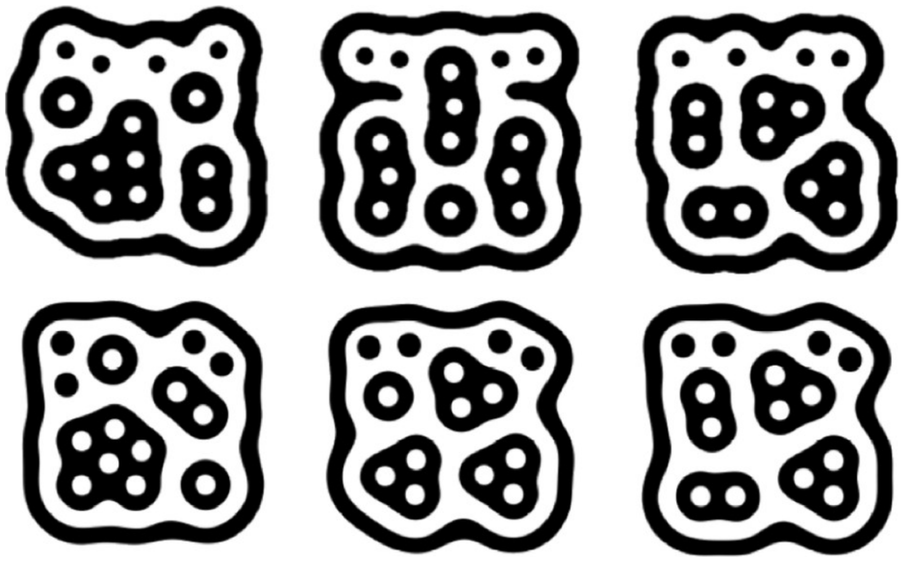
Figure 13: comparing the original symbol rendering (top row) to the improved rendering method (bottom row)

An improved graphical rendering method for the amoeba symbols introduces relatively enlarged leaf node proportions, which also support a better tracking performance of small scale or distant markers. Finally this new symbol set has been released in a vector based PDF format, which allows a high quality printing of the marker collection in any arbitrary size.

In order to increase the number of available marker IDs, the collection of 108 plus 12 standard symbols has been doubled though the addition of the inverted symbols with an initial black root node, resulting in a total number of 216 plus 24 standard symbols that are delivered with the current public release. This total of 240 distinguishable default markers should be sufficient for most application cases, considering that the individual marker IDs can also be repeated within the same context.

7.2.4.5 *Performance Evaluation*

The combination of the three marker tracking methods yield a satisfactory tracking performance even with fast moving objects, provided the camera and illumination settings are optimally configured with an appropriate short exposure time, which also guarantees a low latency image acquisition. The following chart illustrates the improvements in these boundary conditions, where the symbol structure is partially destroyed due to motion blur caused by expressive object handling as shown above.

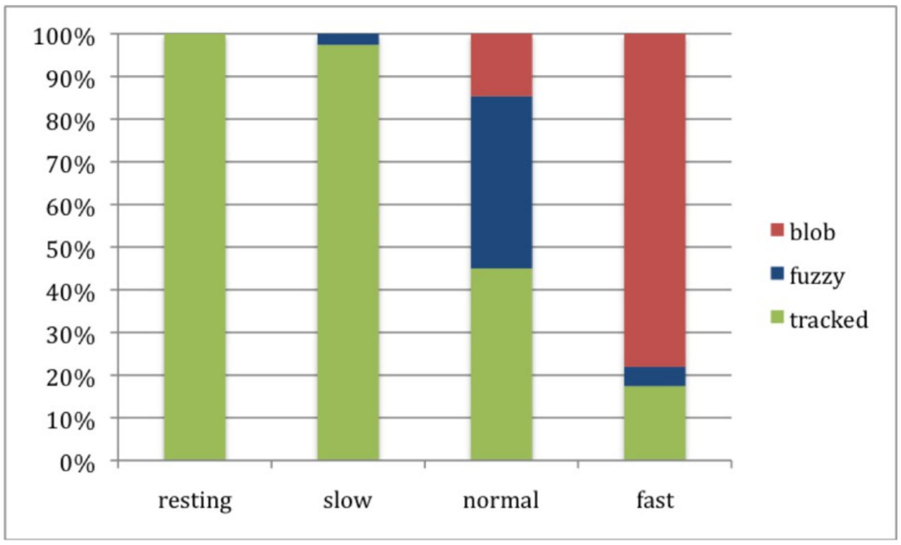

Figure 14: Tracking modes at varying speeds

An evaluation of the tracking accuracy of resting symbols showed a standard deviation of around 0.05 pixels for the symbol centre point and a standard deviation of around 0.5 degrees for the rotation angle. These results allow for single pixel position accuracy and a rotation angle accuracy of three degrees in normal conditions, without significant jitter problems. We expect to improve the tracking accuracy for the rotation angle with the introduction of an additional Kalman filter component.

Performance measures of the latency of the current image processing chain depend on various factors, such as the camera resolution and resulting buffer size, the platform, compiler and CPU speed of the test system. Evaluating the frame latency on a 2GHz Core Duo Macbook on Linux, the median processing, analysis and delivery time for a VGA sized frame ranges around 5ms, which results in an acceptable system latency considering the complexity of the task. The following table illustrates the evolution of the processing latency for fiducial tracking adding the additional finger and blob tracking layers.

	ICC 11.0	**GCC 4.4**
Fiducial tracking	4.8 ms	5.6 ms
Fiducial & touch tracking	5.0 ms	6.0 ms
Fiducial & touch & contour	5.4 ms	6.5 ms

A more detailed analysis and review of the tracking performance for various conditions such as fiducial size, and comparison with other marker systems would unfortunately exceed the limits of this article and will be therefore addressed in a subsequent publication.

7.2.5 The TUIO Protocol

The original TUIO protocol specification was concentrating on the specific needs of the reacTable project, mainly focusing on tagged object and finger tracking in the context of a remote collaboration scenario, while ensuring the overall robustness of the networked distributed system. During the development and feature enhancements of our own tracking application, as well as with the integration of the TUIO protocol into further projects, several issues regarding missing features within the present profiles and the need for additional protocol extensions emerged. The extension of the existing message structure needs to be planned carefully though, considering the stability of all current implementations that rely on solid shared protocol definition.

7.2.5.1 *Original TUIO Specification*

A TUIO profile defines two central messages: set messages and alive messages. Set messages are used to communicate information about a token's state such as position, orientation, velocity and acceleration. Alive messages indicate the current set of tokens present on the surface using a list of unique session IDs. Additional fseq messages are defined to tag each frame update with a unique sequence ID. At typical TUIO bundle is therefore typically comprised of at least three messages, while the set messages can be accumulated in order to fully use the available space of a UDP packet. TUIO messages are commonly delivered to UDP port 3333 in the default configuration, although alternative transport methods are equally allowed. Please note that the following clear text message representations are only for demonstration purposes, the actual OSC message is transmitted in a compact binary format.

```
/tuio/[profile] alive [active session_IDs]
/tuio/[profile] set [session_ID attributes]
/tuio/[profile] fseq [int32]
```

There are two basic profiles for the description of pointers and tokens (here cursors and objects), which are commonly used within the context of a 2D surface. There exist additional profiles for 2.5D environments, which include the distance to the surface, as well as 3D environments, which also provide 3D rotation information for the object profiles. All profile types are generally designed to describe the surface or the space above an interactive table environment. Most currently available TUIO implementations are concentrating on the 2D profiles though. The specific set message syntax for the cursor and object profiles include attributes such as position, velocity and acceleration and is structured as following:

```
/tuio/2Dobj set sid id xpos ypos angle xvel yvel rvel macc racc
/tuio/2Dcur set sid xpos ypos xvel yvel macc
```

7.2.5.2 *Updated TUIO 1.1 Specification*

In order to provide a smooth transition path we are introducing an intermediate and backwards-compatible TUIO 1.1 specification, which adds two new features to the existing protocol specification, without breaking the existing client implementations: A third profile for the description of untagged objects and the possibility of multiplexing multiple tracker sources.

The complementary blob profile allows the further distinction between identified tagged symbols and unidentified plain blob objects, which are also providing basic additional geometric information.

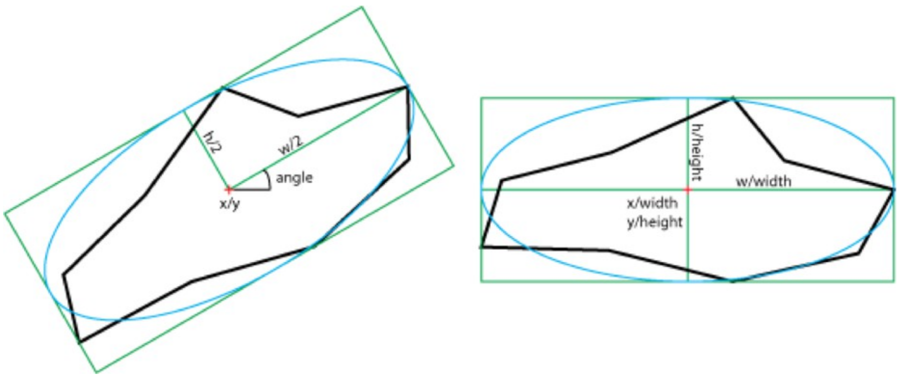

Figure 15: Simple description of a blob enclosure

```
/tuio/2Dblb set sid xpos ypos angle width height area xvel yvel
    rvel macc racc
```

The profile's set message format describes the inner ellipse of an oriented bounding box, with its center point, the angle of the longer axis, its width and height as well as the blob area. Hence this compact format describes the approximate elliptical blob enclosure, which also allows the reconstruction of the oriented bounding box. The blob area is normalized by pixels/width*height, providing quick access to the overall blob size. The blob dimensions are defined as normalized values after performing an inverse rotation by -angle.

```
/tuio/[profile] source [name@address]
```

In order to allow the multiplexing of several TUIO trackers on the client side, an optional source message can be transmitted within each TUIO bundle, which enables the identification of the bundle's origin. The name@address argument is a single string that specifies the application name and any unique source address.

An issue, which currently cannot be addressed within a backward compatible TUIO extension, is the lack of reliable timing information. Assuming that OSC does already provide a sufficient time tag within the actual bundle header, we decided to not include a redundant time tag into the TUIO message structure. Unfortunately OSC implementations interpret the bundle time as a delivery time, which in some cases could cause the OSC layer to drop bundles with an earlier time stamp. The current TUIO implementations partially intend to compensate this with the inclusion of velocity and acceleration attributes within the set message structure.

Since several TUIO developers are working with Actionscript, the need for an alternative communication model for Flash, which currently does not support UDP sockets has emerged. The presently used workaround, which expands TUIO/OSC messages to an XML format that can be interpreted by Flash, is not comparable to the good performance results delivered by the common UDP transport method. As an alternative, TUIO can support an additional TUIO/TCP mode or a Flash local connection, which can be used for connecting to this kind of closed environments. The transparency of alternative transport methods is an advantage of the chosen OSC encoding.

7.2.5.3 *Future TUIO 2.0 Specification*

It has become clear that even with the intermediate protocol extensions the current simplistic approach is by far not sufficient for the description of a generalized tangible interaction environment. While TUIO 1.1 already addresses the basic needs for an additional descriptor for the object geometry, the strict separation of cursor, object and blob profiles is one of the mayor limitations for the future protocol extensions. Also, the existing object profile is lacking the possibility of transmitting marker content data while the cursor profile is missing important attributes such as cursor ID and pressure data. Finally TUIO is also missing detailed timing information, which unfortunately cannot be retrieved from the OSC bundle time tag as originally intended. The number of potential enhancements and changes to the current protocol structure justifies the introduction of a new TUIO 2.0 protocol specification, which eventually will resolve the shortcomings and design limitations of the current protocol generation.

As a consequence TUIO 2.0 will allow a more substantial update for the existing TUIO infrastructure. A flat and more extensible profile structure, which also integrates better into the overall OSC bundle and message formatting, will allow future incremental message updates which can add additional descriptors and functionality. Token, Pointer and Geometry messages can now be handled in parallel and can share the same session ID if these are actually referring to the very same object. Therefore for example Pointer messages can be extended with a bundled Bounds message, which transmits the actual geometry in addition to the generic pointer information. The basic geometry descriptors, which are similar to the format introduced in TUIO 1.1, can be incrementally extended with additional messages describing the Contour, Skeleton or full Area of the described region. It depends on the capabilities of the tracker implementation or the actual application setup. Tokens will carry an additional type ID, which allows the multiplexing of various symbol types within a session. A Token can be extended with more detail by an optional Symbol message that encodes the information about the actual marker type and content, which will allow the introduction of alternative marker types, such as data matrix (or QR) codes or RFID tags. Pointers will include various additional attributes, such as pointer ID and type ID as well as pressure and region of influence, and can be also extended with optional Control messages, which allow the encoding of additional control dimensions from buttons, wheels, knobs or sliders. Finally the new TUIO protocol generation will also allow the description of object associations, such as container relationships or mechanical connections between individual objects. This for example allows the encoding of token-constraint systems as well as constructive object assemblies, which will extend the overall encoding possibilities from the purely spatial approach of the original specification and will therefore extend the scope of the protocol to support a broader range of tangible user interfaces.

