Multimedia and Live Performance

Ian Willcock

Submitted in partial fulfilment of the requirements for the award of PhD

Institute of Creative Technologies
De Montfort University

December 2011
Volume Two
Bibliography


Audacity (2010) [Software] (version 1.3) Available at http://audacity.sourceforge.net/about/?lang=en


http://www.transjam.com/eternal/eternal_client.html


Coniglio, M (2008) *Isadora* [Software]
Cope, D. (1997) EMI (Experiments in Musical Intelligence) [software]

Creative Commons (no date) About – Creative Commons [http://creativecommons.org/about](http://creativecommons.org/about) (Accessed November 26, 2011)


Dewey, J. (1933) *How We Think. A restatement of the relation of reflective thinking to the educative process* (Revised edition), Boston: D. C. Heath


*How Theme Parks Work* (no date) [web page] MKPE Consulting LLC


Max6 (2011) [Software] (version 6) Cycling 74 http://cycling74.com/products/max/


Neary, D., (2005). *Linux.ie:: Subversion - a better CVS*. Available at:


http://www.education.duq.edu/institutes/PDF/papers2002/Parkany.pdf


Prior, A., (2011). Noise at the interface. In *NYHEDSAVISEN Public-Interfaces*. Public Interfaces. Aarhus University, Denmark. Available at:
http://darc.imv.au.dk/publicinterfaces/?p=114 (Accessed September 4\textsuperscript{th} 2011).


*Software for Dancers: Description* (2001) [web page]

*Software for Dancers: Participants* (2001) [Web page]


Stephenson, N. (1999) *In the Beginning was the Command Line*. Perennial.

Stockhausen, K. (1968) *Aus den siebenTagen* (Nr.26) Vienna, Austria. Universal Edition


Turner, J (2011) *Emergent Dance* [Unpublished PhD Thesis], University Of Chichester, An Accredited Institution Of The University Of Southampton


Appendix 1

Practitioner Survey Materials

Interview Overview

Firstly, thank you for agreeing to be interviewed and to allow your experience of working with Multimedia in Live Performance to inform my research. I am trying to see if there is a common set of functionality that people are using when they work with Multimedia in Live Performance – and, if there is, I will try and develop a system in collaboration with others to provide a support framework, allowing people to focus on their particular, individual artistic explorations rather than having to start right from the beginning each time a project is begun.

The interview will be structured around a framework of 9 areas of focus on multimedia use in live performance. This structure is made up of main and follow-up questions intended to enable you to provide detailed information about areas you have particular experience or knowledge, while not having to spend time on aspects that are not relevant to their practice. The framework and content are dynamic and will be continuously redesigned based on responses to completed interviews – i.e. we are not constrained by the framework, it just serves to make sure we consider certain areas of use.

Researcher’s Introduction

The interview will start with a brief explanation of why I have approached you for an interview - your work and experience in the field of inquiry. Then, I will describe the research project’s aims and methodologies and of the intentions behind the interview including the use to which data will be put. At this stage, you will be asked formally to agree to take part in the survey and for our discussion to be recorded.

The Interview

The structured part of the interview begins by me asking you to describe briefly the work you have been doing.

We will then talk through a list I have made of ways artists could use multimedia in live performance. This is both to familiarise you with the entries and also to provide the opportunity for discussion about the basis for selecting and identifying these items. You may want to identify additions or amendments you feel are required and this is very much part of the way I want to work; the list is not a set of fixed categories, but rather a set of possible starting points. You may well have other useful starting points and I expect (and hope) that as my
research proceeds, I will make changes to the list based on the suggestions of those I talk to.

My current list of ways one might use Multimedia in Live Performance is:

- Gathering input from performers or users
- Gathering information about the environment
- Outputting and presenting information
- Controlling devices and systems
- Processing information
- Planning and preparing performances
- Remembering and documenting
- Generating material
- Connecting performers or audiences

Each of the areas will then be looked at in turn. For each, you will be shown a diagram which includes both examples of technology-use in that category and names of technical resources used to produce that functionality. These items are not intended to be definitive, but are more to suggest starting points for you to review aspects of your practice. I will then ask you the main question(s) and, where it would be useful, some or all of the follow-up questions.

The whole interview should take about 45 minutes.
Gathering input from performers or users

Starting points
- Have you used digital technology or multimedia to collect information from performers (see above)?

- Have you used digital technology or multimedia to enable members of an audience to influence a performance (see above)?

Follow-up questions
- How did you actually do this gathering of information - do you remember using any of the examples in the diagram above?

- Did it work as you wanted and was it difficult to get it working as you wished?

- Did you allow performers or audience members to have control of or input to the performance in ways that are not included in the diagram above?

- Has this use of technology been central to any of your artistic intentions?
Gathering information about the environment

Starting points
- In your work have you gathered information or monitored something via technology which you've used to influence the way a performance unfolds (see above)?

Follow-up questions
- How did you actually do this information gathering - do you remember using any of the examples in the diagram above?

- Did it work as you wanted and was it difficult to get it working as you wished?

- Did you sense or monitor anything in ways that are not included in the diagram above?

- Has this use of technology been central to any of your artistic intentions?
Outputting and presenting information

- Telematics
- Sending sms messages
- Virtual performers
- Projecting video, images or text
- Sound projection
- spat
- Jitter
- Max MSP
- Isadora

Starting points

- Have you used digital technology to output or present information of any kind or to show participants (performers or audience) what is happening in a performance (see above)?

Follow-up questions

- How did you actually do this presentation of information - do you remember using any of the examples in the diagram above?

- Did it work as you wanted and was it difficult to get it working as you wished?

- Did you show the audience or performers what was happening in a performance in ways that are not included in the diagram above?

- Has this use of technology been central to any of your artistic intentions?
Controlling devices or systems

Starting points

- Have you used digital technology or multimedia to control or trigger things in performance (see above)?

Follow-up questions

- How did you actually do this controlling or triggering - do you remember using any of the technology examples in the diagram above?

- Did it work as you wanted and was it difficult to get it working as you wished?

- Did you control or trigger anything not included in the diagram above?

- Has this use of technology been central to any of your artistic intentions?
Processing information

Starting points

- Have you used technology to decide how a performance proceeds (see above)?

- Have you used live conversion or mixing during performances (see above)?

Follow-up questions

- How did you actually implement this decision-making? Do you remember using any of the examples in the diagram above?

- How did you actually do this mixing or combining? What sort(s) of information did you work with - any of the examples in the diagram above?

- Did it work as you wanted and was it difficult to get it working as you wished?

- Did you process information in ways that are not included in the diagram above?

- Has this use of technology been central to any of your artistic intentions?
Planning and preparing

Starting points

- Have you used digital technology or multimedia to prepare materials or set up cues or equipment for performances (see above)?

Follow-up questions

- How did you actually do this preparation? What sort(s) of materials were involved - any of the examples in the diagram above?

- How did you actually do this checking or validating - do you remember using any of the examples in the diagram above?

- Did it work as you wanted and was it difficult to get it working as you wished?

- Did you prepare or plan using digital multimedia in ways that are not included in the diagram above?

- Has this use of technology been central to any of your artistic intentions?
Generating material

- Generating animations
- Synthesising sounds
- Generating text or images
- Emulating performers
- Max Msp
- Soft Image
- Flash
- Director

Starting points

- Have you used digital technology or multimedia to generate or synthesise material (see above)?

Follow-up questions

- How did you actually do this generation of material(s) - do you remember using any of the examples in the diagram above?

- Did it work as you wanted and was it difficult to get it working as you wished?

- Did you generate material(s) in ways that are not included in the diagram above?

- Has this use of technology been central to any of your artistic intentions?
Connecting performers or audiences

Starting points
- Have you used digital technology or multimedia to influence how performers or audience connect physically – either with the performance or each other (see above)?

Follow-up questions
- How did you actually do this connecting - do you remember using any of the technology examples in the diagram above?

- Did it work as you wanted and was it difficult to get it working as you wished?

- Did you connect the performers or audience in ways that are not included in the diagram above?

- Has this use of technology been central to any of your artistic intentions?
Appendix 2

Survey Subjects - Bibliographical Notes

All texts are by the subjects.

**Joby Burgess**

One of Britain’s most diverse percussionists, Joby is best known for his virtuosic, often lissom performances, daring collaborations, extensive education work, and regularly appears throughout Europe, the USA and beyond.

