Technical Spec[ific/ul]ations

This document covers the "technical" aspects of the txoOm system, including hardware, software and related systems. It includes design, implemetation details along with blue sky speculation (hopefully it should be clear which is which).

Changes in brief (for more details see cvs changelog)
1.0 based on tGarden "Tech_Spec" mathematica notebook (credits: sha xin wei, nik gaffney, joel ryan, yon visell, shi yifan, dave tonnesen, stock, ozan cakmakci, pix, tGarden consortium)
1.1 included parts of the design notes from brussels meeting.

1 System Desiderata

‡ 1.1 Function

The txOom system observes people's activities via sensors and cameras, modifies the global processes of the room, and generates the sound and imagery to fill the physical space, influenced by these observations.

‡ 1.2 Engineering Considerations

Stability

Each of the system components should operate without fatal error. Interoperating, the components should degrade gracefully, when receiving too little or too much data, or data that's inconsistent with its current context.

Responsivity

The system will be running in realtime, so latency must be kept to a minimum. Each component should be designed to be as reponsive as possible, and the overall system must provide no more than a 30ms lag between a gesture and (computed) response. While the system may have reponses which occur over longer timescales, the ideal minimum response time would be around 4ms. (while this may prove infeasable, it is a stated design goal.)

Testing

Each component will be individually tested before being integrated into the system. The system as a whole will be tested at the integration/test site as well as at the presentation venue. This Technical Specification document provides a metric for feature implementation, while the accompanying Design Specification provides a way of determining the experiential effectiveness of the system.

Flexibility

txOom should have a degree of configurability and be able to incorporate changes in components or sub-components relatively easily.

Extensibility

txOom should be designed in such a way as to allow it to be scaled, and remain a solid base for development beyond the current version. All code and hardware configurations should be well documented to enable extension, revision or modification by someone who has not necessarily had contact with the components from their inception.

Openness + Availability

The system should remain as open as practical. Software components developed for the txOom project should be licensed under the GPL. The system will rely on 3rd party software and hardware for some time, and while some of these packages are not free-software we should make an effort to use software that is freely available, or design the systems in a manner which allows propriety components to be replaced, if a viable alternative becomes available.

‡ 1.3 Implementation Environments

We choose the implementation environments on two sets of criteria: (1) formal merits like computational efficiency, expressive power, syntactic match to our application's procedures and representations, and (2) usage merits: familiarity to the programmer, legibility to development team, debugging facility, etc.

Strategically, we also choose hardware/OS platforms and development frameworks that minimize cost, in financial terms as well as on terms of compatibility, interoperability, future migration path, openness of support and importantly - availabilty of hardware at presentaion sites.

The mix of implementation environments maximizes the power of the media components, yet registers the media choreography and sensor interpretation in a relatively high-level representation that allows designers to rapidly write and rewrite different media choreographies and interpretive policies.

2 System Architecture

‡ 2.1 System components overview

These are the functional components:

Garments; clothing worn by the players, may incorporate "Wearable component"
Wearable component: sensors, microcontroller + transmitter
Vision tracking component; cameras, video capture, motion tracking
Oz: Sensor Map + Room Logic software;
Visual component: computer graphics, video post-processing
Sound component; making noises
Audiovisual Output component: Video projection, digital effects and sound system
Network component; Networking hardware, and communicaiton protocols

‡ 2.2 Hardware Overview

ü Sensors and Wireless (Wearable Component)

wearable units: consisting of sensors (integrated into costumes) a microcontroller assembly, 802.11b card, and battery pack. (1 per player)

ü Fixed Location

Computers; apple g4 computers, required graphics cards, soundcard. etc

Networking; 100baseT ethernet hub + cables connecting fixed computers.
wireless ethernet basestation.

Projectors 1 to 4, 2000 lumens or above

Soundsystem: 6x6ch mixing desk, 6 speakers

Digital Effects box?

Video Cameras: 2 for stereo pair, 1 overhead, 1 side for monitoring?

Proxima mutlichannel video recorder.