Although there are recent developments towards the clarification of the OSC bundle timing with the introduction of a revised OSC 1.1 specification [59], TUIO 2.0 will add a redundant time tag to the fseq message of each frame, providing fine-grained timing information, which is necessary for correct gesture analysis. Since TUIO is based on OSC, any implementation can already choose to define its private message space in order to transmit custom controller data. TUIO augments this possibility with the definition of a Custom message syntax, which allows associating these custom attributes to the existing TUIO objects. There have been suggestions to introduce a back channel for the configuration of the tracker environment, but there are currently no plans to abandon the simplicity of the current unidirectional protocol approach.

The TUIO 2.0 specification draft is already close to its finalization, although we will wait until the consolidation of the intermediate TUIO 1.1, before we will start with the implementation of this next generation protocol. In order to support the migration towards the new version, the according client implementations will support both protocol generations.

7.2.5.4 *Third Party Tuio Implementations*

During the early stages of the TUIO development, the available tracker and client implementations were limited to the reacTable and similar environments, for which we initially designed this protocol. The first external project, which picked up the TUIO protocol as an abstraction for multi-touch interaction, was the touchlib library by David Wallin. This was also one of the first publicly available multi-touch tracking applications, since reacTIVision only implemented the touch functionality at a later point. Since then, a growing number of multi-touch and tangible interaction platforms have implemented the TUIO protocol, which lead to its more widespread adoption. A recently established community website provides detailed information about the current and future TUIO specifications, implementation notes for the development of TUIO enabled software as well as a growing list of client and tracker applications that support our protocol.[3] Please also refer to this website for further information about the projects mentioned in the following software selection.

3 http://www.tuio.org/

7.2.5.5 *TUIO Trackers*

The currently available tracker implementations mostly include multi-touch software based on computer vision, such as touché, BBTouch and Community Core Vision (formerly tBeta). Further TUIO tracker implementations are based on controller hardware such as the Wiimote controller device, where WiimoteTUIO for example allows the rapid development of Whiteboard applications using only the IR tracking capabilities of a Wiimote controller and a suitable TUIO client application. In addition to that, there exist TUIO bridges for dedicated multi-touch hardware, such as the devices from N-trig, which are presently used for most available multi-touch tablet PCs. Similar integration initiatives have been started for Windows 7 and the Microsoft Surface, which have been extended to provide TUIO support at the system level [60]. Finally there also exist a variety of iPhone applications, which allow the usage of this hand-held device as a remote multi-touch controller that can send the TUIO over its wireless network connection. It has been shown that especially for this application case, the TUIO state model proved to be very robust on this error prone channel.

7.2.5.6 *TUIO Clients*

Apart from the primary TUIO client implementations, which are available for most mainstream programming languages and multimedia environments, the community contributed a large collection of additional TUIO implementations for several other environment that were not directly support by ourselves. This includes programming languages such as Objective C, Python, Smalltalk, Ruby and Actionscript as well as sound and media environments such as VVVV, SuperCollider, Chuck or Open Frameworks, and there are also several higher-level programming environments for gesture recognition and tangible interface development for Java, C# or C++ available, that are using TUIO as the common input layer. Based on the TUIO client reference implementations, which basically decode the touch and object events from the protocol, there is also a growing number of end user applications available, taking advantage if these input events as an alternative controller interface. Applications such as NASA World-Wind, Google Earth, Second Life or the Blender game engine have been enhanced with multi-touch control with the help of the TUIO protocol.

Most recent versions of mainstream operating systems such as Windows 7 and Mac OS X 10.6 already include system level support for multi-touch input. Within the X-Window system, which is commonly used on Linux operating systems, the multi-pointer X-Server MPX [61] has recently been included into the main branch and will therefore soon become a standard component of all major Linux distributions. We are currently involved in the integration of the TUIO framework into MPX through the development of the xf86-input-tuio driver component, which will allow the seamless integration of the existing tracker software and libraries into the operating system infrastructure.

7.2.6 Conclusions And Future Work

We have shown and documented the improvements and current functionality of the reacTIVision framework and provided an outlook to the future blob geometry extension that will be included within the next public version, which is also reflected within an intermediate update to the TUIO 1.1 protocol. In parallel there is ongoing work towards the definition and implementation of a future TUIO 2.0 protocol, which will hopefully provide a solid base for the realization of more versatile interactive surface environments. The future work on the tangible interaction framework will also shift the focus to a more generalized view of the overall TUIO platform, where reacTIVision will serve as a common reference implementation for the newly defined protocol features, which intends to open the further development to third party implementations based on alternative technologies.

Robust tracking performance regarding speed, latency and reliability are even more important in the context of expressive musical performance. Although our improvements have shown to be suitable for live performance conditions, optical tracking systems have also clear limitations regarding their temporal resolution. State of the art industrial cameras can deliver images at frame rates between 60-200 Hz, hence while we are adding further features to our tracking engine, we will ensure that frame rates up to 200 Hz can be processed in real time and at a reasonable latency. Apart from the common image analysis, we are currently evaluating the incorporation of dedicated GPU programming methods, which promise to deliver improved image processing performance. This approach will provide a responsive controller for musical performance and will allow the implementation of more fine-grained musical control gestures, such as performing a vibrato or the tapping of a rhythm. Additionally we are also looking into alternative sensor methods, which can augment and improve the overall input performance.

7.2.7 Acknowledgments

This research has been conducted in the context of the reacTable project, which is a team effort and has been developed in collaboration with my colleagues Günter Geiger, Marcos Alonso and Sergi Jordà within the Music Technology Group at the Pompeu Fabra University in Barcelona, Spain. The reacTIVision fiducial tracking engine and amoeba symbols have been developed in collaboration with Ross Bencina, and I'd also like to thank Enrico Costanza for his initial contribution to its development, as well as Till Bovermann for his participation in the TUIO initiative, which had been partially supported by the European Commission Cost287 ConGAS action.

7.3 remarks & analysis

The publication presented above, documents the various feature and performance improvements that had been provided by the now historical reacTIVision version 1.4, which was released in May 2009. Since then I have released one major update to reacTIVision 1.5 in October 2014, followed by a revised reacTIVision 1.5.1 release in May 2016. I am currently working on the finalization of the next major reacTIVision version 1.6, which will represent the final reference implementation of a feature complete TUIO 1.1 tracker application. In the following the most significant improvements that will be included in this upcoming version will be discussed, which should be due to official release soon after the submission of this thesis. The source code of this currently already rather stable development version can be retrieved from its public Github repository.[4]

7.3.1 Platform Updates

After five years without any publicly available updates to reacTIVision, the existing application binaries unfortunately started to show several compatibility issues on newer operating systems and with more recent digital cameras. While the core fiducial tracking engine and multi-touch tracking features were fully operational, a major update to the overall operating system support as well as for the camera acquisition layer became necessary. Apart from updates for the feature set of more recent OS versions, such as then MacOS X 10.8 and Windows 7, now there are also binaries for the 64bit versions of these two major platforms since version 1.5. Since these previously already had been available for the Linux version, the 64bit binaries also showed notable performance advantages on MaxOS X and Windows without many necessary changes to its multi-platform code base. With the availability of popular single board computers such as the Raspberry Pi, I also added some minor tweaks to the Linux port in order to support those ARM based embedded computing platforms. Additional updates became also necessary to support the Samsung SUR40 (Microsoft Pixelsense) platform[62] under Linux.

7.3.2 PortVideo Library

The overall architecture of the PortVideo camera acquisition layer, which has already been described earlier has seen two major updates. reacTIVision 1.5 provided an updated AVFoundation camera driver on MacOS X, as well as an improved DirectShow driver for Windows, which greatly improved the overall camera performance on these systems, also providing higher frame rates for newer digital cameras with compressed image formats such as MJPEG. This also added native support for the still popular PS3eye camera, and for affordable industrial cameras implementing the IIDC-over-USB protocol.

4 http://github.com/mkalten/reacTIVision

The next version of PortVideo currently developed within reacTIVision 1.6, has seen a major overhaul of its internal architecture, providing a much improved camera configuration interface on all platforms, as well as the convenient enumeration of available camera formats and features, in order to allow interactive camera selection and configuration. Along with the upcoming release of reacTIVision, the PortVideo layer will be released again as a separate LGPL licensed library along with multi-platform example projects, allowing its integration into third-party computer-vision projects.

7.3.3 Blob Analysis

One of the major design objectives of any feature extensions for reacTIVision was to avoid unnecessary CPU overhead caused by expensive computer-vision algorithms. Therefore our implementation of the additional Blob tracking feature - introduced with the TUIO 1.1 specification update - intends to use the already existing high-level data structures from the image-segmentation that is employed for our fiducial tracking algorithm. This includes the region-adjacency graph, which provides an ordered access to all available image regions (or blobs), along with a span-line representation for each region[35].

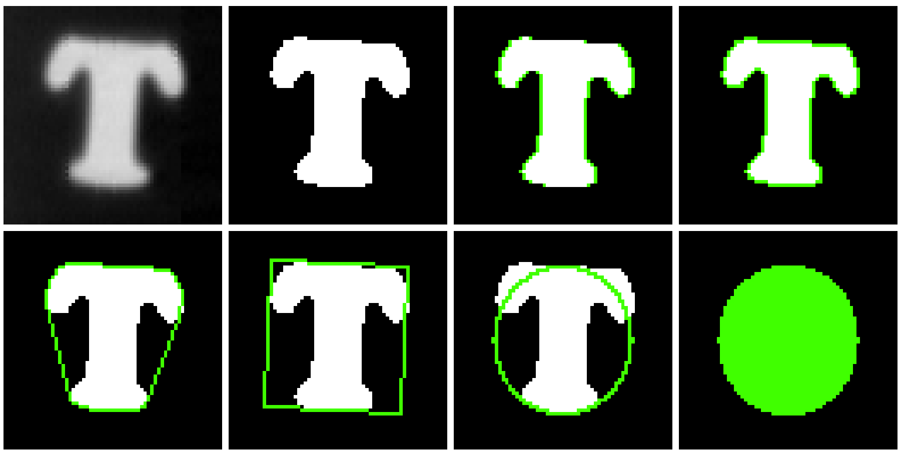

Figure 16: Blob analysis: a) source b) threshold c) spans d) contour e) convex hull f) bounding box g) enclosure h) abstraction

In order to retrieve candidate regions for our blob analysis, we therefore simply have to traverse this region-adjacency graph, and select all white-coloured regions within a determined size range, which is already associated to each region. From these selected regions, we now have direct access to a linked span-list, which constitutes the full area for each individual blob, by marking the start- and end-points for each span as well as a link to the following span. Traversing this span list, we can efficiently reconstruct a compact list of the relevant outer contour points of each candidate region. From this outer contour list, we then calculate a convex hull, which serves as the basis for the calculation of the four corner points of an oriented bounding box. This oriented bounding box is finally described by its center point, along with its width, height and rotation angle.

7.3.4 Finger Tracking

Based on the blob analysis method described above, it became also possible to improve the performance and precision of the finger-tracking algorithm. In order to detect finger blobs, we first select all white regions within a configurable size range, matching the average footprint of a finger blob. After analyzing each finger blob candidate with the method described above, we can now calculate the deviation from all outer contour points from the enclosing ellipse defined by the oriented bounding box. This not only allows the robust exclusion of malformed false-positives, but also greatly increases the precision of the position tracking of each finger, compared to the center point of a simple bounding box, which was used in previous versions.

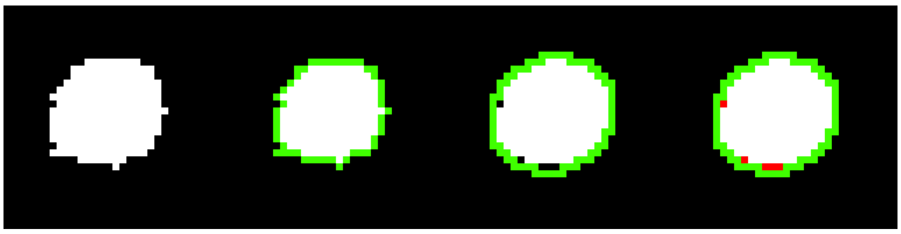

Figure 17: Touch blob analysis: a) raw blob b) outer contour c) enclosing ellipse d) accumulated error

7.3.5 Yamaarashi Symbols

While the previous reacTIVision version 1.5 was mainly dedicated to resolve various platform and camera compatibility issues, the current version 1.6 also provides significant feature updates to the core tracking components. In addition the plain blob tracking and analysis described above, this also includes a new type of fiducials, which I named Yamaarashi (see below). Although our current amoeba fiducials still provide the core marker tracking engine for reacTIVision due to their robustness, this method also has shown several shortcomings. These are mostly based on the growing complexity of the region adjacency graph and the resulting symbol size, when introducing larger ID sets based on region adjacencies. Our current standard fiducial set included with reacTIVision provides 216 amoeba symbols, which are usually printed with a diameter of 5-6 centimeters when used with a standard camera resolution. reacTIVision also includes an additional smaller set of fiducials which work fine at an average diameter of 4 centimeters, but are limited to 24 different IDs only. On the other hand, providing a set with a larger number of more than 1000 symbol IDs would grow the necessary symbol size to 7-8 centimeters. Also the generation of an optimal set of larger symbols would require a significant computing and selection effort based on our genetic fiducial breeding algorithm and rendering method.[36]

For the design of the new Yamaarashi symbols I therefore chose a hybrid approach, by combining the robustness of the core amoeba fiducial design with an additional bitcode allowing the encoding of a larger number of different symbol IDs. Therefore employing a very simple and relatively small amoeba fiducial (w012211) is employed at the center of each symbol, with a ring of individual code bits that are arranged around this circular core symbol. In the current configuration we can arrange a total number of 24 bits around each symbol, from which we actually use 20 bits to encode 1.048.576 different IDs, while employing the additional bits to encode a simple checksum in order to eliminate possible false positives.