Dedicated to the development of the percussion repertoire, often in combination with electronics, Joby spends much of his time commissioning and recording new music, with Powerplant, New Noise and ensemblebash. Recent highlights have included extensive tours with Peter Gabriel’s New Blood Orchestra and Graham Fitkin’s all star band; releases of Gabriel Prokofiev’s *Import/Export* on Nonclassical and *Frozen River Flows* on Oboe Classics.

Joby regularly performs, records and collaborates with artists including Stewart Copeland, Michael Finnissy, Graham Fitkin, Peter Gabriel, John Kenny, Akram Khan, Sarah Leonard, Joanna MacGregor, Peter Maxwell Davies, Nitin Sawhney, Andy Sheppard, Keith Tippett and Nana Vasconcelos, along with many of the world’s leading chamber ensembles. Joby is also a member of Stephen Deazley’s Edinburgh based ensemble, Music at the Brewhouse, for which he was commissioned to arrange A-ha’s pop classic *Take On Me*, for the 2008 St Magnus Festival in Orkney.

In 2004 Joby was appointed professor of percussion and director of percussion ensembles at Junior Trinity College of Music, Greenwich. Joby studied at the Guildhall School of Music & Drama, London.

**Steve Dixon**

President of Lasalle College of Arts in Singapore and Professor of Digital Performance. His 800 page book *Digital Performance: A History of New Media in Theater, Dance, Performance Art and Installation* (2007, MIT Press) is the most comprehensive study of the field to date, and has won two international book awards.

His creative practice-as-research includes international multimedia theatre tours as director of *The Chameleons Group* (since 1994), two award winning CD-ROMs, interactive Internet performances, and telematic arts events.
Andy Lavender
Professor of Theatre & Performance and Head of the School of Arts at the University of Surrey. He was previously Dean of Research at Central School of Speech & Drama, University of London. He is the artistic director of the theatre/performance company Lightwork.

He is a co-editor of two recent volumes, *Making Contemporary Theatre: International Rehearsal Processes* (Manchester University Press, 2010) and *Mapping Intermediality in Performance* (Amsterdam University Press, 2010), and co-convener of the Intermediality working group of the International Federation of Theatre Research.

Sarah Rubidge

In order to advance understanding of the intricate interplay between artistic and philosophical practice, Sarah also reflects on her practice in papers and articles that address the interweaving of the philosophical concepts that are embodied in her work. She was appointed Professor of Choreography and New Media at the University of Chichester in 2008.

Jane Turner - choreographer
has worked extensively as a choreographer/director in dance, theatre, TV/film and experimental contexts since forming her own dancetheatre co. in 1990. For the company she has consistently created innovative, visually rich "emotionally propelled choreography...formidable" The Stage, London, dance works which have been presented at leading venues throughout London, the U.K. and in Europe. Her work has drawn on extensive experience working with film for music videos, commercials and TV and particularly with animation director Mario Cavalli - their most recent collaboration being Libre (2008). For her own dance theatre touring productions she has extensively worked with image manipulation and increasingly experiments with the integration of interactive software such as Isadora as part of the performance matrix. Triggered
(2010/11) which premiered at Kings Place, London, involved music composition evolving in real time to dancers' responses to sculptures embedded with sound sensors.

Her choreographic method involves exploring the contemporary body and its multiple narratives. Her recent triptych of works explored femaleness: biology, representations, expression, to create vivid dancetheatre in: bodies, biology and the big bang Baby (2003/6); direct engagement with the theatrical Troop (2007), 'a classic seduction parade' Judith Mackrell, The Guardian; and our ritualistic relationship with Nature’s systems Herd (2009).

An interest in how the world works underpins an aesthetic that embraces the epic in the everyday, interconnections between the personal, cultural and scientific. Such ideas have informed a wide range of projects building bridges between professionals and enthusiasts, artists and organisations, via education, community and cross-arts collaborations. Ground-breaking projects working with scientists and computer programmers in close relationship with the Institute of Contemporary Arts, emergence (2002) and e-Merge (2004), fed into a recently completed PhD at the University of Chichester.


**Martyn Ware**

Born in 1956 in Sheffield, UK. After leaving school worked in computers for 3 years, in 1978 formed The Human League. Formed production company/label British Electric Foundation in 1980 and formed Heaven 17 the same year. Martyn has written, performed and produced two Human League, two BEF and nine Heaven 17 albums. As record producer and artist has featured on recordings totaling over 50 million sales worldwide - producing Tina Turner, Terence Trent D'Arby, Chaka Khan, Erasure, Marc Almond and Mavis Staples, etc..

Founded Illustrious Co. Ltd. with Vince Clarke in 2000 to exploit the creative and commercial possibilities of their unique three-dimensional sound technology in collaboration with fine artists, the performing arts and corporate clients around the world.

Martyn produces and presents a series of events entitled ‘Future Of Sound’ (21 so far) in UK and around the world and has created sonic architectural works at the British Pavilion at the Venice Architectural Biennale in 2006 amongst many others.

Martyn is also…

- Visiting Professor at C4DM at Queen Mary College, University of London
- Fellow of the Royal Society for the Arts
- Visiting lecturer at the Royal College of Art
- Visiting lecturer at the Harvard Graduate School of Design
- Founding member of the LA-based international think-tank group Matter
- Fellow of the Academy of Urbanism
- Founder member of 5D (a US based organization promoting future development of immersive design).
- Member of the Writer’s Guild of America
- Member of the advisory board for the De Montfort University ‘DMU Creative’ record label/internet radio project
- Member of the advisory board for Queen Mary College, University of London, Doctoral Training Centre in Digital Music & Media for the Creative Economy
- Curator for the main stage at the Vintage At Goodwood festival in August 2010
- Honorary Patron of the Sensoria Festival, Sheffield

He also lectures extensively on music production, technology, and creativity at universities and colleges across the world and he is Head of Sonic Experience and co-founder of the world-leading sound-branding agency - SonicID.

Recent and forthcoming projects include…

- **West Street Story** (November 2011)
  Massive 3D soundscape in the centre of Brighton as part of the White Night Festival
- **Tate Topology** (November 2011)
  3D soundscape embodying immersive three-dimensional topology manifestations at the Tate Modern
- **Beatles Immersive** (ten year install – opened July 2011)
  3D soundscape celebrating the early years of the Beatles in the brand new Museum of Liverpool
- **Michael Clarke Company** (June 2011)
Huge-scale immersive 3D soundscape to accompany Michael Clarke’s new show in the enormous Tate Modern turbine hall

- **Elizabeth Garrett Anderson Museum** (April 2011)
  3D soundscape featuring history of famous women’s medical and social campaigner at the brand new museum in London

- **Creators Project** (June-Oct 2010)
  New York, London, Sao Paolo, Seoul and Shanghai play host to the most amazingly eclectic group show of installations, talks, performance and events. Illustrious has created a 3D spatialization of Nick Zinner’s composition for his immersive photographic and sound installation called A.D.A.B.A.

- **Yves Saint Laurent, ‘Belle D’Opium’ Five Senses installation** (June 2010)
  Together with Skrapic Consulting in Paris, Illustrious created a 3D soundscape composition together with 360 degree projection, theatrical lighting, rotating stage and olfactory installation to create an immersive impression of the essence of their brand new perfume.

- **Pret-A-Porter Paris** (January 2010) – a large scale immersive 3D soundscape created for this Paris fashion trade fair based around the sounds and impressions of the various quarters of the city – incorporating contemporary and historical recordings to create a 'magic realist' impression of Paris.

- **3D DJ Soundclash** (February 2010) – Two DJ teams (Warp Records and Ninja Tunes) battle it out in a huge Illustrious 3D soundscape underneath the Royal Albert Hall – in collaboration with Red Bull Music Academy

- **Soundlife London** (June 2009) – a giant 3D soundfield one-hour looping composition distilling the sonic essence of London into three dimensional space occupying the whole of Leicester Square in June 2009 for 10 hours a day for 10 days – playing to over 2 million people. This was created in collaboration with a variety of community groups, schools, ethnic associations and reminiscence groups and in association with the City Of Westminster Council

- **Large Scale Audio Immersion Experiment** (November 2009) – in collaboration with Goldsmiths College, London and Duran Audio, Illustrious created a 70m x 70m outdoor sound installation using unique Intellivox speakers controlled by Wave Field Synthesis technology and 3DAudioScape. A competition was held for students’ soundscape compositions and the winning pieces were exhibited in the space.

- **Acoustic Ecologies** (2010-2012) – in collaboration with arts organization Education4Conservation, a series of works entitled ‘Nocturne/Wild Echoes’ based upon the notion of outdoor immersive soundscapes in countryside environments, creating unexpected sonic interventions in natural environments.