(check full tech rider after meeting at interactive institute)

‡ 2.3 Software Overview

There are 7 main software components:

- sensor component; sensor data aquisition + transmission.
- vision tracking component; tracks players in the room
- Oz (sensor mapping component); distributes + interprets sensor data to direct the media components.
- Oz (room logic); represents high level 'choreographic' description of the enviroments bahaviour.
- visual component; for processing and rendering of visual output.

- sound component; creates the audible aspect of the txOom;
- networking component; enables component systems to communicate with each other.

‡ 3.4 System Achitecture Overview

<<insert system detail diagram>>

‡ 3.5 Sytem Causal Chain

One of the desgin goals of the txOom is to keep the latency of the entire system as low as possible, in order to maintain a sense of responsivity in the room. This diagram shows the causal (temporal) relationship between componets, and may provide a guide for targeted optimisations.

<<insert system latency diagram>>

3 Sensors and Wearable Component

The sensors measure acceleration and rotation information in 3 dimensions from players. Each player will have 1 or 2 sets of them on their body or limbs.The sensor data is transmitted over wireless ethernet to a basestation on the network.

sensors:accelerometers,1 or 2 sets per player.(eg.Adaptive Technologies,ADXL202)

wireless: will involve sensorsÆmicrocontrollerÆwireless ethernet (802.11b)

<more details from todor,bert,nik>

‡ 3.1 Communication with Oz

ü From Wearable system to Oz

All wearables in the Room tranmit data to a wireless base-station, conencted to the computer running Oz. There is a program running on the wearable which reads the incomming sensor data and formats it into some-thing that the rest of the system can use (see Networking section for formats)

ü Sensitivity

The microcontroller may need to be adjustable to determine, for example the time characteristics of the sensor response, and the quantum of smallest sensor change that will be acknowledged as a real meaningful bit of gesture. This may provide less noisy data, and facilitate more efficient data analysis by oz

‡ 3.2 Size, Weight, Heat, Lifetime(s)

The components should in total, weigh no more than about 500 grams.
They must as small as possible, so that they can be strapped onto the body at convenient locations: e.g.

on the back
on chest
on waist
sensor on limbs

If the wearable components are scattered to separate locations on the body or garment, then the connectors become critically important. In fact, robust connections are always critically important. The wearable engineer should confer with the garment designers to integrate the components and connections into the garment, paying attention to:

Ballistics - -the kinetics of the costume can help generate more meaningful data that the sensor picks up from motion of body.

Strength/protection -- the costume may be constructed with forms of ligaments or armor that protect the wearable.

The power supply must last for at least 1 hour per person, ideally an entire day = eight 1 hour cycles.
They must not build up excessive heat since they will be worn on the body. Power supplies could perhaps be fast charging low weight cells.

The wearable must withstand the potential shock of being rolled over by an entire body, and should last at least 7 days of such use.

‡ 3.3 Latency and Bandwidth

(include notes from TWiki on bandwidth estimates)

The time resolution for a human is in the order of 30ms with rhythm perception in trained musicians being closer to 4ms (see Joel re. threshol d of perception).

‡ 4.3 Pseudo-sensors

It is possible to provide some information about the visual and audio state of the Room, as well as the internal states of components by reporting data from the subsystems to Oz as if it were sensor data. Pseudo-sensors are simply pieces of code included in those components that report activity in a standard format to the txOom system over the network.

For example; a particle density sensor; a subroutine in the particle systems which "detects" the density of particles in a specified region. a 'player existence sensor' which determines when a player enters or leaves the room.

4 Vision Tracking Component

For version 1.0 we are primarily interested in the (x,y) coordinates of each Player in the room. (In the future, shape information may also be used.)

Since there is no easily adapted off-the-shelf package we can use, we will build a custom tracking system based on computer vision techniques interpreting data from a set of video cameras.

‡ 4.1 Hardware Requirements

The vision system uses a pair of cameras about 30cm (1 foot) apart from one of edges of the ceiling (or top) for stereo information, plus a third camera suspended from above the centre of the room..