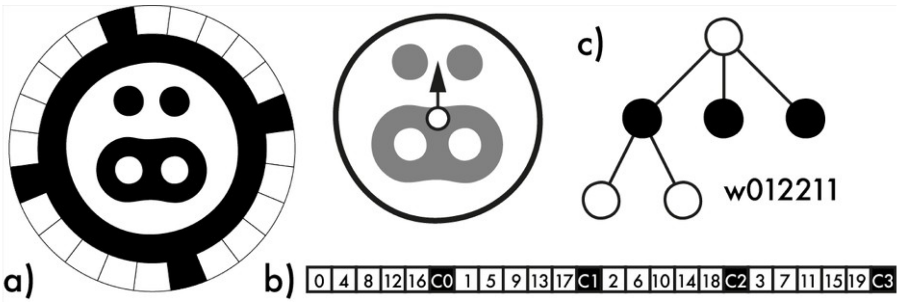
a)
b)
c)
w012211
0 4 8 12 16 C0 1 5 9 13 17 C1 2 6 10 14 18 C2 3 7 11 15 19 C3

Figure 18: Yamaarashi symbol structure:
a) overall design b) bitcode sequence c) core symbol topology

The core fiducial is therefore first recognized together with all other amoeba symbols from the standard fiducial set through libfidtrack. This step already provides its position and rotation angle, along with its specially reserved Yamaarashi ID. In order to then decode the actual symbol ID for each individual Yamaarashi, we then also analyze the enclosing ellipse of the central symbol's white root region, in order to calculate and compensate a possible distortion of the detected symbol. With the provided orientation of the core amoeba symbol combined with the actual size of the root region, we can now simply decode and verify the circular bitcode by determining the actual colour at each predicted bit position. Due to the simplicity of the core fiducial topology, the libfidtrack engine may detect this rather simple structure in arbitrary noise, therefore we are further using the heuristics about the geometric relationship of the root node to its leaf nodes in order to eliminate possible false positives, which is further supported by the 4 bit checksum based on a simple XOR operation applied to five 4 bit segments of the total 20 bit code, which most likely fails when decoding the arbitrary surroundings of such a false positive.

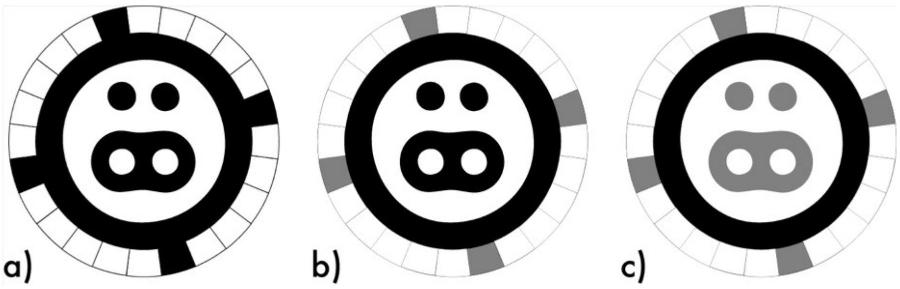

Figure 19: Yamaarashi tracking modes:
a) bitcode identification b) core amoeba c) root region

In analogy with the multi-level fiducial tracking heuristics which is already applied for the tracking of amoeba symbols, I also introduced several levels for the more robust tracking of Yamaarashi symbols, which are required in certain conditions of reduced image quality, fast motion blur or partial symbol occlusion. The full symbol decoding level is necessary to determine the actual symbol ID at the beginning. Once decoded we can further track the Yamaarashi core symbol without the additional need of decoding its actual ID within every frame. Finally we are still able to maintain a stable and precise symbol tracking based on the circular root region, even if the bitcode and/or core symbol may not be decoded in an individual frame.

Although reacTIVision already includes a PDF file containing an optimized set of amoeba fiducials, the large number of more than one million Yamaarashi symbols would exceed a reasonable file size with its more than 50.000 pages. Therefore I created a simple tool based on Processing[5], which allows the rendering of an arbitrary subset of Yamaarashi symbols to a custom PDF file. This tool also allows the adjustment of the actual symbol rendering size, defaulting to a diameter of 4 centimeters in conjunction with an average camera resolution, which represents only two thirds of the standard amoeba fiducial size.

Since the standard amoeba fiducials have a rather unique aesthetic appearance, it was our objective to maintain this particular aspect also within our design of the new set of Yamaarashi fiducials. Since I found that the central symbol structure loosely resembled a comic animal face, while the circular bitcode could be interpreted as hairy spikes, I decided to name our symbols with the Japanese word for porcupine. Hence the new Yamaarashi symbol design has achieved its initial design goals by providing a larger number of IDs combined with a smaller and more uniform footprint, while maintaining the overall robustness and particular aesthetic aspects of the original amoeba design.

7.3.6 Performance Improvements

As already mentioned in the paper, one of the long-planned feature enhancements for reacTIVision was the introduction of a proper filter method in order to reduce the possible noise for the position and orientation tracking. Due to the multi-node averaging we implemented for our amoeba symbols, our fiducial tracking algorithm generally provide sub-pixel accuracy for position, and a one-degree accuracy for rotation. Since in some circumstances, such as in bad lighting conditions or when using very small symbols with low-resolution cameras, noisy position and angle data would cause occasional jumps of resting fiducials, a one-pixel position and three-degree angle threshold was introduced as a simple work-around. While this appeared to be stable enough and sufficient for most application scenarios, it also caused some degree of surprise for other researchers, who even-

5 http://www.processing.org

tually tested reacTIVision in comparison with their superior fiducial tracking methods.[63] I therefore now integrated Casiez' et.al. 1€ Filter[64] instead of the previously considered Kalman[57] Filter, since this simple method provides stable results with acceptable latencies, while preserving the original sub-pixel position and one-degree angle resolution.

Although many computer vision environments already employ GPU based processing methods such as OpenCL in order to decrease CPU usage and improve the overall processing speed through parallelization, reacTIVision still relies on standard C++/C code to maintain its current cross-platform portability. In order to further improve the speed of the rather CPU intensive computer-vision portions though, multi-threaded image processing was added wherever possible, such as within the FrameThresholder component, which was quite easy to adapt for parallel processing. This most importantly improved the overall performance on slower single board computers, by leveraging the full potential of their multiple ARM cores. In this context is is noteworthy that an old iBook G3 laptop (single-core 32bit PPC, running at 700MHz) is able to process a 640x480 VGA camera image at 30fps, while in comparison a current Odroid C2 board[6] (quad-core 64bit ARM, running at 1.7GHz) we can process a 1280x720 HD camera image at 60fps in real-time. Therefore the installation of reacTIVision on these embedded systems allows the construction of compact, low-cost and high-performance smart cameras, providing TUIO 1.1 through their Ethernet port.

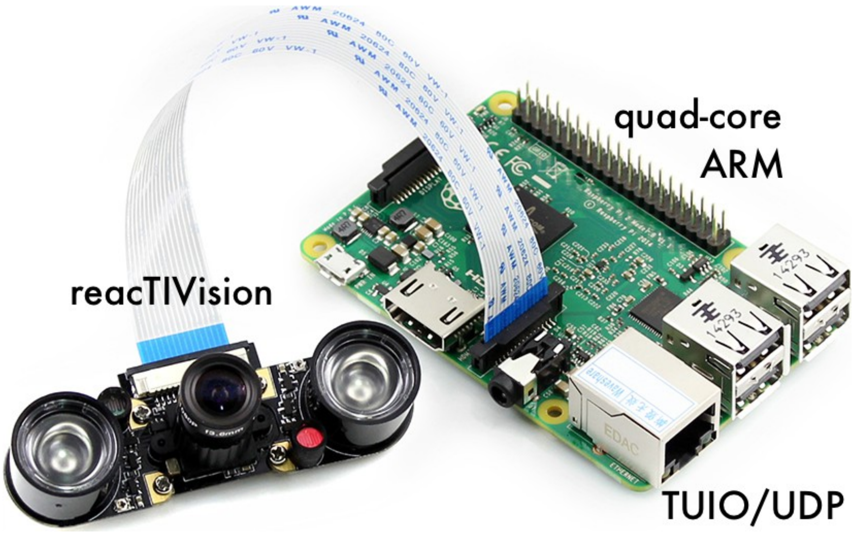

Figure 20: An embedded TUIO camera system configuration.
Photo: aliexpress.com with annotations

For this purpose, reacTIVision now also supports headless operation for the configuration and use without graphical user interface. Therefore the internal architecture of its core image-processing components has been also cleaned up from any previous dependencies to the SDL interface, also facilitating its future port to other platforms, such as iOS and Android.

6 http://www.hardkernel.com

7.3.7 Library Integration

Due to its historical development process reacTIVision still provided an internal custom TUIO implementation until its most recent 1.5 series. Along the internal architecture changes within reacTIVision 1.6 it finally also became possible to integrate the standard TUIO 1.1 reference implementation in the form of a cross-platform C++ library. This library not only defines the necessary feature complete TUIO server API, which encapsulates all TUIO components such as objects, cursors and blobs, but now also allows the alternative use TUIO/TCP, TUIO/WEB and and TUIO/FLC transport channels in addition to the standard TUIO/UDP method. While the TUIO/FLC transport for Flash developers is still maintained, today the more standard-compliant TUIO/WEB web-socket method is recommended for the development of TUIO-enabled web-applications. Due to the library integration the optional MIDI output had to be dropped, and needs to be provided by more versatile external TUIO-to-MIDI conversion tools, such as OSCulator.[7] Another benefit of the increased modularization is the parallel use of multiple transport channels, such as sending TUIO/UDP to various machines or ports. In order to take advantage of the new source multiplexing feature introduced with TUIO 1.1, we also added the optional source message configuration in order to allow the execution and distinction of multiple reacTIVision instances on a single machine, which can be used for the realization of multi-camera configurations. The standard library integration will eventually also facilitate the switch to the next TUIO2 generation within a future reacTIVision 2.0 version.

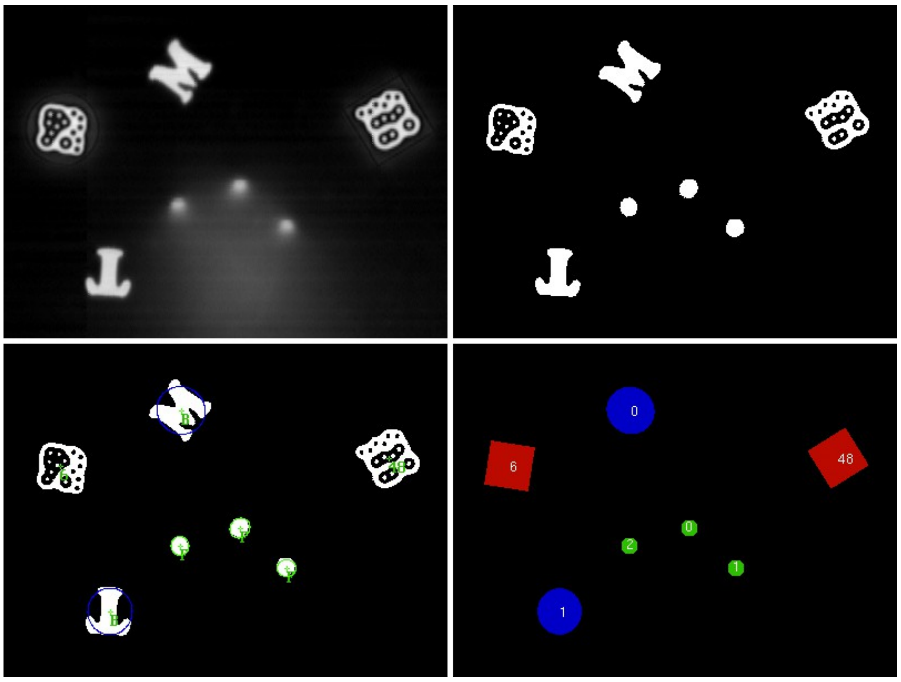

Figure 21: TUIO 1.1 abstraction: a) raw sensor input b) image processing c) object, cursor & blob recognition d) TUIO client representation

7 http://osculator.net/

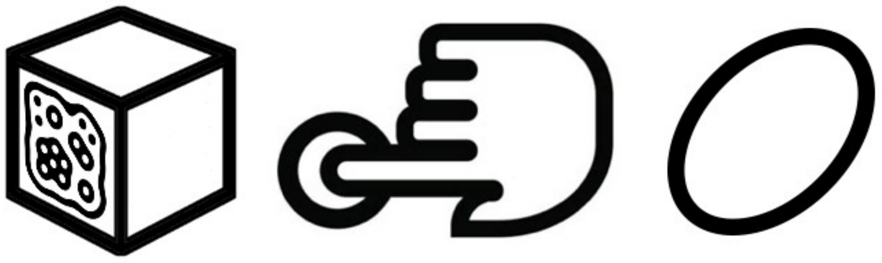

Figure 22: TUIO 1.1 components: objects, cursors & blobs

7.3.8 TUIO 1.1 Model

Since the historical development of the current TUIO specification has been documented throughout the various publications which have been presented in this thesis, I'd like to summarize to overall architecture of the abstraction model that is represented by the final TUIO 1.1 specification. The original TUIO 1.0 specification defined two principal interface components: **Objects** represent simple tangible tokens, which can be identified through 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, are referenced through their surface position. Since neither objects or cursors provide any information about their actual geometry, I defined **Blobs** in order to specify the additional spatial extension for objects and cursors, as well as for generically untagged physical objects. TUIO 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.