- **Breathing Trees** (2010) – a collaboration between light artists Creatmosphere and Illustrious, the third iteration of this exhibition will take place in Lima, Peru and will feature a huge scale (200m x 50m x
25m) installation in a park in the centre of the city based on the concept of wordless communication in sound and light between two trees, one male and one female.

- **Threeways SEN School Sensory Studio** (2006–2012) – this ongoing project is at the core of the working philosophy of Illustrious – a world-first Sensory Studio for autistic, disabled and SEN pupils using state-of-the-art technology to create a highly responsive, immersive 3D sound, light and HD projection environment which can be easily controlled and modulated using a variety of sensors and controllers by teachers and the pupils themselves.
Appendix 3

eMerge Performance Scripting Guide

Version 1.4 – 9th November 2011

Ian Willcock

ian@willcock.org
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>I (300)</td>
</tr>
<tr>
<td>Language overview</td>
<td>1 (301)</td>
</tr>
<tr>
<td>Events</td>
<td>6 (306)</td>
</tr>
<tr>
<td>Commands</td>
<td>14 (314)</td>
</tr>
<tr>
<td>Keyword Listing</td>
<td>21 (321)</td>
</tr>
</tbody>
</table>
Introduction

The eMerge system is a networked system that tracks performance events in real-time and is then able to issue commands to performers based upon a rule-set which has been created before the performance.

The eMerge scripting language, which controls the decision making and cue-issuing process, is based on simple text-based definitions of rules, events and commands. These are written by the user(s) in everyday language, and are then interpreted by the system. When you enter a rule or a command, the system checks that what you have entered contains the information it needs. It then either stores the information if it part of performance preparation, or it carries out the requested action(s) if it is a command for immediate execution.
Language Overview

The eMerge system

The eMerge system connects live performers and other system-level sensors to a central server. The system has two main modes – a pre-performance, preparation state and an active, performance state.

In the preparation state, the system configuration is defined and checked and the rule structures that will determine how the performance will unfold are entered and edited. In performance, the server stores information about what is happening and continually compares what has happened against a stored set of circumstantial directives – e.g. if such and such an event occurs, carry out this action.

Scripting language syntax

The basic structure of the eMerge scripting language is the Rule. These are entered before a performance and are stored in the system. They define what the system should do during a performance when a particular set of events occurs. Rules can be active or inactive – so that responses to sets of events can change as a performance progresses.

When the triggering events occur for a rule, its command(s) are executed and the triggering events are cleared from the system’s memory (so that the rule does not trigger again straight away). The rule itself however remains active unless it specifically makes itself inactive through one of its commands.

The other type of input is a Command. When a command is entered on its own (as opposed to when it is a part of a rule – see below), the system will carry out the requested action immediately. Commands can be entered at any time - while the system is in preparation mode or during a live performance.

Commands can cause a wide variety of actions to be carried out, these include causing information to be sent to one or more performers, or changes to the internal storage and operation of the emerge system itself.
**Rules**

Rules determine how the system operates when it is running a performance. A rule has two parts: events and commands. If an event happens during a performance, the command will be issued:

<table>
<thead>
<tr>
<th>If</th>
</tr>
</thead>
<tbody>
<tr>
<td>ian says “boo”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Then</th>
</tr>
</thead>
<tbody>
<tr>
<td>say “go” to tom</td>
</tr>
</tbody>
</table>

Rules can be triggered by any one of a number of events happening, and can carry out more than one command:

<table>
<thead>
<tr>
<th>If</th>
</tr>
</thead>
</table>
| ian says “boo”  
Jane says “boo”  
tom says “boo”  
sonja says “boo” |

<table>
<thead>
<tr>
<th>Then</th>
</tr>
</thead>
</table>
| say “go” to sven  
show “start playing section four” to craig  
start section four |

Here, if any one of the named says (types) “boo” then the commands will be issued.

The events needed to trigger a rule can be also combined using ‘and’:

<table>
<thead>
<tr>
<th>If</th>
</tr>
</thead>
</table>
| jane says “boo”  
and  
Section three is active |

<table>
<thead>
<tr>
<th>Then</th>
</tr>
</thead>
</table>
| say “go” to sven  
show “start playing section four” to craig  
start section four |
Events

An event specification has three parts; a reference to a performer or other system object, a description of something they might do – their state - and the test for the event to be judged as having happened.

Here are some events:

<table>
<thead>
<tr>
<th>Object reference</th>
<th>test</th>
<th>What</th>
</tr>
</thead>
<tbody>
<tr>
<td>nick</td>
<td>says</td>
<td>“stop”</td>
</tr>
<tr>
<td>rule SysName_RULE_2</td>
<td>is</td>
<td>active</td>
</tr>
<tr>
<td>performance time</td>
<td>is more than</td>
<td>10 minutes</td>
</tr>
<tr>
<td>zone view1 x</td>
<td>is less than</td>
<td>3.5</td>
</tr>
</tbody>
</table>

The reference part of an event must be a word or phrase that refers to something the system knows about and holds information about (see the detailed section on referring for more information).

Things the system knows about include:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>performers</td>
<td>zones</td>
</tr>
<tr>
<td>rules</td>
<td>sections</td>
</tr>
<tr>
<td>the performance</td>
<td></td>
</tr>
</tbody>
</table>

There are four tests that may be applied to judge if the event has happened. The user can use different words and symbols to specify them. Here are just some ways of writing the four tests:

<table>
<thead>
<tr>
<th>equals</th>
<th>does not equal</th>
<th>is higher than</th>
<th>is lower than</th>
</tr>
</thead>
<tbody>
<tr>
<td>is</td>
<td>isn’t</td>
<td>is more than</td>
<td>is less than</td>
</tr>
<tr>
<td>=</td>
<td>≠</td>
<td>&gt;</td>
<td>&lt;</td>
</tr>
</tbody>
</table>
Commands

Commands make the system do something. They may be triggered by events happening or they may be entered directly by a user at any time – even when the system is not running a performance.

Commands have two or three parts; an action to be carried out, a reference to a performer or other system object which is the target for that action and, where required, a description of what is to be done or sent to that target.

Here are some commands:

<table>
<thead>
<tr>
<th>Action</th>
<th>Object reference</th>
<th>What</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>performance</td>
<td></td>
</tr>
<tr>
<td>say (to)</td>
<td>Hannah</td>
<td>“start singing”</td>
</tr>
<tr>
<td>set</td>
<td>rule 3</td>
<td>INACTIVE</td>
</tr>
</tbody>
</table>

The reference part of a command must follow the same rules as for events – it must be a word or phrase that refers to something the system knows about and holds information about.

The action definitions that can be used depend partly on what the system and its components (performers, editors, sensors etc.) can do. Some of the words that will always be understood by the system are:

<table>
<thead>
<tr>
<th>say</th>
<th>set</th>
<th>show</th>
<th>play</th>
</tr>
</thead>
<tbody>
<tr>
<td>give a</td>
<td>start</td>
<td>stop</td>
<td></td>
</tr>
<tr>
<td>delete</td>
<td>get</td>
<td>make</td>
<td></td>
</tr>
</tbody>
</table>
Referring to Things or Objects

When a rule or command is entered into the eMerge system, the user has to provide enough information for the system to be able to identify exactly which information should be examined or sent – depending whether the reference occurs in a rule or a command.

The system works this out by examining the object reference you provide and (particularly for events) the test you specify. If a word is not one of the built-in system words, it is assumed to be the name of an object. For rules, sections, and zones you need to specify its type:

- zoneleft_space
- section introduction
- rule SysName_RULE_21

For performers, you just need to use the name they are logged in as:

- jane
- ian

Object names cannot contain spaces or punctuation marks.

Describing States or Values

There are four different ways of describing what value should be tested or sent to a system component.

<table>
<thead>
<tr>
<th>generalised descriptions</th>
<th>ANYTHING, NOTHING, EMPTY, ACTIVE, INACTIVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolute descriptions</td>
<td>“hello”, 125, “F# 3”, 2 minutes</td>
</tr>
<tr>
<td>references to data</td>
<td>image “symbol1”, sound “low buzz”, midi “riff”, movie “gesture 7”</td>
</tr>
<tr>
<td>references to objects</td>
<td>zone upstage, rule SysName_RULE_21, performance time</td>
</tr>
</tbody>
</table>
Events

The eMerge system is able to keep track of, and respond to, a large number of different sorts of events. When specifying events in rules, there are often additional pieces of information about the events that can be included in rule definitions. The nature of an event is specified in rules by a combination of the choice of ‘action’ word and information type. Events may be associated with a particular performer – or with a system-level input device such as a pressure pad or ambient light level sensor.