‡ 4.2 Communication with Oz

ü **From Vision system to Oz**

The vision component system is responsible for reporting the x,y position of each person in the Room at frequent intervals (aprox. 10 Hz). the Vision component will report a list of objects: $\{o_1, o_2, o_3, o_4, \ldots, o_n\}$, where n *should but may possibly not* equal the number of Players actually in the room. Each o_k = < x, y, person_id, confidence level (1-5) >. The confidence is the vision system's estimate that the object o_k is indeed person_id. For example, if a Player is obscured by another Player or moving obstacle, or if the lighting becomes too dark, the vision system will provide a progressively more uncertain guess as to the locations and identities of the objects. There must be an agreement between Oz and the Vision system on what happens when the tracking system breaks down or confuses one person for another.

messages from the vision system are in the format
/tg/0/1/ <list>
where list comprises elements in the form outlined above.

ü **From Oz to the vision system:**

The vision system may expect from Oz, a signal when a Player enters or leaves the room. This takes the form of a pseudo sensor, in the format outlined in the networking section.

5 OZ , Room Logic Component

Operationally, Oz consists of 2 main subsystems, the Sensor Map, and Room Logic.

The Sensor Map is responsible for routing sensor data to other components over the network and to the internal analysis routines. The Room Logic involves the interpretation of data from sensors (videocamera, body sensors and software pseudo-sensors), judging what is happening in the txOom system, and sending hints to the sound and visual systems based on the judgments.

The Room Logic subsystem provides macroscopic logic of how the Room responds to people's actions, and should avoid giving too detailed requests on the visual and aural output.   The other components may modify their own internal states on the basis of Oz' hints, as well as the (continuous) output from the Sensor Map .

In version 1.0, it is anticipated that the immediate responses of the room will be provided primarily by component media systems mapping of sensor data, while Oz will provide larger scale responses by using the Room Logic to alter the configuraiotn of the Sensor Map.

For further details see Oz Specification notebook.

‡ 5.1 Sub-Components

ü **Sensor Map**

The Sensor Map is responsible for filtering the raw sensor data and routing the reformatted, or modified data to the componetns that require it. Where appropriate, the sensor data is rescaled and output as a float in range 0 to 1.

Some modifications (e.g. by re-scaling, averaging, leaky accumulation and other data sluicery) of the sensor data can occur, in order to produce a looser coupling between the sensor input and the sensory output. This has the advantage of giving the system a more languid yet responsive feel to the user, and allowing slightly longer response times.

Throughput is also determined by SensorMap, and each component system may be able to ask for more/less data.

Interpolation of sensor data can happen in a number of ways,

- leaky accumulator (linear or geometric),
- zeno accumulator (linear or geometric),
- running average, tunable by averaging window.
- periodic sampling, tunable by time between datapoints.

filtering occurs with

- lo-pass, tuneable by frequency.
- hi-pass, tuneable by frequency.
- band-pass, frequency range.
- peak filters, zero crossing, or threshold.

ü Regularity testing

Oz will test for regularities in the sensor data (human motions) to help transition the Room between "passive" and "alert" states, and decide when the Room should feel more responsive to the Player. *Remember that this coupling between Player virtuosity and Room responsivity is not modeled in either the Player nor Room state alone, but must be a function of both.*

Oz has set of real-valued tests based on sensor map data to look for regularities in a Player's behavior:

Periodicity (Beat or rhythm)
Synchronicity
Auto-correlation

Each test has values [0,1], and its sensitivity will be scaled from 0 = off to 1 = full sensitivity.

One of these, the computationally cheapest and most lax test, will be turned on after the Player has exited Relation 0, which corresponds to room state "minimally sensitive". (Player cannot return to Relation 0 in any one Cycle.) As this test increases in value, the other tests will be scaled up in sensitivity.

ü State Engine

Oz keeps track of the state of each Player, and of the Room (Player 0). These states can be discrete, or continuous mixtures.

Reminder: Two states S_1 and S_2 overlap means there must be at least one continuous parameter that they share which can be used to vary between S_1 and S_2 . Not all states overlap. In practice, we should have a pretty sparse topology on Oz's state space.

The state engine changes the Player States based on the state topology and transition conditions, then modifies Oz' internal parameters (such as sensorMap parameters, or sensitivity of its regularity tests), and then suggests hints to the other components.

How Oz' communicates with Vision, Visual and Sound Components:

The main mode of output from Oz to the component systems occurs through the Sensor Map, which relays sensor data to the required destinations.

The secondary mode of communicaiton is state information.