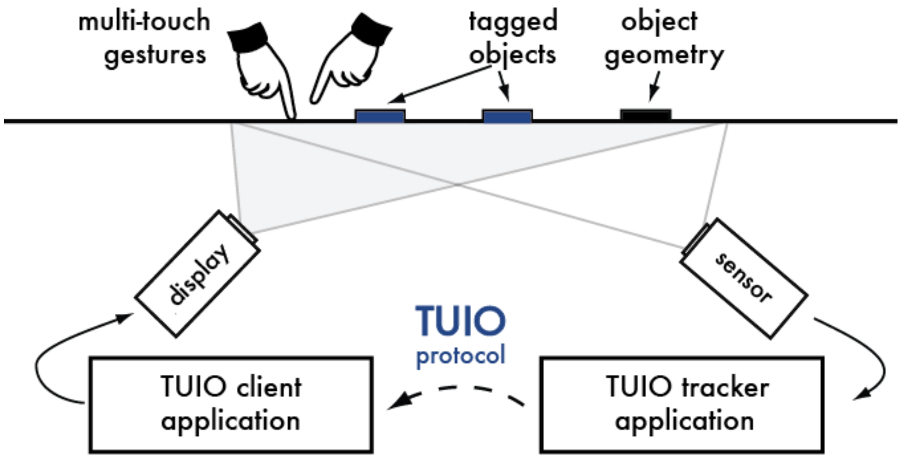

Figure 23: TUIO 1.1 architecture

The distributed architecture of the TUIO framework is based on the separation of the hardware abstraction layer provided by TUIO tracker for a particular sensor platform, from the actual application layer through the integration of a TUIO client. The TUIO protocol and its encapsulation of abstract component descriptors connect the physical surface environment to the digital application layer, by providing a continuous representation of all present component states.

This separation of the sensor component from the application layer facilitated the development of a distributed TUIO ecosystem, which led to the adoption of the TUIO protocol as de-facto standard for the integration of multi-touch tangible interfaces. TUIO hardware trackers today include an impressive variety of camera based systems, optical touch frames, capacitive touch foils as well mobile phone and tablet platforms. Apart from TUIO client implementations for numerous programming languages, several professional application programming environments provide native TUIO support: This includes Kivy for Python, TuioFX for Java, Qt for cross-platform GUI development, the Unity 3D game engine as well as several creative coding platforms such as Processing, OpenFrameworks, Cinder and TouchDesigner just to name a few of the most relevant TUIO-enabled environments.

7.3.9 TUIO 1.1 Limitations

This publication also openly discusses the actual limitations, which emerged through the lifespan of the current TUIO abstraction model and its representation within the TUIO 1.0 protocol syntax. Some of the most imminent problems, such as the missing description of a component's footprint and the lacking distinction between various incoming TUIO streams, have been partially repaired through the introduction of a third **Blob profile** and the additional **Source message** within the TUIO 1.1 specification.

While these intermediate extensions proved to be useful for standard multi-touch application scenarios, there still remain several limitations of the current TUIO 1.1 abstraction model, which unfortunately can't be resolved by simple additions to its current protocol syntax:

- the definition of separate object, cursor and blob profiles limits the cross referencing between components
- the lack of a dedicated bundle time limits proper gesture recognition on the client side
- missing attributes cannot be simply added to component messages without breaking the syntax
- a simple object ID limits the introduction of additional symbol types

The ongoing development of the reacTIVision tracker and the analysis of further multi-touch applications and tangible interaction platforms, also identified the need for additional geometry detail and the definition of new interface components and their physical relationship. This eventually suggested the introduction of a completely new protocol generation, which I will discuss in detail in the following chapter along with an improved abstraction model.

Part III

MODEL EXTENSION

8 TUIO2

With the evaluation of the previous work related to the TUIO protocol and its application framework, I have identified the need for a fundamental redesign of its underlying component model in order to allow its extension to a broader scope of tangible application scenarios. Since the initial TUIO specification was mainly driven by the specific requirements of the Reactable platform, it had been designed for the representation of tagged physical objects as well as multi-touch gesture control. Although it has been shown that the simple bottom-up model approach of the initial TUIO specification allowed the construction of rather versatile application scenarios, I identified several shortcomings of the existing model, as well as the additional need of its radical extension to allow for a larger base of possible input technologies as well as a broader range of application scenarios. Therefore I developed an improved abstraction model, which on the one hand extends the various attribute descriptors of the existing object model, and on the other hand defines several additional tangible interface components, also introducing the concept of component relationships. This new model is based on an extensive analysis of the related research on tangible object models and application design patterns. While the original TUIO object model was influenced by a specific application domain, the proposed abstraction model is based on more common naming conventions in order to reflect its relation to previous research and to facilitate its integration with more general application scenarios and technologies. The primary focus of this work is the further technical implementation within our own and third party tangible interaction frameworks. After a presentation of the proposed abstraction model, I will also outline its actual implementation within a next generation TUIO protocol and framework, along with several examples showcasing the new model capabilities.

8.1 AN EXTENDED ABSTRACTION MODEL

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 design of this interaction framework targets a simplicity, which should ideally not exceed the complexity of single mouse input integration into a standard GUI application. 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 identified control instances 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 tanglible components and their attributes that can be also grasped by non-expert users, is one of the most important design principles of this framework.

8.1.1 Global Context

8.1.1.1 *Interactive Surface*

Our abstraction model has been designed for the specific context of a two-dimensional interactive surface. This includes the surface extension 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.

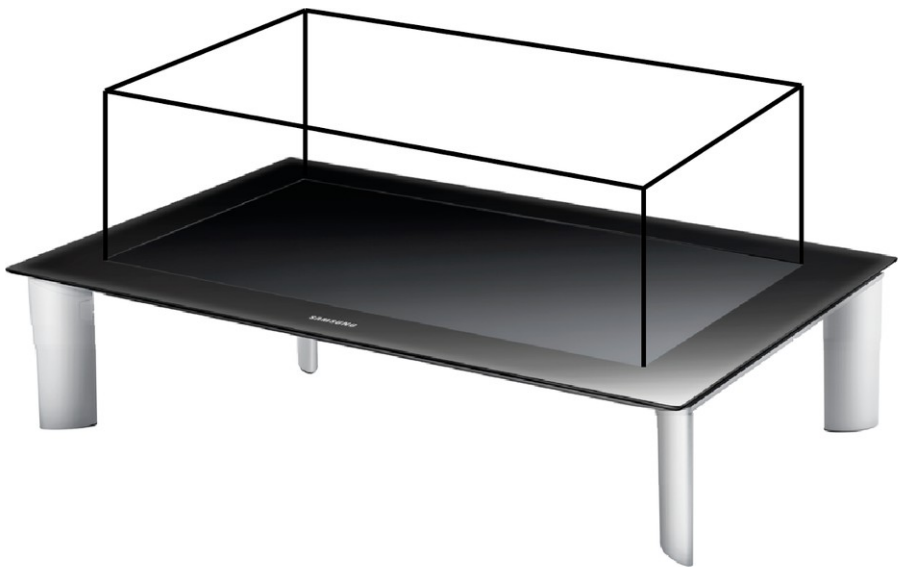

Figure 24: fishtank scenario: space extending above an interactive surface.

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.

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 primarily only relevant to the individual application scenario though.

8.1.1.2 *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 should be also understandable by non-expert users, and also allow for direct data interpretation by environments with limited computational infrastructure.

8.1.1.3 *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[65] or directly to the application layer itself.

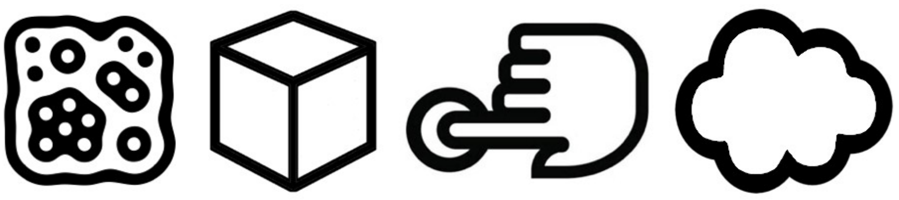

Figure 25: TUIO 2.0 components: symbols, tokens, pointers & geometries

8.1.2 Explicit Interface Components

8.1.2.1 *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 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.

8.1.2.2 *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 color tracking methods can also generate various classes of distinguishable physical tokens, while alternative 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.

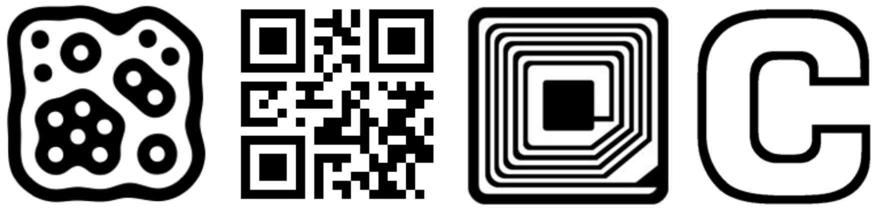

Figure 26: symbol examples: a) fiducial b) QR code c) RFID tag d) character

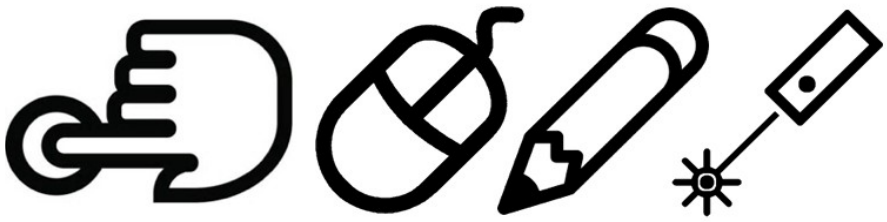

Figure 27: pointer examples: a) touch b) mouse c) pencil d) laser

8.1.2.3 *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 pencil. 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.

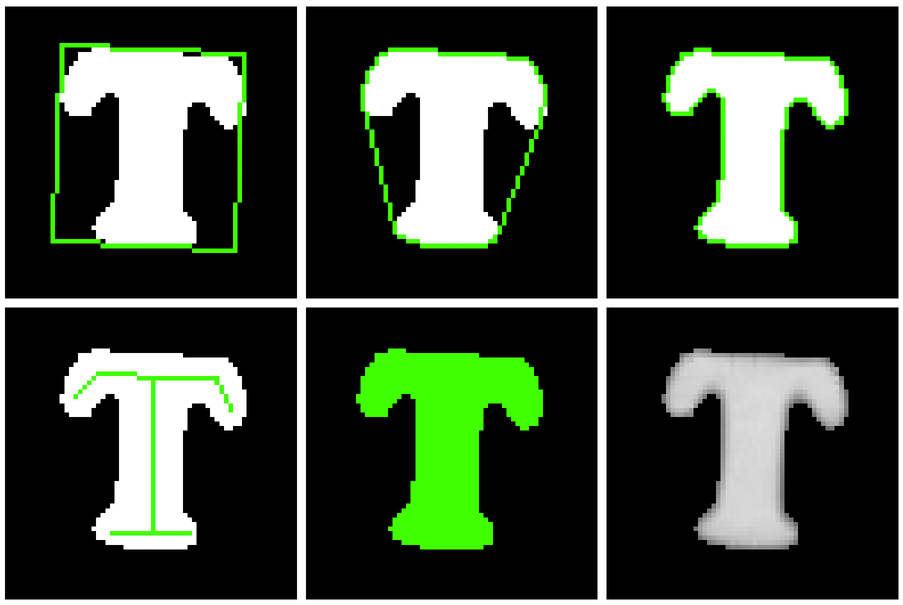

Figure 28: geometry detail: a) bounding box b) convex hull c) contour d) skeleton e) area spans f) raw data

8.1.2.4 *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.

8.1.2.5 *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.

8.1.3 Component Relation

8.1.3.1 *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**.[11] 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. Additional container associations allow the representation of **physically nested components**.

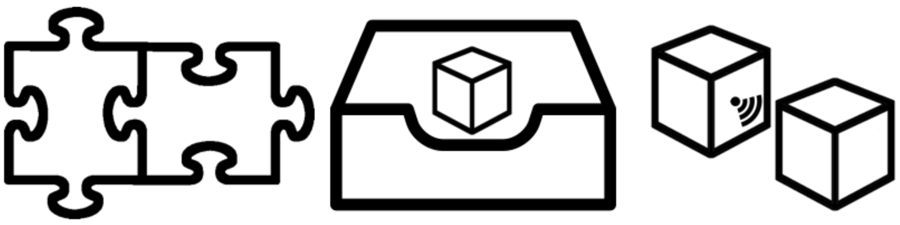

Figure 29: component relations: a) physical link b) container c) signal

8.1.3.2 *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.