Identifying event sources

Performers and the system-level objects (zones, sections and rules) are identified by name.

The ANYONE and NOONE keywords are also allowed.

Examples

(if) Jane gives a mouse signal
(if) ANYONE gives a mouse signal

System objects are referred to by name.

Examples

(if) performance time is more than 5 minutes
(if) section coda time is 2 minutes
(if) rule SysName_RULE_3 is ACTIVE
(if) zone myZone x-position is 2.0

After the source identification, an event definition must contain information about what sort of input and what value that input might have to be considered meaningful. The types of input that are currently implemented include; signals (of various types), text and midi.
Signals

Signal events are simple cues generated by the user or device interfacing with a client. They could include such things as mouse clicks, a key being pressed on the computer, a sudden significant change in sound level (such as a performer saying “bah”) or a sensor sending a message about location etc.. Each signal is sent to the system together with information about what sort of event it is. This additional information can be used or not as a situation demands.

Examples
The general form, which will trigger on any signal event is: -

(if) Dean gives a signal

To specify the type of input, use a type keyword in front of signal: -

(if) Dean gives a mouse signal

To specify additional things, add keywords (see below for which words can be used with each input type): -

(if) Dean gives a double mouse signal

<table>
<thead>
<tr>
<th>mouse signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. Dean gives a mouse signal</td>
</tr>
<tr>
<td>This type of signal is generated in response to a mouse click. It has 3 possible additional properties, <em>short</em>, <em>long</em> and <em>double</em>.</td>
</tr>
<tr>
<td><strong>short</strong></td>
</tr>
<tr>
<td>e.g. Dean gives a short mouse signal</td>
</tr>
<tr>
<td><strong>long</strong></td>
</tr>
<tr>
<td>e.g. Dean gives a long mouse signal</td>
</tr>
<tr>
<td><strong>double</strong></td>
</tr>
<tr>
<td>e.g. Dean gives a double mouse signal</td>
</tr>
</tbody>
</table>
### sound signal

**e.g. Claire gives a sound signal**

This type of signal is generated in response to a sudden sound pulse often through a performer’s headset. It has 3 possible additional properties, *short*, *long* and *double*.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>short</strong></td>
<td>a short sound, for example “ki”.</td>
<td>e.g. Claire gives a short sound signal</td>
</tr>
<tr>
<td><strong>long</strong></td>
<td>A sound in which a high level is maintained for more than half a second, for example “shar”.</td>
<td>e.g. Claire gives a long sound signal</td>
</tr>
<tr>
<td><strong>double</strong></td>
<td>A double sound impulse, where the second sound starts within a third of a second of the first, for example, “di ka”.</td>
<td>e.g. Claire gives a double sound signal</td>
</tr>
</tbody>
</table>

### key signal

**e.g. Ethan gives a key signal**

This type of signal is generated when the performer presses and releases a single key on their computer’s keyboard. It has 3 possible additional properties, *short* and *long* together with the letter value of the key being pressed.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>short</strong></td>
<td>a ‘normal’ quick key press.</td>
<td>e.g. Ethan gives a short key signal</td>
</tr>
<tr>
<td><strong>long</strong></td>
<td>A key press where the key is held down for more than half a second before being released.</td>
<td>e.g. Ethan gives a long key signal</td>
</tr>
<tr>
<td><strong>Letter value</strong></td>
<td>The letter value of the key – for character keys, this is the character that would appear on screen if only that key were pressed, the letter is placed in speech marks. For control keys the following keyword can be used: - RETURN. SPACE is also allowed instead of “ “.</td>
<td>e.g. Ethan gives a “z” key signal ethan gives a RETURN key signal</td>
</tr>
</tbody>
</table>
Text events

Performance events involving text can, in principle, be of two types; typed or spoken (only the former is currently implemented). They are distinguished through the use of different keywords; types and says. Both types of event need the information to be matched to be specified either using a string of characters enclosed in speech marks or through a keyword. At the present time, both are synonymous in event descriptions.

Allowable keywords are; NOTHING and ANYTHING.

Examples

(if) susan types “welcome”

(if) ali types ANYTHING

If speech to text is implemented, the following can be used

(if) kate says “hello”

(if) mike says NOTHING

types

<table>
<thead>
<tr>
<th>e.g. Molly types “I wandered lonely as a cloud”</th>
</tr>
</thead>
</table>

This type of event gives access to text input. The event is sent when the performer presses the RETURN key.

Note that the performer client cannot support both typed text input and key and mouse signals at the same time.
MIDI Events

Performance events involving MIDI can be collected by the performer client in 2 different ways – as individual events or completed notes. These collection settings are set in the performer client.

*Events* generates a midi signal each time certain sorts of midi events are received by the client. They are described in event specifications by the keywords `gives a`, see below for details.

*Notes* generates a separate performance event for each note that is played (i.e. one event for each noteOn-noteOff pair) and are described in event specifications using the `plays` keyword.

The following 2 collection strategies may be implemented in future versions:

- **Phrases** generates a list of notes played whenever the player pauses for more than 2 seconds and no notes are being held.

- **Statements** are similar to phrases but they are generated (as a performance event which includes a list of notes played) whenever the player presses a particular key (usually the top or bottom note on a keyboard).

MIDI note events are specified using the `plays` keyword together with a short piece of text identifying the pitch to be matched. Notes are specified by letter name (in upper case) and the modifiers ‘#’ for sharp and ‘b’ for flat. There should then be a space and a number indicating which octave is wanted. The whole note specification should be contained in speech marks. The keywords NOTHING and ANYTHING are allowed.

Examples

- (if) sophie plays “A 4”
- (if) ian plays “F# 2”
- (if) frank plays ANYTHING

<table>
<thead>
<tr>
<th>plays</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. rashid plays “C 4”</td>
</tr>
</tbody>
</table>

This type of event gives access to high level midi input. When the event is sent depends upon the performer client settings. It will be when one of the following conditions is met:

- Note - when a noteOn-noteOff pair is completed

Individual MIDI events are captured as signals
### midi signal

e.g. yolanda gives a midi signal

This type of signal gives event-level access to midi input devices. It generates a lot of data and *midi input will generally be better gathered using higher level MIDI events* (see above). It has 2 possible additional properties, `eventType` and `note`.

<table>
<thead>
<tr>
<th><code>eventType</code></th>
<th>The type of MIDI event that generated the signal. Possible types are noteOn, noteOff, controlChange, programChange and pitchBend.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>e.g. yolanda gives a noteOn midi signal</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><code>note</code></th>
<th>A short piece of text which identifies the note (if any) that is associated with the MIDI event. Middle C is identified as “C 3”. Notes are specified by letter name (in upper case) and the modifiers ‘#’ for sharp and ‘b’ for flat. There should then be a space and a number indicating which octave is wanted. The whole note specification should be contained in speech marks.</th>
</tr>
</thead>
</table>
|        | e.g. yolanda gives a “G# 5” midi signal  
yolanda gives a “Bb 0” midi signal  
Note that the above examples will trigger on both the starts and ends of midi notes. To get a single notification for each note that is played, use the dedicated MIDI event type (see *play* above). |
**System events**

The system is able to generate events arising from the *performance*, time based performance divisions called *sections*, representation of physical locations called *zones* and the state of *rules*.

---

**performance**

- e.g. performance is active

Performances have 2 qualities that can be used in specifying events; activity and elapsed time.

When a performance is started, it is set to active and its clock starts counting the seconds since the start performance command was issued. To create an event which is fired at a particular point in a performance, test for the number of seconds:

- performance time is 120
- or minutes: 
  - performance time is 2 minutes
- ‘more than’ and ‘less than’ are also allowed: 
  - performance time > 45
  - performance time is more than 5 minutes
  - performance time < 360
  - performance time is less than 6 minutes

---

**zone**

- e.g. zone zone1 is populated

Zones are ways that physical locations can be dynamically represented by the system. They have 2 qualities; whether they are populated or empty and a 3 dimensional position.