There are occasions where the vision, visual or sound components send data back to Oz, each of these components should be able to scale each of its publicly inspectable parameters in a range 0 <= **l** <= 1, (or a predetermined format for non-scalar data) and send this data to Oz using a pseudo-sensor. (since Oz only *reads* and does not write those parameters).

The formats for this data are specified the Networking section.

6 Visual Component: Particle Operating System (POS) & 3D Graphics

6.1 Video preprocessing

For some purposes, it may be useful to supply the POS system with a video of the scene in the Room.

‡ 6.2 Particle system

ü **- integration**

- force accumulation
- integration of various forces: linear, rotational, heat (opt)

- neighbor finding algorithm
  - range search on 3D position
- data structures
  - room as collection of everything
  - sets of players
  - sets of particles
  - player data structure
    - player's particles
    - current relation
    - player pos, velocity, orientation, ang. velocity, color, etc...
    - players in "accordance"
  - three types of particle data structure
    - player particle
      - position, velocity, force accumulator, mass, orientation, ang. velocity, torque accumulator, inertia tensor, age, heat (opt.), etc...
      - list of current behaviours
    - "red" particle
      - subset of player particle attributes plus hunger
    - explosions
      - size (radius), directional, color, life, etc....
  - wind fields
    - single player vortices ("dust devils")
    - group player vortices ("tornados")
    - controls for single and group vortices
    - mappings to OZ
  - kd-tree or octtree (for neighbour finding)

ü **- generalized particle behaviours (ie heuristics)**

- scratch
- flocking
- follow path over time
- hunt
- chase food (for "red" chasing player particles)
- eat
- death
- birth
- explode
- general rules to define inter-particle interaction
- others to be defined in future

ü **- particle forces**

- linear damping
- linear dashpot
- rotational damping
- rotational dashpot
- gravity
- LJ (or similar for flocking)
- other orientation specific forces
- repulsion
- wobble
- up/down

ü **- mapping from [sub]relations to the combinations of**
forces, behaviours, integration, neighbour finding.

ü **- interface to OZ**

- communication protocol (UDP/OSC)
- definition of communication messaging protocol

- input from OZ

  - for each player: position, velocity
  - for each player: processed accelerometer data, 4 channels
  - players in "accordance", how much, and which players
  - gesture recognition
  - scaling of sensor data

- trigger events signalling transition between [sub]relations
- smooth transition data

- output to OZ

  - spec and implementation of data formats
  - particle system configuration
  - trigger events for signalling transitions between [sub]relations

ü **- communication to the rendering system**

  - spec and implementation of data formats
  - geometry to render & description of how to render
  - scripting interface (?)

- gcc/g++, cvs/rcs, debugger, lint, etc... (on linux)

- gui + slider box for testing, debugging, tuning off system

‡ 6.3 Rendering system

- communication from the particle system (see above)
- scene description at each time frame
  - geometry to render and shading model
  - virtual camera, room, lighting, etc...
- graphics library (mesa)
- gcc/g++, cvs/rcs, debugger, lint, etc...

The rendering can be farmed out to another processor to parallelize the visual computation.

- video output
- video splitter

‡ 6.4 Video post-processing

To provide a visual richness, we can present video in conjunction with synthetic computer graphics. There are four ways we can mix pre-fabricated video:

(1) Mix video from a videosource (e.g. DVD) with computer output using a separate video mixer;

(2) Use surface particles as control points to hint the morphing of the 2D video data.

(3) Put texture map into a background layer which shows through the triangles acting as alpha-keys.

(4) Texture map the prefabricated video onto the 3D structures formed by the particle system.

‡ 6.5 Communication with Oz

ü **From POS to Oz**

ü **From Oz to POS:**

7 Sound Component

‡ 7.1 Overall

- granulator
  - supercollider
- state space
  - importing + exporting state information
- communication
  - osc + scripting interface

‡ 7.2 Communication with Oz

ü **From SuperCollider to Oz**

ü **From Oz to SuperCollider:**

8 Network Communications

The sensor to computer dataflow will be sensor dependent, however dataflow between the computer based subsystems can be standardized.

‡ 9.1 Network configuration

Configure networking manually, with the following setup.