8.1.4 Component Context

8.1.4.1 *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.

8.1.4.2 *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.

8.1.5 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.

8.1.5.1 *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.

8.1.5.2 *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.

8.1.5.3 *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.

8.1.5.4 *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.

8.1.5.5 *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.

8.2 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 are 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 upon the provided object model representations, by defining its dedicated application model only locally. This eventually allows the installation of these applications on different hardware platforms while they 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 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.

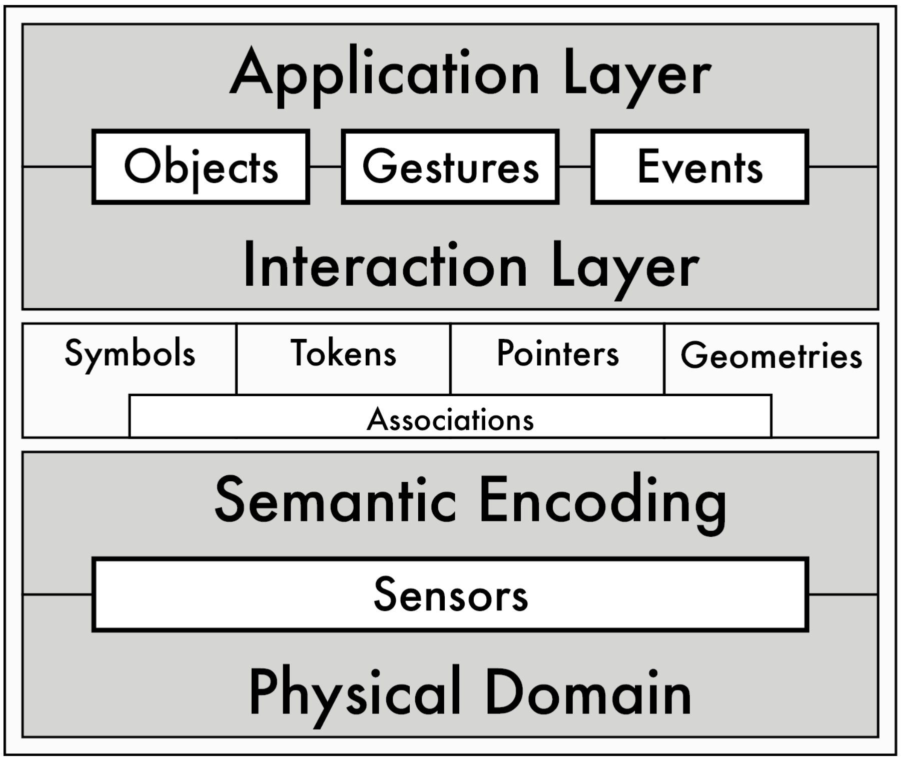

Figure 30: Layers of the Abstraction Model

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.

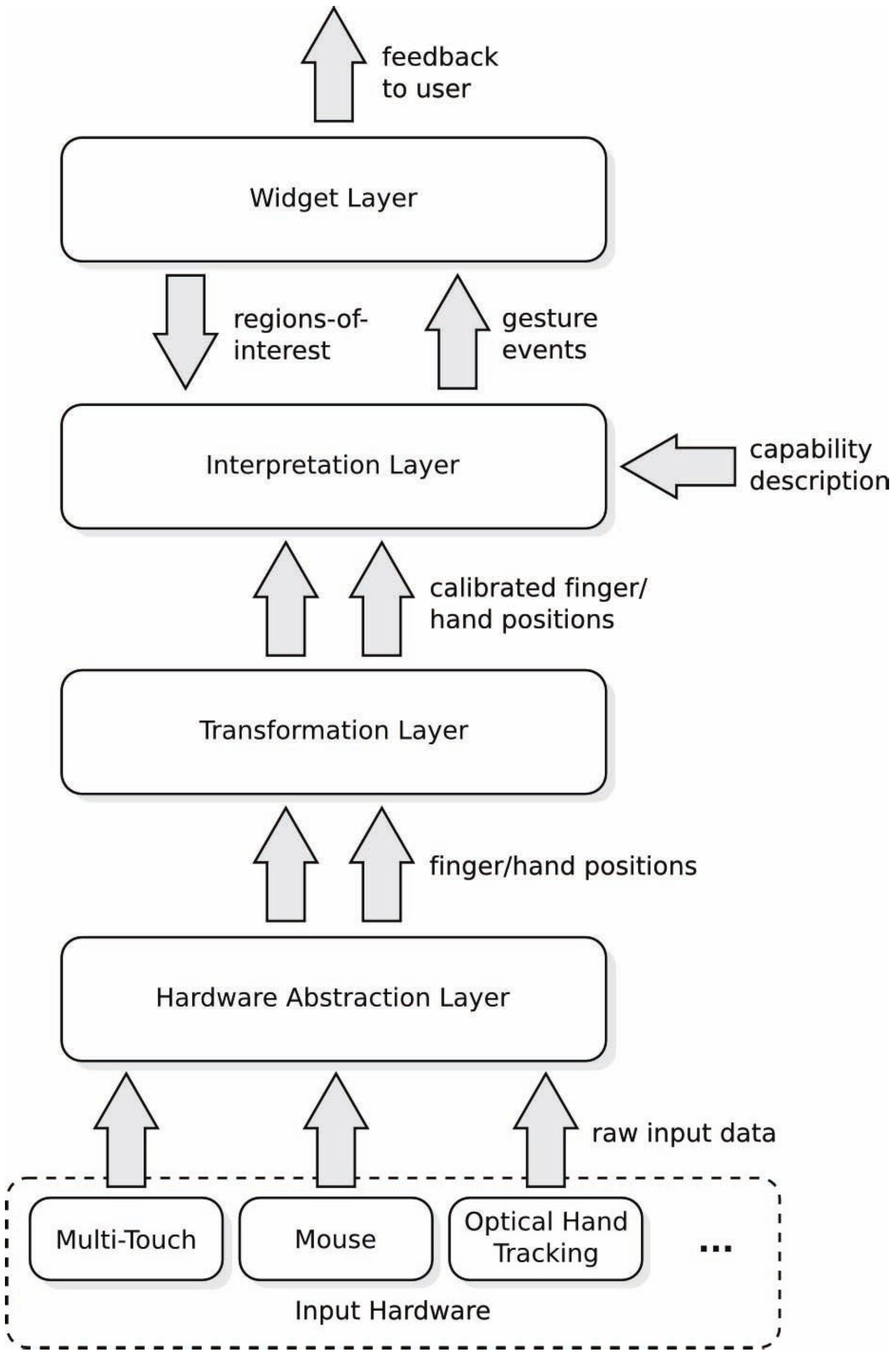

Figure 31: A Multitouch Software Architecture. Illustration: Echtler et.al.

Taking the example of Echter's et.al. multitouch software architecture[66], the TUIO abstraction can be simply represented within the hardware abstraction 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 components, it could be subsequently extended with tangible interface components that are provided by the abstraction model.

8.3 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. As in its predecessor, the TUIO 2.0 components and attributes are encoded using the Open Sound Control (OSC) format.

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.

8.3.1 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.

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.

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.

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.

8.3.2 Global Messages

8.3.2.1 *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, 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 for the out of order execution of redundant 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.

8.3.2.2 *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.

8.3.3 Component Messages

8.3.3.1 *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. 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.

8.3.3.2 *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 the 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 defines a Component ID that allows the distinction of individual pointer components during a whole session. 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 the more detailed extension of the pointer.

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 input devices 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 encodes the pressure velocity and acceleration instead of the rotation values here.

8.3.3.3 *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 center 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.

8.3.3.4 *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)

8.3.3.5 *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.

8.3.3.6 *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).

8.3.3.7 *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]
```

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.

8.3.4 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.

8.3.4.1 *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.

8.3.4.2 *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.

8.3.4.3 *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.

8.3.4.4 *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.

8.3.4.5 *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.

8.3.4.6 *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.

8.3.4.7 *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.

8.3.4.8 *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 previous width attribute specifies the relative distance between two points in order to correctly reconstruct the amount of samples between the initial and final span points. Although this value can be also retrieved from the surface dimension attribute 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.

8.3.5 Content Messages

8.3.5.1 *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 incorporated pressure sensor or for example. This (open length) list of variable float or boolean values, encodes each individual control dimension as discrete 0/1 bool 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.

8.3.5.2 *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. The 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

8.3.5.3 *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.

8.3.6 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.

8.3.6.1 *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.

8.3.6.2 *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.

8.3.6.3 *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 tupel 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).

8.3.6.4 *LLA (linked list association)*

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

8.3.6.5 *LTA (linked tree association)*

```
/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 int
    32
```

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 tupels are structured as defined in the LIA (link association) message specified above. The LTA node jump uses the float data type in order to allow its correct identification.

8.3.7 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.

8.3.8 Timing Model

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 surface within one second, results in a movement velocity of (1.0 0.0) The rotation velocity is defined as one full rotation per second. Therefore as an example, performing one object rotation within one second, results in a rotation velocity of 1.0. The acceleration values then simple are calculated as a function of speed changes over time (in seconds).

8.3.9 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 current TUIO message structure used a single name space for all messages within a bundle, which eventually caused problems with some OSC implementations. TUIO2 already takes this into account with its newly defined message structure, 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 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
```

Parameter	Short		OSC Type	Range
Session ID	s_id	S	32bit int, (uint32)	0 . . . 4.294.967.295
Component ID	c_id	I	32bit int, (uint32)	0 . . . 4.294.967.295
Type/User ID	tu_id	T	32bit int, (2x uint16)	0 ... 65535
Frame ID	f_id	F	32bit int, (uint32)	0 ... 4.294.967.295
Frame Time	time	t	64bit time	sec/nsec
Point Coordinate	x_pos, y_pos, z_pos	x, y, z	32bit float	0.0f ... 1.0f
Angle	angle	a	32bit float	0.0f ... $2 \cdot \pi$
Pressure/Hover	press	p	32bit float	-1.0f ...1.0f
Motion Velocity	x_vel, y_vel, z_vel	X, Y, Z	32bit float	-n ... n
Acceleration	m_acc	m	32bit float	-n ... n
Rotation Velocity	r_vel	A	32bit float	-n ... n
Rotation Accel	r_acc	r	32bit float	-n ... n
Width	width	w	32bit float	0.0f ... 1.0f
Height	height	h	32bit float	0.0f ... 1.0
Depth	depth	d	32bit float	0.0f ... 1.0f
Area/Volume	area/vol	f/v	32bit float	0.0f ... 1.0f
Link ID	l_id	L	32bit int, (2x uint16)	0 ... 65535
Source Name	source	N	OSC string	$4 \times n$ bytes
Symbol Data	data	D	OSC string, OSC Blob	$4 \times n$ bytes
Data MIME Type	mime	M	OSC string	$4 \times n$ bytes
Symbol Group	group	G	OSC string	$4 \times n$ bytes
Node Jump	node	n	32bit int, (uint32)	0 ... 4.294.967.295
Custom Parameter	string, float, int ...	P	OSC type	variable

8.3.10 Compact Message List

Global Messages

```
/tuio2/frm f_id time dim source
/tuio2/alv s_id0 ... s_idN
```

Component Messages

```
/tuio2/tok s_id tu_id c_id x_pos y_pos angle [x_vel y_vel m_acc r
    _vel r_acc]
/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/bnd s_id x_pos y_pos angle width height area [x_vel y_vel
    a_vel m_acc r_acc]
/tuio2/sym s_id tu_id c_id t_des data
/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/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 m_acc]
/tuio2/b3d s_id x_pos y_pos z_pos angle x-ax y-ax z-ax width
    height depth vol [x_vel y_vel z_vel r_vel m_acc r_acc]
```

Geometry Messages

```
/tuio2/chg s_id x_p0 y_p0 ... x_pN y_pN
/tuio2/ocg s_id x_p0 y_p0 ... x_pN y_pN
/tuio2/icg s_id x_p0 y_p0 ... x_pN y_pN true x_p0 x_p0 ... x_pN y
    _pN
/tuio2/skg s_id x_p0 y_p0 x_p1 y_p1 node ... x_pN y_pN
/tuio2/s3d s_id x_p0 y_p0 z_p0 x_p1 y_p1 z_p1 node ... x_pN y_pN
    z_pN
/tuio2/svg s_id r0 ... rN
/tuio2/arg s_id x0 y0 w0 ... xN yN wN
/tuio2/raw s_id width data
```

Content Messages

```
/tuio2/ctl s_id c0 ... cN
/tuio2/dat s_id mime data
/tuio2/sig s_id c_id s_id0 ... s_idN
```

Association Messages

```
/tuio2/ala s_id0 ... s_idN
/tuio2/coa s_id slot s_id0 ... s_idN
/tuio2/lia s_id bool s_id0 l_id0 ... s_idN l_idN
/tuio2/lla s_id bool s_id0 l_id0 ... s_idN l_idN
/tuio2/lta s_id bool s_id0 l_id0 s_id1l_id1 node ... s_idN l_idN
```

Custom Messages

```
/tuio2/_[attr] s_id [list of attributes]
/tuiox/[ext] s_id [list of attributes]
```

8.3.11 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 server and client reference implementations will be provided for the most common programming languages, such as C++, Java and C#. 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.

8.3.12 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.

8.4 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 several well known interactive surface environments, as well as some tangible musical applications. 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.