Zones are identified by a name:

- zonemyzone

You can use sense data to update their properties (see commands section) and then make rules which fire when a zone is ‘in’ a particular location:

- zonemyzone position is (1.5,2.0,0.0)
or when the zone is populated: -
zonemyzone is populated

or when one of the zone’s position co-ordinates is equal to, less than or more than a specific value: -
zonemyzone x is 5.0
zonemyzone y is less than 1.3
zonemyzone z > 10.5

section

e.g. section interlude1 is active

Sections are ways that time-based divisions of the performance can be dynamically represented by the system. They have 2 qualities; whether they are active or inactive and an elapsed time.

Sections are identified by a name:-

section intro

Sections are created when a new section name is encountered by the system, but they are not started (set to active and their clock reset to 0 and started) unless the start command is used: -

start section intro

Sections are stopped by the stop command: -

stop section intro

When a section is started, its activity can be used to fire events in rules: -

(if) section intro is active

Its elapsed time, counted in seconds, can also be used in rules (until a section is started, it remains at 0):-

(if) section intro time > 4 minutes
(if) section main time is less than 20
Commands

Commands cause the system to do something. They can be issued by the user or triggered by a rule if its triggering circumstances arise. All commands require a target reference (e.g. performer 2, rule 7), although this is implicit in a small number of cases (see entry for 'get' below) and a data value. The particular action triggered by a command is often dependent on both the keyword used and the data type supplied.

The ordering of elements in a command usually follows one of 2 syntactical models, as far as possible following that of 'natural' English usage for each keyword. In all cases, the command keyword must be the first item.

Examples

- set zone upstage to populated  (keyword-target-data)
- say “hello” to kevin  (keyword-data-target)

Commands can be divided into 2 main categories based on the type of action they cause to happen; cuing and data management commands.

Cuing commands

These cause messages or orders to be sent to performer clients, usually accompanying one or more pieces of data and requesting the client to display it to the performer in an appropriate manner.

Examples

- say “hello” to daniel  (speaks)
- play “D# 4” to daniel  (plays a tone)
- show “start section 7” to daniel  (displays text)
- show image score1.gif to daniel  (displays image)

Data management commands

These direct the system to carry out actions on its own internal data representation. This could cause a change in the system operation, for example starting and stopping performances, or they might alter the database of stored rules.

Examples

- start section intro
- set rule 2 to INACTIVE
- set zone camera1 position to (1.2,2.0,0.0)
Identifying targets

Performers and rules are identified by name. If the first word of a target is not a keyword, it is assumed to be the name of a performer or rule.

The EVERYONE keyword is also allowed for cuing commands (those whose target is a ‘performer’).

Examples

say “hello” to joseph
show image image1.jpg to EVERYONE

Cuing Commands

<table>
<thead>
<tr>
<th>say</th>
<th>saydata to target</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. say “hello” to michelle</td>
<td></td>
</tr>
</tbody>
</table>

The say command sends a text string to a performer client and asks it to speak it to the performer. The text to be spoken should be enclosed by speech marks.

The character sequences .txt and .html are not allowed in text strings as they are used to distinguish strings from filepaths (future performance clients will offer the capability to read from text files).

If the performer client does not have speech capability, the text will be displayed on screen instead.

<table>
<thead>
<tr>
<th>give</th>
<th>give a dataeventTypesignal to target</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. give a “F# 3” noteOn midi signal to mixer</td>
<td></td>
</tr>
</tbody>
</table>

The give command causes the named client to generate a single MIDI event. It has 2 required additional properties, eventType and note.

The noteOn and noteOff signals require a pitch, other MIDI signal types require an integer.

<table>
<thead>
<tr>
<th>eventType</th>
<th>The type of MIDI event that generated the signal. Possible types are noteOn, noteOff, controlChange, programChange and pitchBend.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>e.g. give a “F 5” noteOn midi signal to sarah</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>note</th>
<th>A short piece of text which identifies the note or integer value that is associated with the MIDI event.</th>
</tr>
</thead>
</table>
### show

<table>
<thead>
<tr>
<th>showdata to target</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. show “hello” to monica</td>
</tr>
</tbody>
</table>

The show command sends a text string or an image to a performer client and asks it to display it on the computer’s monitor.

Note that the current version of the performer client cannot display both text and images at the same time. However other modes of feedback (speech, MIDI etc.) do not affect the visual display.

When a performer client displays a new text or image item, it displaces whatever was previously being displayed.

Displaying text from a file is not yet implemented.

### text

For text, the data expression may be either an absolute expression or a file reference (only the former is currently implemented). For absolute expressions, speech marks should enclose the text.

- e.g. show “start playing now” to Jane

- show polemic.txt to Tony

The character sequences .txt and .html are not allowed in absolute expressions as they are used to distinguish strings from filepaths. File names may not include spaces.

For files, the text should be formatted as plain text or as (simple) html.

### images

For images, only a file reference is allowed which must be preceded by the keyword image. File names may not include spaces. Images should be stored in jpeg or gif format.

If the image is larger than the display area (950 wide, 600 or 485 high depending on the keyboard input mode) will be scaled to the display area’s size. The proportions of the image are preserved.

- e.g. show image xmas_card.jpg to EVERYONE

The image will first be downloaded to the performer client’s computer – which may cause a delay if files are large and networks are slow. However, the client caches all downloaded assets locally so that they are available instantly for subsequent use.
The play command sends a text string identifying a MIDI note or a file reference to a performer client and asks the client to play the file through the appropriate playback system. The file may be either a MIDI file or a sound file – the type is indicated using the `midi` or `sound` keywords. File playback is not yet implemented.

Note that playing back a sound file may interfere with spoken feedback.

When a performer client receives a request to play a new piece of sound or MIDI information, it interrupts any playback that is already taking place.

There is currently no way to set instruments for MIDI playback using rules, although this can be done manually in the performance client.

Playback is currently restricted to a single MIDI channel – which is selectable in the performance client.

Only MIDI note playback is currently implemented.

### MIDI note
Notes are specified by letter name (in upper case) and the optional modifiers ‘#’ for sharp and ‘b’ for flat. There should then be a space and a number indicating which octave is wanted. The whole note specification should be contained in speech marks.

- e.g. play “Bb 4” to Jane

### MIDI file
For MIDI files, a file reference, must be preceded by the keyword `midi`. File names may not include spaces.

- e.g. play midi fanfare1.mid to performer 5
- e.g. play midi 10.0.1.1/midi/fragment6.mid to performer 5

### Sound file
For sound files, a file reference, must be preceded by the keyword `sound`. File names may not include spaces.

Permissible sound file formats are .aif, .wav and .mp3

- e.g. play sound stretched.aif to performer 5
**Data and system management commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>get</strong></td>
<td>The get command causes a data object in the system to be retrieved and returned to the target (which has to be <em>me</em>, the requesting client at present)</td>
</tr>
<tr>
<td>e.g. <code>get me rule SysName_RULE_14</code></td>
<td></td>
</tr>
<tr>
<td><strong>rule</strong></td>
<td>For rules, the events and commands making up a rule are returned in an XML format. At present, a rule’s state and labels are not returned.</td>
</tr>
<tr>
<td><strong>rule-list</strong></td>
<td>Returns an XML listing of all current rule’s (unique) system names. These are in the form of SysName_RULE_n, where n is an ID number</td>
</tr>
<tr>
<td>e.g. <code>set rule 2 to INACTIVE</code></td>
<td></td>
</tr>
<tr>
<td>e.g. <code>set zone origin position to (2.5, 2.5, 5.0)</code></td>
<td>The set command causes a data object in the system to be loaded with a new value. There are a number of allowable targets each of which affects the permissible values for <em>data</em>; performance, rule, section (not implemented).</td>
</tr>
<tr>
<td><strong>rule</strong></td>
<td>For rules, data can have two values, ACTIVE and INACTIVE. If a rule is active, its conditions are considered by the system when it judges if any significant events have taken place.</td>
</tr>
<tr>
<td>e.g. <code>set rule 2 to ACTIVE</code></td>
<td>A rule can make itself inactive as one of its commands – meaning that it will only be triggered once.</td>
</tr>
<tr>
<td><strong>section</strong></td>
<td>For sections, data must be a text string.</td>
</tr>
<tr>
<td>e.g. <code>set section “transition 2” to active</code></td>
<td>Assigning section a new name (even to the same name) causes the system to reset the section time. Section can thus be used in event definitions to make commands conditional on stages of a performance. Progress through a section is then given by section time.</td>
</tr>
<tr>
<td>e.g. <code>(if) section time &gt; 1 minute</code></td>
<td></td>
</tr>
<tr>
<td><strong>performance</strong></td>
<td>A performance can be either ACTIVE or INACTIVE. When the system starts, all performances are set to inactive with a performance time of 0.</td>
</tr>
</tbody>
</table>
Setting performance to active can also be done using the `start` command (see below).

e.g. set performance to active

If a performance is active, its performance time will be continually increased. When it is stopped (by setting it to inactive), the performance time is reset to zero.