Ethernet

IP numbers	Oz;	10.0.0.2
	Vision;	10.0.0.3
	POS;	10.0.0.4
	Sound;	10.0.0.5

Wearables (configured by DHCP)

Netmask 255.255.0.0

No router, no gateway

‡ 8.2 Open Sound Control (OSC)

Communication of sensor data, and sensor map info will use the OSC protocol (a lightweight, stateless, syncronisable, packet based format) any subsystems using the sensor map may require initialisation with the current sensor descriptions + player info. Any modified sensor data, or pseudo-sensor is assigned a sensor id using the same scheme.

Since OSC uses an adressing system similar to uri syntax, we can build a standard heirachical address space, which can be accesed by the routing and messge parsing functions in the OSC library. These functions enable standard a tree traversal, or pattern matching approach to extracting the required data.

There is a C library (part of the CNMAT OSC-Kit) which provides the glue functions that components call to communicate via OSC. The library also contains functions to decode, encode and route OSC data.

more information about OSC can be found at http://cnmat.CNMAT.Berkeley.EDU/OSC/

ü port numbers

All OSC communication within the txOom network will happen on **port 5333**

‡ 8.3 Scalar Sensor Data

Each sensor value sent from Oz will be identified by a player + sensor number. The room is represented as player 0, all other players have an integer id. Bounded Scalar sensor data is scaled to a float value between 0 and 1 before being sent to any component. If the data cannot be meaningfully scaled, it is sent in the same format, without any type information. Therefore it is upto each component system to do type or bounds checking of incoming sensor data if required.

Standard format for scalar sensor data: **/tg/<player-id>/<sensor-id> /<scaled-sensor-data>**

Example: /tg/2/4/0.01865 (txOom address space, player 2, sensor 4, value 0.01865)

‡ 8.4 Sensor Configuration

Any subsystems using the sensor map may need to know the current sensor descriptions. this can happen once, for example at initialisation time, since there shouldn't be any major reconfiguration of sensors during a show.

This configuration list is intended to be a human readable guide to what sensors are available and what their numeric mappings are.
It is primarily intended for testing, debugging and system calibration/configuration. It could be used to create more meaningful labels, varaible names, etc. since the /tg/* namespace has been designed with minimum packet size (and thus network speed) in mind.

the sensor list is in the format; **/tg/sensor-setup/<player-id>/<list>**

an example list could be parsed quite easily to look like this

1 "accelerometer on arm",
2 "accelerometer on body",
3 "peaks from body movement",
4 "sluiced values from arm",
5 "particle energy",
6 "x position",
7 "y position"

Players will not necessarily have all sensors mapped. It is important that each sensor (physical or virtual) have a unique id, thus the sensor map setup is the complete list of all sensors and cooked sensor values)represented as pseudo-sensors).

‡ 8.5 Statespace Data

State information is transmitted using the same mechanism as sensor data, but for clarity states are named by a string, and assigned a coefficient, a floating point value between 0 and 1.

Standard format for State data: **/tg/<player-id>/<state-name>/<state-coefficient>**

Example state information: /tg/2/scratch/ 0.5

Room state information may contain extra information, such as matrices to initialize vector fields in the particle system, or bitmaps (as sparse matrices?) from the video tracking component system. These packets will have a similar format, but can not be assumed to contain data in the standard format. Since the room is represented as player 0, some type checking + bounds checking are recommended when parsing /tg/0/* packets.

example non-standard format: /tg/0/turbulence/init/(44,7)(133,5)(455,-3)(628,4)

(may decrease network traffic in future versions if states are assigned a numeric id in the same way as sensors - for now, state information is sent only when it changes)

‡ 8.6 Commands

Any of the component systems can query Oz for current sensor parameters, and control the rate at which they receive data from Oz. These messages use addresses in the /tg/oz heirarchy

Example messages:

/tg/oz/update-sensor-list (request for oz to send a new sensor-list)
/tg/oz/udpate-sensor/2/4 (send one value from player 2, sensor 4)
/tg/oz/sensor-rate/2/4/500 (send one value from player 2, sensor 4 each 500 milliseconds)
/tg/oz/sensor-rate/*/4/500 (send one value from sensor 4 on all players, each 500 milliseconds)
/tg/oz/sensor-rate/all/1 (send all values as quickly as possible)
/tg/oz/sensor-rate/all/0 (don't send any values)

If required, a component system can provide an internal utility function like:

SetSensorSensitivity[player-id_integer,sensor-id_integer, rate]

which would send an OSC packet containing the required messges.