8.4.1 Pointer Identification: Diamondtouch

The MERL Diamondtouch platform[67] 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[68] 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

Figure 32: multi-user identification on the Diamond Touch. Photo: MERL

8.4.2 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 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, 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
```

Figure 33: iPad Pro pencil and touch interaction. Photo: Apple Incorporated

8.4.3 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
```

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

8.4.4 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
```

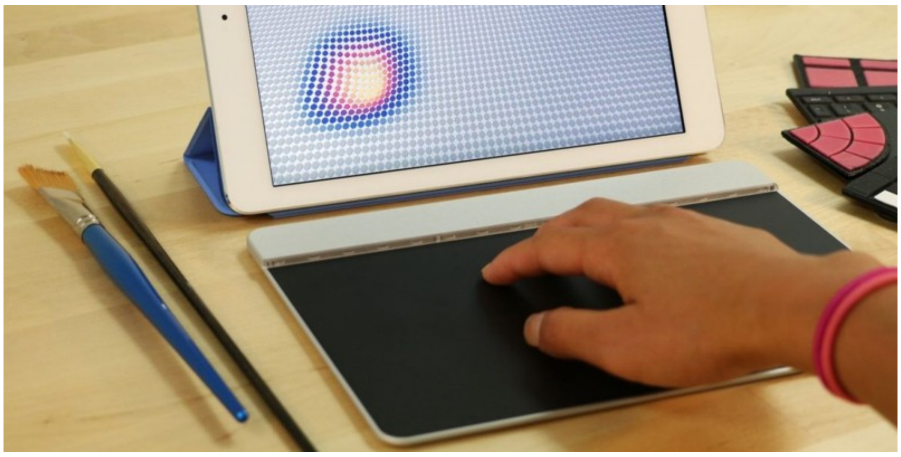

Figure 34: The Morph tablet and its pressure map display
Photo: Sensel Incorporated

8.4.5 Tokens, Pointers, Symbols & Bounds: Surface

The original Microsoft Surface platform supported multi-touch input as well as tagged and untagged object recognition. For object tagging there are two different fiducial types available, where the identity tag is capable of encoding a 128bit symbol. The generic Contact component can distinguish between finger touches, tags and untagged objects. Applying the TUIO abstraction model to these platform features therefore suggests the definition of Pointers, Tokens, Symbols

2 http://www.sensel.com/

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/tok s_id tu_id c_id x_pos y_pos angle
/tuio2/sym s_id tu_id c_id ms/id 0x00...00
/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
```

8.4.6 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 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.

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

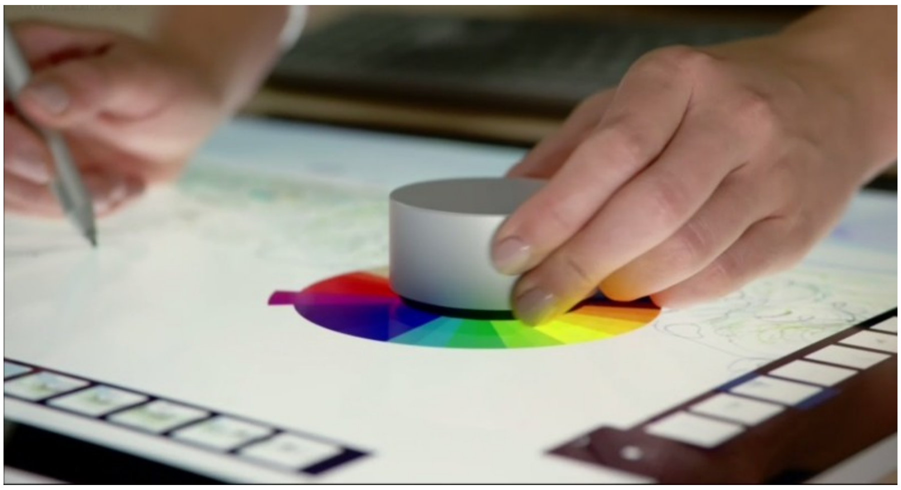

Figure 35: The Surface Studio with pen and dial.
Photo: Microsoft Corporation

8.4.7 Tokens, Pointers and Geometries: reacTIVision

The current development version of reacTIVision 1.6 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
```

8.4.8 Symbols and Bounds: Papier Maché

Klemmer's Papier Maché[22] 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
```

8.4.9 Tokens and Controls: Slap Widgets

Weiss' et.al. SLAP widgets[69] 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.

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

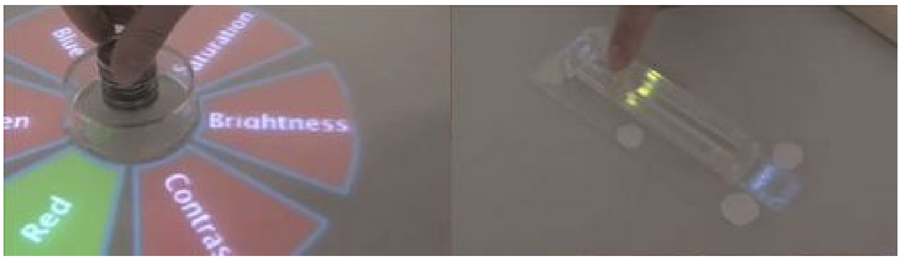

Figure 36: Two SLAP widgets with integrated controls.
Photos: Malte Weiss et.al.

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.

8.4.10 Spatial Interaction: Tisch

Echtler's Tisch[70] 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 employing a Microsoft Kinect for spatial tracking, 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 3D Pointer components can be employed for the representation of each hand and finger pointer and combine those through an optional Link Association. The complementary hand shadow from the surface can be also associated to the 3D hand pointer by using the same Session ID.

```
/tuio2/p3d s_id tu_id c_id x_pos y_pos z_pos x_ax y_ax z_ax rad
/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
```

8.4.11 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, not 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
```

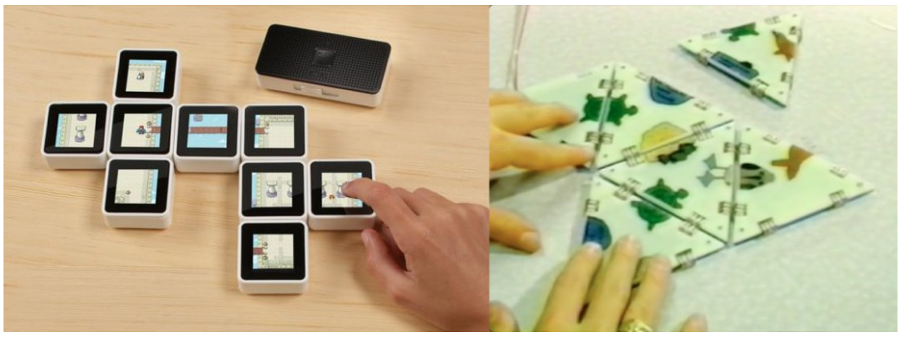

Figure 37: a) Sifteo Cubes and b) Triangles
Photos: David Merrill and Matthew Gorbet

8.4.12 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 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
```

8.4.13 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 0 tu_id c_id x_pos y_pos angle
/tuio2/tok 1 tu_id c_id x_pos y_pos angle
/tuio2/sig s_id0 0 s_id1
```

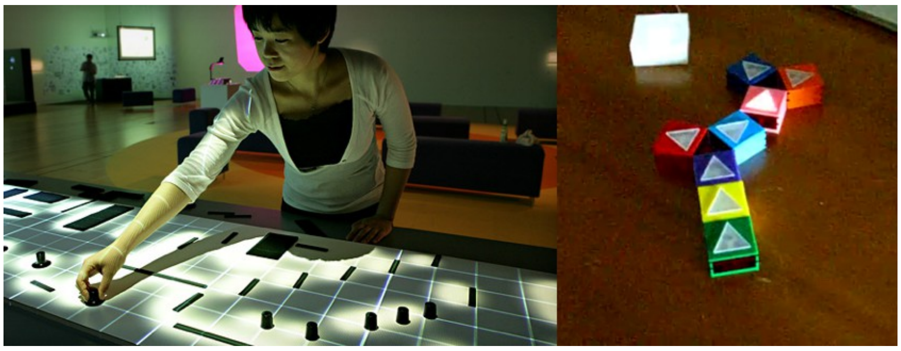
Figure 38: a) Scrapple installation and b) Tangible Sequencer
Photos: Golan Levin and Jeffrey Traer

8.4.14 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 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 ypN
```

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

8.4.15 Actuated Tokens: Pico

Patten's Pico platform[71] 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
```

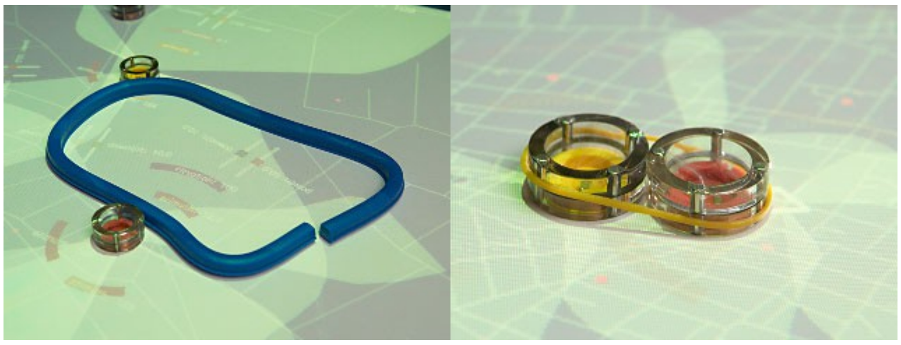
Figure 39: Physical constraints on the Pico platform. Photos: James Patten

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
```

8.4.16 Bonus Example: Marble Answering Machine

Although not based on an interactive surface, Bishop's Marble Answering Machine[72] is a classical example for several key properties of a tangible user inter interface. Therefore this interface is also suitable for an example encoding within the proposed abstraction model, in order to show its versatile application area, which can eventually cover a wider range of tangible user interfaces not only restricted to surface environments.

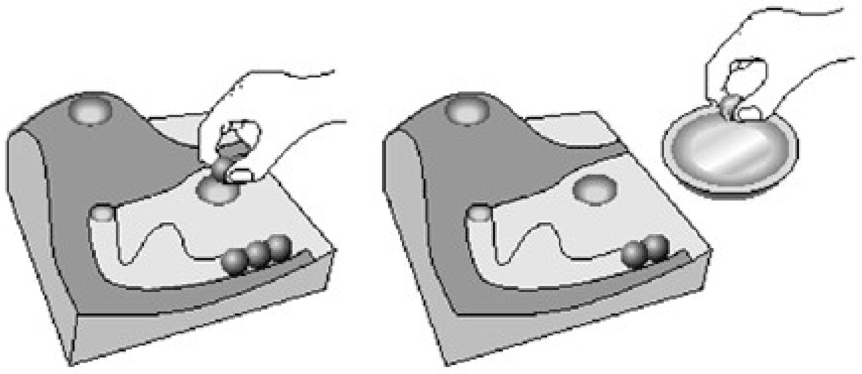

Figure 40: The Marble Answering Machine. Illustration: Durrell Bishop

The device is comprised of two key components, the answering machine body itself as well as a collection of marbles, which are representing the recorded messages. Although being physical tokens, the representation of these marbles can be simply realized with an according Symbol component, since the spatial position and orientation of the marbles are not relevant in this scenario. On the other hand this Symbol is also capable of storing the associated caller's phone number. The body of the answering machine has to be divided into two parts, which are serving as physical constraints for the marble tokens. This primarily includes the tray for the incoming messages as well as the device body containing various slots for the message manipulation. These device components can also be referenced through permanently present Token components with a dedicated Type ID, which identifies their distinct function from the marbles. Additional Data components can be employed to encode and transmit a recording of the incoming messages, and will be associated to an according marble Symbol after a message has been recorded. Finally a Container association can be employed in order to encode the token-constraint relationship between the marble and the primary machine components according to their current state.

In its initial state the marble answering contains a number of marbles, which in this default case do not need to be marked neither active nor explicitly associated to the answering machine itself. The two defined answering machine components, the machine body and the tray only need to be referenced once at the beginning of a session and then kept permanently active by a constant reference within the global ALV message.

Once a call is received a new Symbol component is generated containing the caller's phone number and encoded within an according SYM message. The recorded audio of the call is stored and transmitted by an associated DAT message. Since a TUIO bundle contains a detailed time stamp, the time of the call as well as its duration can be reconstructed from the according FRM and DAT message attributes. Finally as soon as the marble is dropped onto the tray, a Container association is established by sending a combination of ALA and COA messages. The ALA messages have to be sent with every bundle while the referenced marble Symbols are present on the tray. The position of the marbles on the tray can be also derived from the order of Session IDs within the COA message.

When a user selects a marble its association state remains unchanged until the marble is dropped into one of the slots for the playback, callback or deletion of the message. In this moment the container association with the tray token is released by omitting the according Session ID from the ALA list and COA Container association referencing the tray's Session ID. A new container relationship is then established by generating a new COA message referencing the machine body's Session ID as well as the according slot number. Once the according action has been performed the marble Symbol is either again associated to the tray or deleted completely from any reference, depending on the chosen slot number. Dropping a marble into the callback or playback slot will perform the according action and then drop the marble back onto the tray, while the delete slot will consequently deactivate the marble and keep it in the default waiting state within the body of the machine for later use.

```
/tuio2/tok s_id u_id:t_id c_id x_pos y_pos angle
/tuio2/sym s_id u_id:t_id c_id phone +43-1-123-456-789
/tuio2/dat s_id x-audio/wav 0x000000000000000000000000
/tuio2/ala s_id0 ... s_idN
/tuio2/coa s_id slot s_id0 ... s_idN
```

8.4.17 Conclusion

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 unlikely to provide a platform or toolkit 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.