### zone

A zone can be set to both a position and to ‘populated’ or ‘empty’. Zones are referred to by name.

- e.g. set zone upstage to populated
- set zone avater1 position to (0.0, 3.1)

If the zone name has not been previously encountered by the system in the project you are working in, it is created.

Positions can be 2 dimensional or 3 dimensional and are specified 2 or 3 numbers within parentheses, separated by commas:

- e.g. (3.2,1.0,-2.3)

---

**start**

<table>
<thead>
<tr>
<th>e.g. start section introduction</th>
<th>start performance start section name</th>
</tr>
</thead>
</table>

The `start` command is a synonym for set `formalUnit` to active (see set section above)

It can be used with performance or section, when a name for the section ust be supplied.

When the command is executed, the performance or section clock is started and is updated until the section or performance is stopped, reset or set to inactive.

**performance**

For the performance, no further information is needed.

- e.g. start performance

The performance is always set to stopped (or inactive) when it is created or loaded from the database.

**section**

For sections, if the section has already been involved in the performance as either an event source or as the target of a command, it is set to active and its internal elapsed time clock is started.
If it has not been involved in the performance, it is created and its clock started.

e.g. start section introduction

Sections are not saved by the system – they are created as needed during a performance (this process should be transparent to the user – if you mention a section and it does not already exist, it will be created)

<table>
<thead>
<tr>
<th>stop</th>
<th>stop performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. stop section introduction</td>
<td>stop section name</td>
</tr>
</tbody>
</table>

The stop command is a synonym for set `formalUnit` to inactive (see `set section` above)

It can be used with performance or section, when a name for the section must be supplied.

When the command is executed, the performance or section clock is stopped and set to 0.0.

<table>
<thead>
<tr>
<th>performance</th>
<th>section</th>
</tr>
</thead>
<tbody>
<tr>
<td>For the performance, no further information is needed.</td>
<td>For sections, if the section has already been involved in the performance as either an event source or as the target of a command, it is set to inactive and its internal elapsed time clock is stopped and set to 0.0.</td>
</tr>
<tr>
<td>e.g. stop performance</td>
<td>e.g. stop section introduction</td>
</tr>
<tr>
<td>The performance is always set to stopped (or inactive) when it is created or loaded from the database.</td>
<td>Sections are not deleted by stop so that their activity may still be used in event descriptions</td>
</tr>
</tbody>
</table>

If it has not been involved in the performance, it is created.

e.g. stop section introduction

Sections are not deleted by stop so that their activity may still be used in event descriptions

e.g. (if) section introduction is inactive

However, note that there is currently no way to test if a section has ever been active.
Keyword listing

ACTIVE
ANYONE
ANYTHING
controlChange
delete
double
event
EVERYONE
get
give a
gives a
image
INACTIVE
is
is more than
is less than
key
long
make
midi
minutes
mouse
movie
NOONE
noteOff
noteOn
NOTHING
osc
Performer
performance
performance time
pitchBend
plays
position
programChange
RETURN
rule
rule-list
says
seconds
section
section time
short
show
signal
sound
SPACE
time
to
trigger
types
x
x-position
y
y-position
z
z-position
Appendix 4

Live Interactive Multimedia Performance Toolbox (LIMPT) System Software

Please see the file Read-Me.rtf on the DVD for a manifest of the software submitted as a part of this thesis.
Software library – rationale

As part of the multimedia server system development process, a software library containing a number of packages have been produced. These are intended to formalise and focus the development along best-practice software production guidelines and also to simplify the production of third-party specialised clients for different platforms and purposes since they can be used with a limited or customised GUI.

The library is currently implemented in ActionScript 2.0, although migration to AS3.0 is planned.

The latest versions of all packages are available from the project repository, http://bartleby.ioc.t.dmu.ac.uk/repos/mm_server/client/

Software packages - documentation

<table>
<thead>
<tr>
<th>app</th>
</tr>
</thead>
<tbody>
<tr>
<td>GlobalHooks</td>
</tr>
<tr>
<td>AppManager</td>
</tr>
<tr>
<td>AppPreferences</td>
</tr>
<tr>
<td>GuiManager</td>
</tr>
<tr>
<td>StatusManager</td>
</tr>
</tbody>
</table>

The app package contains classes that are highly customised for a particular client implementation; it is anticipated that they will be largely rewritten for each new client implementation.

GlobalHooks
Provides a single instance, global point of access to application components.

AppManager
Controls the startup process of the Flash data structures and managers. Hands control to the GUI manager when loading and initialisation etc. are complete.

AppPreferences
A CDT for global application preferences.

GuiManager
Controls the frame shown by a Flash GUI timeline; it allows different system components to reliably change the GUI display without interference.

StatusManager
Keeps track of client status and notifies any objects that have registered to receive util.IntDataObj messages.
The component package contains classes that handle Flash GUI and input/output components for a client. These are loaded dynamically since different client implementations may well require different collections of settings and opportunities for user customisation.

**CapabilityDB**  
This class maintains a database of a client’s input and output media handling capabilities and which components provide them. When a message arrives from the central controller requesting that information is output in a particular way, this class returns the appropriate component reference to the `message.Despatcher` class.

**CapObject**  
The custom data type for the capabilityDB class.

**CompData**  
The custom data type for the ComponentManager.

**CompLoader**  
A simple class that loads a single Flash .swf component and reports the result. Used by the ComponentLoadManager.

**ComponentLoadManager**  
A helper class for the ComponentManager that is responsible for loading all components.

**ComponentManager**  
The main class that handles Flash GUI components. It maintains a database of loaded components and routes system calls to them. It also deals with getting and setting user preferences for client components.

**Preferable**  
An interface which components must implement if they are to use the user preferences services provided by the client (through the Preference Manager).
The **lexical** package contains classes that support editing and entry of rules on the central controller.

**Command**
This is an abstract data type that represents a stand-alone command in the rule editor.

**Rule**
This is an abstract data type that represents a single rule in the RuleEditor.

**RuleEditor**
A simple database class that maintains a local list of rules in the current performance. It also parses XML representations of rules (received from the central controller) into entries in its database so they can be displayed or edited.

**RuleEditorServerProxy**
A class to handle communication with the server for the RuleEditor.

---

### user

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UserData</strong></td>
</tr>
<tr>
<td><strong>UserManager</strong></td>
</tr>
<tr>
<td><strong>PreferenceManager</strong></td>
</tr>
</tbody>
</table>

The **user** package is a simple database system which allows a single client to be operated under different user names. This is optional and will not be required for all implementations. Although support for a user password is provided, passwords are not encrypted and are stored as plain text (they are hidden from other users, but vulnerable).

**UserData**
The custom data type for the UserManager.

**UserManager**
A class providing a database of users and passwords. The username is attached to all events passed to the central controller and determines which part of its state variables are updated.

**PreferenceManager**
A class that loads and saves preferences for a user for all software components that implement the *Preferable* interface.

<table>
<thead>
<tr>
<th>network</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
</tr>
<tr>
<td>ConnectionsDB</td>
</tr>
<tr>
<td>NetworkManager</td>
</tr>
<tr>
<td>SocketController</td>
</tr>
</tbody>
</table>

The *network* package contains classes that maintain a database of available connections and also provide a single, global point of access to control and use the network connection.

Connection
A custom data type for the *ConnectionsDB* class.

ConnectionsDB
A simple database of network connections.

NetworkManager
A class that is responsible for making and ending network connections and for providing a global point of access for sending and receiving information.

SocketController
A low-level socket class which provides an interface between a tcp/ip socket and the *NetworkManager* instance. It is based on code by Colin Moock. Other classes could be used to provide similar connectivity using different protocols (e.g. OSC)

<table>
<thead>
<tr>
<th>xml</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMLBuilder</td>
</tr>
<tr>
<td>XMLParser</td>
</tr>
</tbody>
</table>

The *xml* package provides helper classes that make manipulating XML data easier. They also ensure that changes in the base XML API (especially those in AS 3.0) will not require extensive refactoring of classes that use XML data.

XMLBuilder
This class provides a number of static functions which allow the user to create XML document elements.

XMLParser
This class defines an object that loads and parses XML specified as a URL or parses XML passed as a string. It also provides static methods for accessing data in an XML document.