(the "limit data on request" feature might not be in version 1.0, since it could adversely affect transmission speeds unless implemented efficiently.)

9 Development Schedule

TBD

10 Source and Documentation Management

‡ 10.1 Version control

ü CVS

We will be using CVS as a version control system for all parallel and interdependent computer based work, this includes documentation as well as code. Please make sure the CVS repository is kept up to date. This should ensure everyone has access to the most current version of code and documents.

For general instructions on using cvs see -> http://cvsbook.red-bean.com/cvsbook.html (on linux, man cvs)
for using MacCVS Pro (recommended) see -> http://maccvs.org/
http://sourceforge.net/projects/maccvspro/

Mac users need MacCVSPro version 2.7b3 or later, and may need Carbonlib 1.3.1.
http://downloAd.info.apple.com/Apple_Support_Area/Apple_Software_Updates/English-North_American/-Macintosh/System/Other_System/CarbonLib_1.3.1.smi.bin

The following settings need to be changed (or made)

Mac users, look in the "Session Settings" option in the "Edit" menu, and make
sure you have the correct settings ,.

```
 Checkout and Update Options
   Local Tree: set to whever you want your local copy
   Merge Policy: select automerge text and update binaries
   Default Module: tg-docs
 Remote Host Information
   Server Hostname: f0.am
   CVS Root: /space/foam/cvs/public
   Authentication method: SSH
   CVS Username: your username
   CVS Password: your passwd
   select save password for ease of memory.
```

when your first connect, you will be promted for a password.
(if you require a new user/pass for f0.am (xdv.org) i will send this to each of you individually)

if you get error messages about /var/cvs/something not being in the cvsroot, you
might have to delete your current local tree + re-checkout the required modules
(i think this is something to do with the way maccvs remembers stuff about the
old cvs repository)

on linux (subsitute 'nik@f0.am' with <you>@f0.am) -->

```
 > export CVS_RSH=/usr/bin/ssh

 > cvs -d :ext:nik@f0.am:/home/nik/f0am/cvs/public init
 > cvs -d :ext:nik@f0.am:/home/nik/f0am/cvs/public checkout tg-docs
```

‡ 10.2 Source modules

tx-docs - general txOom documents (like this one!)
tx-code - all the code
tx-code/net - networking code + libs
tx-code/oz - the code for roomlogic
tx-code/av - sound + visual systems (may have subfolders)
tx-code/wearable - software for wearable components and interfacing.
tx-notes - general notes, schedules, etc+
tx-data - data recorded from the system (or components)

All documentation (internal + external) should be kept in the tx-docs folder in the cvs repository.
html (and images) pdf (if format information is necessary) original format (if not html).

The HTML versions of the documentation will then be available online at http://f0.am/cgi-bin/viewcvs.cgi/
(this should happen automatically from the cvs repository)

please make sure that the html version is consistent with the current version (if not using html as the primary document format), since this is the version that is online and accessible to everyone.

11 Glossary and References

802.11b standard for wireless ethernet
GPL the GNU General Public Licence http://www.gnu.org/copyleft/gpl.html

Cycle is the smallest characteristic temporal interval in txOom, in the case that the environment is approximately periodic. (Nominally 45 minutes.)

Room is the physical, visual and aural space in which the txOom event takes place. At a time t, the Room is a point in the Base Manifold. (See below.)

Player is a human who visits this space in hir costume and sensors.

Trace is a visual or audio mark that a Player leaves in the Room.

Manifold is a topological space that also has a differentiable metric, and locally looks like R^n for some n.

Base Manifold M = M_people x M_visual x M_sound
  M_player=physical body and gesture positions of the players
  M_visual=public parameters of the particle and rendering system
  M_sound=public parameters of the (granular) sound system

State is a vector of numbers that descibes the current physical configuration of players and a snapshot of the software in action.

Hint is a set of tokens+parameters which Oz sends to its visual and sound systems.