9 CONCLUSION & FUTURE WORK

We have seen that the TUIO 1.1 specification has been implemented within a comprehensive ecosystem of trackers, clients, frameworks and tools, which support the versatile development of tangible interactive surface applications on many hard- and software platforms. Since the TUIO 2.0 specification[1] has been officially finalized during a community workshop[73] at the *International Conference for Interactive Tabletops and Surfaces* in autumn 2014, I have started the work on a first C++ reference implementation[2], which is already providing the basic TUIO 2.0 server and client infrastructure. Although TUIO 2.0 has already seen integrations within other application frameworks[65] and libraries[3], its current ecosystem still has not reached the critical mass that would be necessary for a full transition to the new protocol generation. While TUIO 2.0 provides a more comprehensive feature set for the representation of state-of-the-art tangible interactive surfaces, we are currently confronted with a chicken-egg problem, which is hindering a wider acceptance of the new protocol architecture: without the necessary variety of available TUIO 2.0 libraries and application frameworks, existing TUIO trackers cannot benefit from an upgrade and vice-versa. Therefore in analogy to the initial TUIO 1.0 development history, the plan is to provide comparable core implementations of trackers, clients and also conversion tools, which leverage the feature advantages of TUIO 2.0 and also allow for a more seamless transition from TUIO 1.1.

9.1 TUIO 2.0 REFERENCE IMPLEMENTATION

The current C++ reference implementation encapsulates the **Symbol**, **Token**, **Pointer** and **Bound** components, thus it presently only implements the core subset of the TUIO 2.0 specification, the remaining object **Geometries** and **Associations** will be added subsequently. The current codebase also provides Server and Client implementations for TUIO 2.0 encoding/decoding and we are also planning to add TUIO 1.1 backward compatibility. In addition to the standard TUIO/UDP transport layer we already support the existing TUIO/TCP, TUIO/WEB and TUIO/FLC transports and also plan to provide and additional TUIO/DAT layer for the recording and replay of binary TUIO/OSC streams. The current library also supports the multiplexing of multiple TUIO sources. Apart from the actual cross-platform TUIO 2.0 implementation, the code repository also provides several example projects that serve for the illustration and testing of the overall API functionality.

1 http://www.tuio.org/?tuio20
2 http://github.com/mkalten/TUIO20_CPP
3 http://github.com/nomve/Tuio.js

This library also defines a general API for the encapsulation of TUIO components. This API is already quite usable, but still contains several partially implemented sections, which are still subject to completion, change and improvements. Compared to the TUIO 1.1 API, it provides a unified callback and event interface, which provides all TUIO component updates within a generic *TuioComponent* object accompanied by an according *TuioEvent* structure, to any class implementing a simple *TuioListener* interface. Based on the final C++ API we will subsequently provide the according TUIO 2.0 modules for Pure Data, Max/MSP, Open Frameworks and other cross-platform C++ based application environments.

Once the C++ library and the related API reaches the necessary stability, we will then also port its overall structure and interface to other programming languages such as Java, JavaScript and C#, which will also allow its further integration into environments such as Processing, Unity or VVVV.

9.2 REACTIVISION REFERENCE PLATFORM

During the current development cycle of reacTIVision 1.6, there have already emerged several internally available features, which cannot be properly represented with the TUIO 1.1 model. This not only includes the introduction of the new Yamaarashi fiducials, which should be identified separately from the standard Amoeba symbols, but most importantly concerns the more detailed description of the blob geometries, such as its contour or convex hull, which are largely constrained by the currently rather simplistic TUIO 1.1 blob profile. Therefore after the actual release of reacTIVision 1.6, which will also mark the final TUIO 1.1 generation, an intermediate reacTIVision 1.9 version will be provided through the swift integration of the new TUIO 2.0 library generation. With this intermediate release, the plan is not yet to add any additional features, although the transition will already allow to represent the above-mentioned extended TUIO 2.0 such as multiply symbols and detailed blob geometries. In other to facilitate the migration of existing applications, this version will also provide alternative TUIO 1.1 backward compatibility. Therefore reacTIVision 1.9 will already be able to take advantage of the core TUIO 2.0 components, providing its core feature set of **Symbols**, **Tokens**, **Pointers**, **Bounds** and **Geometries**. The availability of these component features within a functional tracker application, will also allow the further refinement and evaluation of the equivalent TUIO 2.0 client functionality.

After this intermediate release, which should follow soon after reacTIVision 1.6, the plan is to extend its internal functionality towards a future reacTIVision 2.0 version, which will allow us to take advantage of additional TUIO 2.0 features. Since the overall finger blob size and shape is already available, its additional footprint and orientation can be easily provided, as well as an approximate pressure value derived from size changes, hence taking advantage of most **Pointer** attributes. Furthermore the existing RFID reader tool *nfosc*[4] should be integrated directly into reacTIVision, which will allow the representation of a complementary electronic **Symbol** type in addition to the computer-vision markers. This will also allow the subsequent integration of various additional symbol types such as QR codes, as it was already mentioned in one of the earlier publications. Since reacTIVision should be also understood as TUIO 2.0 reference platform, as many features as possible of this new protocol generation should be implemented during the further course of its development.

Continuing the further multi-platform integration, there are also plans to extend the PortVideo camera framework to mobile operating systems such as iOS and Android, and will continue to implement additional camera drivers, such as the emerging USB3 Video standard.

9.3 DEVICE SUPPORT

In addition to the hard- and software environments that are dedicated to the realization of tangible interactive surfaces, there is a large number of devices, which can also benefit through an implementation of the new TUIO 2.0 feature set. This includes currently available optical touch frames and capacitive touch overlays for the integration of custom multi-touch displays, as well as several multi-touch enabled devices such as mobile tablets, touch pads, force sensitive touch controllers or depth cameras. There are already several multi-touch tools implementing TUIO 1.1 for these devices such as TuioPad[5] for iOS and TuioDroid[6] for Android, or TongSeng[7] for MacOS. These tools not only allow TUIO remote control configurations, but are also useful for the development and testing of TUIO client applications.

As shown earlier, the current version of Apple's iPad Pro tablets already exceed the features of TUIO 1.1 by providing separate touch input, as well as a force sensitive pencil including its rotation and shear angle. Therefore a new version of the TuioPad application will be provided, which will encode these additional features taking advantage of the full range of Pointer attributes, such as pointer type, size, pressure and angle. More recent pressure-sensitive controller devices such as the Sensel Morph equally can not only benefit from the additional pressure attribute of a Pointer component, but can also encode object contours through their Geometry or by even providing a full regional pressure map through detailed Blob data. Other controllers such as the Leap Motion can also fully encode and identify individual fingers and both hands in their spatial position through Pointer3D components. While other optical and capacitive touch platforms, which often provide native TUIO 1.0 touch support would not directly require any of the new TUIO 2.0 attributes, they will eventually also benefit from a transition to the overall ecosystem of the new protocol generation.

4 http://github.com/mkalten/nfosc
5 http://github.com/mkalten/tuiopad
6 http://github.com/TobiasSchwirten/tuiodroid
7 http://github.com/fajran/tongseng

9.4 UNIVERSAL TUIO TOOL

In order to further support the transition to TUIO 2.0 I am currently designing an *Universal TUIO Tool*, which will not provide a simple but versatile conversion utility between the two protocol generations, but will also serve as a general container for the TUIO integration into standard operating systems and their native multi-touch layer. Within today's TUIO 1.1 ecosystem there already exists a large variety of so called TUIO bridges or gateways, which inject TUIO cursor input into standard system multi-touch APIs, or generate on the other hand TUIO cursor output from the generic single- or multi-touch input on these platforms. This universal tool intends to integrate all these efforts into a single extensible multi-platform application, which shall bring comprehensive TUIO support to the most relevant desktop and mobile systems, and will also allow the easier integration of TUIO enabled multi-touch devices or other interactive hardware. Third party manufacturers who wish to provide TUIO output for their own devices, should be able to extend this tool through an additional plugin interface. In addition to providing alternative TUIO/UDP transport conversions, such as TUIO/WEB for websocket applications, the included TUIO/DAT recording and replay functionality will also benefit TUIO developers for testing purposes.

The following table summarizes the planned layers of a) TUIO 1.1 and 2.0 input and output through UDP, TCP, WebSockets and files; b) TUIO conversion from and to OS specific touch APIs; and c) hardware specific device input to TUIO.

	IN	**OUT**
Cross-Platform	TUIO 1.1 & 2.0: UDP, TCP, DAT	TUIO 1.1 & 2.0: UDP, TCP, WEB, FLC, DAT
Windows	Windows Touch	Windows Touch
MacOS	touchpad	mouse, touch injection
Linux	Xinput	Xinput
Android	touch (TuioDroid)	touch injection (evdev)
iOS	touch, pencil (TuioPad)	-
Devices	HID, SUR40, Wacom, Leap Motion, WiiMote Sensel Morph etc.	-
Plugins	by manufacturers of touch frames, foils etc.	-

Table 2: Universal TUIO Modules

This UniTUIO tool eventually intends to replace various, now partially outdated OS integration and conversion tools, and should also facilitate the further usage of TUIO 1.1 trackers, libraries, application frameworks and hardware within the TUIO 2.0 ecosystem.

9.5 CONCLUSION

With this cumulative dissertation I presented four publications, which document the design, implementation, application and evaluation of a simple, yet versatile abstraction model for tangible interactive surfaces. Based on the comprehensive experiences collected throughout the development of this model and its implementation within a community-driven soft- and hardware ecosystem, I designed an extended new generation of this abstraction model, which intends to reflect the state-of-the-art research and technology for tangible interactive surfaces. I hope that this updated model will foster the further community-driven research efforts not only for gesture-based multi-touch interaction, but more importantly for the ongoing development of the tangible interaction paradigm within the physical domain.

The **initial abstraction model** has 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. In collaboration with the community we also provided a comprehensive collection of software tools, which not only implement the above protocol but also showcase its core capabilities within a dedicated computer-vision toolkit for the design and realization of tangible surface applications. This **open-source approach** also led to the development of additional tools and applications based on the protocol. While most of these external implementations also have proven the overall validity of the generic abstraction model, several instances also allowed for an analysis of certain limitations imposed by this model simplification.

This analysis of the 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**

It is to hope that the new TUIO 2.0 protocol and the related applications, libraries and tools will provide an attractive feature extension in order to initiate the further community support of the new protocol generation, similar to the manifold contributions within the existing TUIO 1.1 ecosystem.

Part IV

APPENDIX

BIBLIOGRAPHY

[1] Eva Hornecker and Orit Shaer. “Tangible User Interfaces: Past, Present, and Future Directions.” In: *Foundations and Trends in Human-Computer Interaction* 3 (2010).

[2] Brygg Ullmer and Hiroshi Ishii. “Human-Computer Interaction in the New Millennium.” In: ed. by J. M. Carroll. Addison-Wesley, 2001. Chap. Emerging Frameworks for Tangible User Interfaces, pp. 579–601.

[3] George W. Fitzmaurice, Hiroshi Ishii, and William A. S. Buxton. “Bricks: Laying the Foundations for Graspable User Interfaces.” In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’95. Denver, Colorado, USA: ACM, 1995.

[4] Ben Shneiderman. “Direct Manipulation: A Step Beyond Programming Languages.” In: *Proceedings of the Joint Conference on Easier and More Productive Use of Computer Systems: Human Interface and the User Interface*. CHI ’81. Ann Arbor, MI: ACM, 1981.

[5] Brygg Ullmer and Hiroshi Ishii. “The metaDESK: Models and Prototypes for Tangible User Interfaces.” In: *Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology*. UIST ’97. Banff, Alberta, Canada: ACM, 1997.

[6] Hiroshi Ishii and Brygg Ullmer. “Tangible Bits: Towards Seamless Interfaces Between People, Bits and Atoms.” In: *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*. CHI ’97. Atlanta, Georgia, USA: ACM, 1997.

[7] Eva Hornecker. “Tangible User Interfaces als kooperationsunterstützendes Medium.” PhD thesis. Universität Bremen, 2004.

[8] John Underkoffler and Hiroshi Ishii. “Urp: A Luminous-tangible Workbench for Urban Planning and Design.” In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’99. Pittsburgh, Pennsylvania: ACM, 1999.

[9] Brygg Ullmer and Hiroshi Ishii. “mediaBlocks: Tangible Interfaces for Online Media.” In: *CHI ’99 Extended Abstracts on Human Factors in Computing Systems*. Pittsburgh, Pennsylvania: ACM, 1999.

[10] Matthew G. Gorbet, Maggie Orth, and Hiroshi Ishii. “Triangles: Tangible Interface for Manipulation and Exploration of Digital Information Topography.” In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Los Angeles, California: ACM, 1998.

[11] Brygg Ullmer and Hiroshi Ishii. “Emerging frameworks for tangible user interfaces.” In: *IBM systems journal* (2000).