<table>
<thead>
<tr>
<th>util</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppProperties</td>
</tr>
<tr>
<td>ConsoleLogger</td>
</tr>
<tr>
<td>DetailedError</td>
</tr>
<tr>
<td>HostInterface</td>
</tr>
<tr>
<td>IntDataObject</td>
</tr>
<tr>
<td>NetInterface</td>
</tr>
<tr>
<td>Observable</td>
</tr>
<tr>
<td>Properties</td>
</tr>
<tr>
<td>PropertyLoader</td>
</tr>
<tr>
<td>Timer</td>
</tr>
</tbody>
</table>

The `util` package provides a number of (mostly low-level) utility classes that are used by other classes or by applications.

**AppProperties**
This class wraps the `Properties` class in an application-specific way. It provides a single, global point of access to a range of data types in a component's loaded property file(s).

**ConsoleLogger**
A simple class providing a default logger for the `app.GlobalHooks` class that routes all output to the output window.

**DetailedError**
An extension of the built-in `Error` class which also records the component name generating an error.

**HostInterface**
A complex class that provides a uniform, buffered, synchronous data and command interface to a host (Director in the current implementation). It prevents a host error cascading into all subsequent function calls (although those to the same Director component are disrupted). It requires a corresponding FlashCommunication object in the host.

**IntDataObject**
A simple custom message class which allows application components to receive notifications from subclasses of the `Observable` class.

**NetInterface**
A class that provides buffered synchronous communication with the central controller via the `NetworkManager`. 
Observable
A simple implementation of the Subject component of the Observer design pattern (although implemented as a superclass rather than an interface). The class provides a simple database of observers and a notify method; state is left to subclasses.

Properties
A low-level class which simulates the java Properties class.

PropertyLoader
Loads individual properties files for the Properties class.

Timer
A very simple timer object which can used for both one-off and repeating tasks.

<table>
<thead>
<tr>
<th>message</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataEdit_Com_MessageBuilder</td>
</tr>
<tr>
<td>DataEdit_Rule_MessageBuilder</td>
</tr>
<tr>
<td>Dispatcher</td>
</tr>
<tr>
<td>Display_MessageProcessor</td>
</tr>
<tr>
<td>Error_MessageProcessor</td>
</tr>
<tr>
<td>MessageBuilder</td>
</tr>
<tr>
<td>MessageBuilderFactory</td>
</tr>
<tr>
<td>MessageProcessor</td>
</tr>
<tr>
<td>MessageProcessorFactory</td>
</tr>
<tr>
<td>PerfData_MessageBuilder</td>
</tr>
<tr>
<td>XmlMessage</td>
</tr>
</tbody>
</table>

The message package provides classes that allow a client to send sensor information to the central controller and to process information received. It contains a number of specialist subclasses which are likely to increase in number as functionality is implemented.

DataEdit_Com_MessageBuilder
A specialised subclass of the MessageBuilder class which builds dataEdit type messages carrying commands to the central controller.

DataEdit_Rule_MessageBuilder
A specialised subclass of the MessageBuilder class which builds dataEdit type messages carrying rules to the central controller.

Dispatcher
A class that takes received data and routes it to the appropriate output module based on information in the component.CapabilityDB (accessed through the single component.ComponentManager instance).

Display_MessageProcessor
A specialised subclass of the \textit{MessageProcessor} class which unpacks messages from the central controller that contain information to be sent to one of the available output modules by the \textit{Despatcher}.

\textit{Error\_MessageProcessor}
A specialised subclass of the \textit{MessageProcessor} class which unpacks error messages from the central controller. The message data is sent to the current error output module by the \textit{Despatcher}.

\textit{MessageBuilder}
A base class for packing data into the XML formats required for transmission to the central controller.

\textit{MessageBuilderFactory}
A class that returns the appropriate \textit{MessageBuilder} subclass for a particular message type.

\textit{MessageProcessor}
A base class for unpacking data from the XML formats used for transmission from the central controller.

\textit{MessageProcessorFactory}
A class that returns the appropriate \textit{MessageProcessor} subclass for a particular message type.

\textit{PerfData\_MessageBuilder}
A specialised subclass of the \textit{MessageBuilder} class which packs sensor data into the XML message format used for transmission to the central controller.

\textit{XmlMessage}
A lightweight custom data object used for sending unpacked messages between system components.

<table>
<thead>
<tr>
<th>\textit{project}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project</td>
</tr>
<tr>
<td>ProjectsPrefs</td>
</tr>
<tr>
<td>ProjectsDB</td>
</tr>
<tr>
<td>ProjectsDBServerProxy</td>
</tr>
</tbody>
</table>

The \textit{project} package provides classes for managing projects on the central controller.

\textit{Project}
This is a CDT representing an individual project.

\textit{ProjectPrefs}
The preferences for a specific project.

\textit{ProjectsDB}
A lightweight database which holds a local list of all available projects on the Central Controller.

*ProjectsDBServerProxy*

A class which manages communication between the projectsDB and the Central Controller.
Appendix 6
LIMPT Workshop Guide

This guide will help you get to know the LIMPT client – this is the piece of software that communicates between a performer (or other element of a performance) and the system.

The client collects and filters activity and tells the system when something significant has happened (this is known as an event). It also presents messages from the system to the performer, these are usually cues of one sort or another; a spoken command, a piece of text or an image.

The LIMPT client - overview

There are three parts of the client programme;

- Client Configuration – where you set up the input and output options.
- Prepare – where you add or edit the rules that will shape a performance.
- Perform – where the client collects various types of input from you while you are performing and passes on cues to you from the system.

You move between these 3 sections using the buttons at the top-left-hand side.
Getting started - configuring the client

1. Double click the *perfClient_dist.osx* icon

![Image of perfClient_dist.osx icon]

2. If you are using some input plug-ins (sound or MIDI), you may be warned that you are using a trial version – just click OK.

3. After the client has started up, if this is your first use, you'll see the configuration interface:

![Image of configuration interface]

4. Make sure *userManagement* is selected in the left-hand column, and then click on the + button at the bottom of the list of users on the right.
5. Enter your name (do not worry about a password) and click the **store** button. You will now be remembered by the client.

6. You may want to configure input or output capabilities – we will turn on the speech option as an example. Click on the **voiceOut** option in the left-hand menu (you can always come back and change any of these options later): -

7. Click on the check-box in the options panel to turn speech output on. To check the voice, click on the blue button. If you don’t hear anything, check that the check box has a blue cross in it and that your computer’s audio is turned up.
8. When you are ready, select the *network* item in the configuration panel:

![Network Settings](image)

9. This panel is where you select the server which will store and run your project. The chosen server will be highlighted in green. In most cases, you will choose *Bartleby* (this is the name of the LIMPT server at De Montfort university).

10. Click the *connect* button – you should see the connection indicator in the top-right hand corner turn green and the activity indicator flash red briefly. If the connection indicator does not turn green, check your computer's Internet connection.

11. A LIMPT server can store (and run) many performance projects, you can now choose which one you will work with; select the *project* item in the left-hand list.

12. You'll see a list of projects stored on the server you are connected to:
13. The project you are working with is highlighted in green. To start with, select *introduction_to_LIMPT*; you will see it highlighted. The client will remember which project you are working on, but you can change it at any time by returning to this configuration panel. *Introduction to LIMPT* is a special project which will help you learn about the LIMPT system and how to use it.

14. You have now finished the initial client set-up and are now ready to work with the LIMPT system. To start with, you will explore how the client works in a performance. Select the *Perform* button in the top left-hand corner:

15. Press the ‘1’ key on your computer keyboard.

You can set the client (see the clientSettings configuration panel) to automatically use your last settings, connect and then go straight to the *Perform* screen (this is the normal way to use it as a performer).

You will not have to go through the configuration process unless you want to change (e.g. work on another project). To return to the client configuration section, choose the *Client* button in the top, left-hand corner.
The LIMPT system and performance

The LIMPT system is based on a very simple model of how a live performance unfolds:

The passing of cues to performers in response to significant actions (events) is governed by rules. A Rule can simply be a way of saying that, if a certain performer does something, then send a cue to another performer.

So, a simple rule might be:

*(if) ian plays a C# 3 (then) say “start section three” to ruth *

A LIMPT project stores all the rules that govern the generation of cues to performers (or other performance elements such as computers, lighting desks, projection systems etc) for a performance.

These rules are written in a fairly simple language called eMerge.
Creating a LIMPT project

1. Click on the Client button in the top left-hand corner.

2. Select the network entry in the list on the left and check you are connected to Bartleby (the name of the server you are connected to is highlighted in green).

3. Select the project entry in the list on the left. You will now make a new project.

4. Click on the + button at the bottom of the project list, the new project panel appears:

   ![Project Details Panel](image)

   5. Give your new project a name that includes your user name (so you'll remember which it is) and then press the Store button. You should see your project name added to the list of projects. You do not need to put anything in the other text fields.

6. Click on your new project name to select it (it will be highlighted in green). This makes it the project you are now working on:

   ![Available Projects](image)

7. Click on the Prepare button in the top left-hand corner, the rule entry and editing screen will show (it will be blank as you have just made a new project):
Controlling performances
In the LIMPT system, the flow of a performance is shaped through a set of rules. These rules are written using the eMerge live performance scripting language. They have two parts; one (or more) events to look out for and one (or more) commands – things to do when the event(s) happens.

There is a detailed language reference which lists every word and gives examples of how to use them and explains the language’s grammar – for now, we will just use a few features to get started.

In rules, performers are referred to by name (e.g. ian, jane etc.). Performers can give signals (e.g. a “1” key signal, a mouse signal) or they can do things (e.g. plays “F# 2”, types “start the show”). In the LIMPT system a ‘performer’ may be a person, another computer programme or piece of equipment.

Cues can be given to performers in a number of ways; for now we will limit ourselves to showing things (e.g. show “Gradually speed up” to ian, show “score_page3.jpg” to jane) and saying things (e.g. say “start copying the performer to your right” to eric).

Adding a Rule to a Project
A project may contain many rules, but they are all entered in the same way in the Prepare screen.

For this example, we will write rules that provide cues when a button is pressed on the computer keyboard – but this is only so that you can test your rules easily; the events that trigger rules could also be MIDI signals, sound levels, elapsed time, distance or signals provided by sensors such as floor pads. The basic principle is, if you can work out a way to sense a quality, you will probably be able to use it as an event in the LIMPT system.

1. To add a rule, you click on the + button at the bottom of the rule-list.
2. The rule edit panel will open:

3. Click on the + button underneath the list of IF… events. Type your user name followed by *gives a “z” key signal* – here is my version: -

4. Click on the store button, you will see your text appear in the list of IF… events.

5. Click on the + button underneath the THEN… list, the entry window opens again

6. Type *show “start circling the stage” to EVERYONE* and then click the Store button. Your rule entry window should look something like this: -
7. Click on the store button; your rule is sent to the LIMPT server and stored there. You should see your rule appear in the list of rules and a message from the server saying **INFO - Rule Stored.**

8. You can now test your rule. Click on the *Perform* button in the top left-hand corner; you will see the perform screen. Press the ‘z’ key – you should get a message from the server telling you to circle the stage!
**Editing a Rule**

Often you will want to change rules; the process is very similar to entering a rule. We will change the event that triggers our rule and make it give a spoken as well as visual cue to performers.

1. Click on the *Prepare* button in the top left-hand corner. The rule(s) will be reloaded from the server.

2. Select the rule you wish to edit in the rule list, it will be highlighted in green:

![Project Rule List](image1.png)

3. Click on the *edit* button at the bottom of the list:

![Rule Entry Panel](image2.png)

4. The rule entry panel opens with the rule you wish to edit. We will first change the event the rule is triggered by. Select the event by clicking on it, it will be highlighted in green:

![Event Listing](image3.png)

5. Click on the *edit* button just underneath the event listing (see above).

6. The text entry panel will open, edit the text so that it is your user name followed by *gives a “x” key signal*. Click the *Store* button. You will see the change in the Rule panel.
7. We will now add another item to the commands. Click the + button underneath the **THEN...** listing (the third list in the rule panel).

8. In the text entry box, type *say “move on to the next section” to EVERYONE*. Click the store button. You should now have 2 items in the command list; one give a visual cue, the other a spoken cue:

```
THEN... (Command list)
show "start circling the stage" to EVERYONE
say "move on to the next section" to EVERYONE
```

9. Click on the **Store** button at the bottom of the rule panel. You should return to the rule list and get a message from the server confirming the update: **info Rule updates stored**.

10. Click on the **Perform** button in the top left-hand corner.

11. Press the ‘x’ key and you should both see and hear a cue.
Appendix 7 Introduction to LIMPT (Online Materials)

The following images are part of the online introduction (there is also a text to speech commentary on each slide) to the system used in workshops, but also permanently available on the Introduction to LIMPT project (press ‘1’ to start).

- **Live**
- **Interactive**
- **Multimedia**
- **Performance**
- **Toolkit**

Press ‘2’ to go on...

Screen 1
Performance elements

• In the LIMPT system there are 2 kinds of elements:
• Physical elements - performers
  – These are the actual things that contribute to a performance through their activity.
  – They include the people involved, but also hardware such as lighting desks or other computer systems.
• Conceptual elements
  – These are things like physical space (zones) or divisions of time (sections) which can also play a part in deciding how a performance should unfold.

Press ‘3’ to go on...
Performers

• Performers can give signals,
  – these may be a key or mouse-button
  press on a computer or controller
  – or a MIDI signal from a controller or sequencer
  – or another sort of activity – e.g sound level.

• Or they do things
  – they can type or say something
  – or play a MIDI note or phrase.

Press ‘4’ to go on...

Screen 3
**Time and Space**

- The LIMPT system has elements called *sections* which can represent parts of the performance.
  - They can be *started* and *stopped* and
  - they can measure *time* passing.

- Physical spaces or locations are called *zones*.
  - they can be *empty* or *occupied*
  - and they have a *3D position* which can be tracked.

---

Press ‘5’ to go on...
Events = important activity

- The LIMPT system senses all the different sorts of activity in a performance.
- It constantly looks out for certain, significant things to happen; these are called events.
- When an event happens, the system responds by sending messages or cues to some or all of the performers (people or devices).
- You specify events and responses by writing rules.

Press ‘6’ to go on...

Screen 5
Cues to Performers

• The LIMPT system can send signals or cues of many different types.
• Cues can be visual:
  – Images, Text.
• Or audio:
  – Spoken text, MIDI notes, sounds.
• Or digital data:
  – MIDI signals.
Controlling Performance

• The LIMPT system stores the events you want it to watch for and the responses you want it to make as a series of rules.

• Each rule has two parts:
  – An event (or several events) which trigger it and
  – A command (or several commands) which describe the response.

Press ‘8’ to go on...

Screen 7
Writing Rules

• Rules are written before the performance by those devising the performance.
• Rules are written using a simple language developed for the LIMPT system.
• A simple rule might be: -

<table>
<thead>
<tr>
<th>If... (event)</th>
<th>ian gives a “z” key signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Then (command)</td>
<td>say “start the last section” to EVERYONE</td>
</tr>
</tbody>
</table>

Press ‘9’ to go on...

Screen 8
More Information

• To get started using the LIMPT system, see the guide which is included with the LIMPT client programme.

• The latest version of the detailed language guide and client software are available from the project website: -

  http://www.willcock.org/works/projects/limpt.html
Appendix 8
DRHA 10 Workshop Participant Survey

**LIMPT Workshop**
Jane Turner and Ian Willcock
DRHA10 – Tuesday 7th September 2010

I hope you’ve enjoyed the workshop and also that it may have sparked new ideas for your own creative or critical practice.

I’d be very grateful if you could provide some feedback about yourself and your initial response to the system. This will be used as a part of my PhD research into the use of interactive multimedia within different performance traditions.

Any information you provide will be kept confidential and you will not be identified in any way in accounts of my research.

Thank you

Ian

<table>
<thead>
<tr>
<th>Question</th>
<th>Not at all</th>
<th>Only to a limited extent</th>
<th>Yes, partly</th>
<th>Yes, easily</th>
<th>Not sure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do you feel the model of live performance (i.e. a series of decision points) behind the LIMPT system could integrate with the ways you think about, and work with, live performance?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Do you find the ways things to look out for (events) and things to do (responses or cues) are written is easy to understand (don’t worry about not knowing all the words)?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Has today’s workshop suggested things to you which you might want to develop further in your creative or critical practice (either with or without the LIMPT system)?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Would you be interested in exploring the LIMPT system further?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Has today’s workshop given you enough information for you to feel confident about exploring the LIMPT system further?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Which performance traditions or creative genres does your practice relate to (creatively or critically)?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Are there any features or improvements you would like to see in the LIMPT system?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>