[12] Hiroshi Ishii, Ali Mazalek, and Jay Lee. "Bottles As a Minimal Interface to Access Digital Information." In: *CHI '01 Extended Abstracts on Human Factors in Computing Systems*. Seattle, Washington: ACM, 2001.

[13] Kenneth P. Fishkin. "A Taxonomy for and Analysis of Tangible Interfaces." In: *Personal Ubiquitous Comput.* 8.5 (2004), pp. 347–358.

[14] Brygg Ullmer, Hiroshi Ishii, and Robert J. K. Jacob. "Token+Constraint Systems for Tangible Interaction with Digital Information." In: *Transactions on Computer-Human Interaction* 12.1 (2005), pp. 81–118.

[15] Orit Shaer, Nancy Leland, Eduardo H. Calvillo-Gamez, and Robert J. Jacob. "The TAC Paradigm: Specifying Tangible User Interfaces." In: *Personal Ubiquitous Computing* 8.5 (2004), pp. 359–369.

[16] Orit Shaer. "A Visual Language for Specifying and Programming Tangible User Interfaces." PhD thesis. Tufts University, 2008.

[17] Hirokazu Kato, Mark Billinghurst, Ivan Poupyrev, and Keihachiro Tachibana. "Virtual Object Manipulation on a Table-Top AR Environment." In: *Proceedings of the International Symposium on Augmented Reality (ISAR2000)*. Munich, Germany, 2000.

[18] Jungong Han, Ling Shao, Dong Xu, and J. Shotton. "Enhanced Computer Vision With Microsoft Kinect Sensor: A Review." In: *IEEE Transactions on Cybernetics* 43.5 (Oct. 2013), pp. 1318–1334.

[19] Saul Greenberg and Chester Fitchett. "Phidgets: Easy Development of Physical Interfaces Through Physical Widgets." In: *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology*. Orlando, Florida: ACM, 2001.

[20] Ayah Bdeir. "Electronics As Material: LittleBits." In: *Proceedings of the 3rd International Conference on Tangible and Embedded Interaction*. Cambridge, United Kingdom: ACM, 2009.

[21] David Merrill, Jeevan Kalanithi, and Pattie Maes. "Siftables: Towards Sensor Network User Interfaces." In: *Proceedings of the 1st International Conference on Tangible and Embedded Interaction*. Baton Rouge, Louisiana: ACM, 2007.

[22] Scott R. Klemmer, Jack Li, James Lin, and James A. Landay. "Papier-Mache: Toolkit Support for Tangible Input." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Vienna, Austria: ACM, 2004.

[23] Nicolas; Kamp Jean-François Gibet Sylvie; Courty, ed. *Gesture in Human-Computer Interaction and Simulation*. Vol. 3881. Lecture Notes in Computer Science. Springer, Berlin, Germany, 2005.

[24] Martin Kaltenbrunner, Günter Geiger, and Sergi Jordà. "Dynamic Patches for Live Musical Performance." In: *Proceedings of the 4th Conference on New Interfaces for Musical Expression (NIME04)*. Hamamatsu, Japan, 2004.

[25] Thomas Hermann, Thomas Henning, and Helge Ritter. "Gesture Desk - An Integrated Multi-Modal Workplace for Interactive Sonification." In: *Proceedings of the International Gesture Workshop*. Genova, Italy, 2003.

[26] Enrico Costanza, Simon B. Shelley, and John Robinson. "D-touch: A Consumer-Grade Tangible Interface Module and Musical Applications." In: *Proceedings of the Conference on HumanComputer Interaction (HCI03)*. Bath, UK, 2003.

[27] Matthew Wright, Adrian Freed, and Momeni Ali. "OpenSound Control: State of the Art 2003." In: *Proceedings of the 3rd Conference on New Instruments for Musical Expression (NIME03)*. Montreal, Canada, 2003.

[28] Ross Bencina. *oscpack*. 2004. URL: http://www.rossbencina.com/code/oscpack.

[29] Jefferson Y. Han. "Multi-touch Sensing Through Frustrated Total Internal Reflection." In: *ACM SIGGRAPH 2005 Sketches*. SIGGRAPH '05. Los Angeles, California: ACM, 2005.

[30] SK Lee, William Buxton, and K. C. Smith. "A Multi-touch Three Dimensional Touch-sensitive Tablet." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '85. San Francisco, California: ACM, 1985.

[31] Johannes Schöning et al. *Multi-Touch Surfaces: A Technical Guide*. Tech. rep. Technische Universität München, 2008.

[32] Shahram Izadi, Steve Hodges, Stuart Taylor, Dan Rosenfeld, Nicolas Villar, Alex Butler, and Jonathan Westhues. "Going Beyond the Display: A Surface Technology with an Electronically Switchable Diffuser." In: *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*. UIST '08. Monterey, CA: ACM, 2008.

[33] Daniel Gallardo and Sergi Jordà. "SLFiducials: 6DoF Markers for Tabletop Interaction." In: *Proceedings of the 2013 ACM International Conference on Interactive Tabletops and Surfaces*. ITS '13. St. Andrews, UK: ACM, 2013.

[34] Sergi Jordà, Günter Geiger, Marcos Alonso, and Martin Kaltenbrunner. "The reacTable: Exploring the Synergy between Live Music Performance and Tabletop Tangible Interfaces." In: *Proceedings of the first international conference on Tangible and Embedded Interaction (TEI07)*. Baton Rouge, Louisiana, 2007.

[35] Ross Bencina, Martin Kaltenbrunner, and Sergi Jordà. "Improved Topological Fiducial Tracking in the reacTIVision System." In: *Proceedings of the IEEE International Workshop on Projector-Camera Systems (Procams 2005)*. San Diego, USA, 2005.

[36] Ross Bencina and Martin Kaltenbrunner. "The Design and Evolution of Fiducials for the reacTIVision System." In: *Proceedings of the 3rd International Conference on Generative Systems in the Electronic Arts (3rd Iteration 2005)*. Melbourne, Australia, 2005.

[37] Sergi Jordà, Martin Kaltenbrunner, Günter Geiger, and Ross Bencina. "The reacTable*." In: *Proceedings of the International Computer Music Conference (ICMC 2005)*. Barcelona, Spain, 2005.

[38] Martin Kaltenbrunner, Till Bovermann, Ross Bencina, and Enrico Costanza. "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, 2005.

[39] MIDI Manufacturers Association. *Musical Instrument Digital Interface*. 1982. URL: http://www.midi.org/.

[40] Jefferson Y. Han. "Low-cost multi-touch sensing through frustrated total internal reflection." In: *Proceedings of the 18th annual ACM symposium on User interface software and technology (UIST05)*. Seattle, USA, 2005.

[41] Mónica Rikić. "Buildasound." In: *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction*. TEI '13. Barcelona, Spain: ACM, 2013.

[42] Martin Kaltenbrunner, Sile O'Modhrain, and Enrico Costanza. "Object Design Considerations for Tangible Musical Interfaces." In: *Proceedings of the COST287-ConGAS Symposium on Gesture Interfaces for Multimedia Systems*. Leeds, UK, 2004.

[43] Sergi Jordà. "Multi-user Instruments: Models, Examples and Promises." In: *In Proceedings of 2005 International Conference on New Interfaces for Musical Expression (NIME05)*. Vancouver, Canada, 2005.

[44] William A. Buxton. "User Centered System Design: New Perspectives on Human-Computer Interaction." In: ed. by Donald A. Norman and Draper Stephen W. Hillsdale, NJ: CRC Press, 1986. Chap. There is more to interaction than meets the eye: some issues in manual input, pp. 319–337.

[45] William A. Buxton. "Readings in Human-Computer Interaction: A Multidisciplinary Approach." In: ed. by Ronald M. Baecker and William A. Buxton. San Mateo, CA: Morgan Kaufmann Publishers, 1988. Chap. The haptic channel, pp. 357–365.

[46] Sergi Jordà. "Instruments and Players: Some thoughts on digital lutherie." In: *Journal of New Music Research* 33 (2005).

[47] Tina Blaine and Tim Perkis. "Jam-O-Drum, A Study in Interaction Design." In: *Proceedings of the ACM DIS 2000 Conference*. New York, NY, 2000.

[48] Sidney Fels, Linda Kaastra, Sachiyo Takahashi, and Graeme McCaig. "Evolving Tooka: from Experiment to Instrument." In: *Proceedings of the 4th Conference on New Interfaces for Musical Expression (NIME04)*. Vancouver, Canada, 2004.

[49] Álvaro Barbosa. "Public Sound Objects: A Shared Environment for Networked Music Practice on the Web." In: *Organized Sound* 10.3 (2005), pp. 233–242.

[50] Álvaro Barbosa. "Displaced Soundscapes: A Survey of Network Systems for Music and Sonic Art Creation." In: *Leonardo Music Journal* 13 (2003), pp. 53–59.

[51] Miller Puckette. "Pure Data." In: *Proceedings of the International Computer Music Conference*. San Francisco, CA, 1996.

[52] Marcos Alonso, Günter Geiger, and Sergi Jordà. "An Internet Browser Plug-in for Real-time Audio Synthesis." In: *Proceedings of International Computer Music Conference (ICMC04)*. Miami, Florida, 2004.

[53] Bill Buxton. "Artists and the Art of the Luthier." In: *SIGGRAPH Computer Graphics* 31.1 (Feb. 1997), pp. 10–11.

[54] Robert Andrews. *ReacTable Tactile Synth Catches Björk's Eye and Ear*. Aug. 2007. URL: `http://archive.wired.com/entertainment/music/news/2007/08/bjork_reacTable`.

[55] Gerhard Reitmayr and Dietmar Schmalstieg. "An open software architecture for virtual reality interaction." In: *Proceedings of the ACM symposium on Virtual reality software and technology (VRST01)*. Baniff, Canada, 2001.

[56] Martin Kaltenbrunner and Ross Bencina. "reacTIVision: A Computer-Vision Framework for Table-Based Tangible Interaction." In: *Proceedings of the first international conference on Tangible and Embedded Interaction (TEI07)*. Baton Rouge, Louisiana, 2007.

[57] R. E. Kalman. "A New Approach to Linear Filtering and Prediction Problems." In: *Transactions of the ASME– Journal of Basic Engineering* (1960), pp. 35–45.

[58] J. Bernsen. "Dynamic thresholding of grey-level images." In: *Proceedings of the 8th International Conference on Pattern Recognition*. 1986.

[59] Adrian Freed and A. Schneider. "Features and Future of Open Sound Control version 1.1 for NIME." In: *Proceedings of the 9th Conference on New Interfaces for Musical Expression (NIME09)*. Pittsburgh, Pennsylvania, 2009.

[60] W. A. König, Roman Rädle, and Harald Reiterer. "Squidy: a zoomable design environment for natural user interfaces." In: *Proceedings of the 27th international conference on Human factors in computing systems (CHI09)*. 2009.

[61] Peter Hutterer and Bruce Thomas. "Groupware support in the windowing system." In: *Proceedings of the 8th Australasian conference on User interface (AUIC07)*. 2007.

[62] Florian Echtler and Martin Kaltenbrunner. "SUR40 Linux: Reanimating an Obsolete Tangible Interaction Platform." In: *Proceedings of the ACM International Conference on Interactive Surfaces and Spaces (ISS2016)*. Niagara Falls, Canada, 2016.

[63] Clemens Nylandsted Klokmose, Janus Bager Kristensen, Rolf Bagge, and Kim Halskov. "BullsEye: High-Precision Fiducial Tracking for Table-based Tangible Interaction." In: *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces*. Dresden, Germany: ACM, 2014.

[64] Géry Casiez, Nicolas Roussel, and Daniel Vogel. "1 &#8364; Filter: A Simple Speed-based Low-pass Filter for Noisy Input in Interactive Systems." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '12. Austin, Texas: ACM, 2012.

[65] Florian Echtler and Andreas Butz. "GISpL: Gestures Made Easy." In: *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction*. Kingston, Ontario, Canada: ACM, 2012.

[66] Florian Echtler and Gudrun Klinker. "A Multitouch Software Architecture." In: *Proceedings of the 5th Nordic Conference on Human-computer Interaction: Building Bridges*. Lund, Sweden: ACM, 2008.

[67] Paul Dietz and Darren Leigh. "DiamondTouch: A Multi-user Touch Technology." In: *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology*. Orlando, Florida: ACM, 2001.

[68] Raf Ramakers, Davy Vanacken, Kris Luyten, Karin Coninx, and Johannes Schöning. "Carpus: A Non-intrusive User Identification Technique for Interactive Surfaces." In: *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*. Cambridge, MA: ACM, 2012.

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

[70] Florian Echtler, Manuel Huber, and Gudrun Klinker. "Shadow Tracking on Multi-touch Tables." In: *Proceedings of the Working Conference on Advanced Visual Interfaces*. Napoli, Italy: ACM, 2008.

[71] James Patten and Hiroshi Ishii. "Mechanical Constraints As Computational Constraints in Tabletop Tangible Interfaces." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. San Jose, California: ACM, 2007.

[72] R. Poynor. "The hand that rocks the cradle." In: *ID Magazine* May/June (1995), pp. 60–65.

[73] Martin Kaltenbrunner and Florian Echtler. "TUIO Hackathon." In: *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces (ITS2014)*. Dresden, Germany, 2014.

LIST OF FIGURES

LIST OF TABLES