An Agent Based Approach to Interaction and Composition

Stephen Pearse
University of Sheffield
spearse198@gmail.com

David Moore
University of Sheffield
d.moore@shef.ac.uk

ABSTRACT

The Agent Tool [1] is a complex composition and performance environment that affords the scripting of abstract agents of varying complexity to control elements of synthesis and sound manipulation. The micro-threaded architecture of the system means that all scripts are written from the perspective of an agent/performer. Consequently, complex compositions, gestures and performances can be crafted in a simple and efficient manner. It is designed to be an open-ended framework whereby all data can be emitted via Open Sound Control (OSC) for external processing if required. User interaction with the system can come in a variety of forms. These include, but are not limited to graphical manipulation, scripting, real time video input and external control via OSC. The system was initially designed as an environment to allow dynamic and efficient graphic sound synthesis through extraction of data from static imagery, video playback or real time video input. The open scripting engine consequently allows the system to perform direct *audification* of image stimuli or conversely allow complex *sonifications* to take place. The Agent Tool is a cross-platform package that runs on various Linux distributions, Mac OSX and Windows operating systems. This paper seeks to discuss the agent based functionality the system offers and consequently the composition and interaction design that the system affords.

1. HISTORICAL CONTEXT

The Agent Tool in its first incarnation was designed to be a flexible and intuitive means of performing image sonification in a similar manner to systems such as UPIC and MetaSynth. Iannis Xenakis' UPIC system [2, 3] and the IanniX system [4] were designed with the belief that minimal direct artistic or technical constraints should be imposed on the composer. In Xenakis' words "the UPIC is a tool that enables one to work in a systematic way on various levels at the same time" [5, 6]. Tools such as MetaSynth, the ANS Synthesizer and HighC used fixed axis mappings such as pitch and time. However, the UPIC and IanniX systems are far more flexible in affording the abstract mapping of graphical data to arbitrary compositional parameters. In the UPIC system, the *arc* feature allowed abstract mapping of data to synthesis parameters [2, 3, 7].

The Agent Tool was born as a means of creating a flexible composition and performance interface based upon these ideas.

2. TECHNICAL OUTLINE

In the Agent Tool we can assign abstract entities, known as agents, to control performance parameters. Within the system an agent's behaviour can be scripted to control synthesis, compositional and/or performance logic to varying degrees of complexity. Consequently the agent Tool offers a form of object oriented composition and interaction. The scripting engine allows users to program dynamic mappings of abstract data to audio and/or control parameters. Agents communicate bi-directionally with external tools and software via OSC. The system is compatible with SuperCollider's OSC interface and features a suite of tools which can dynamically spawn and manipulate nodes on a remote SuperCollider server.

The Agent Tool's AI system, utilizes a form of co-operative multitasking rather than the commonly used Finite-State Machine (FSM). This approach then allows the scripted behaviours to be much simpler. Coroutines are used to allow each AI to operate in its own pseudo-thread without the resource cost of system threads [8]. An agent's behaviour executes sequentially and it's coroutine yields to allow others to execute.

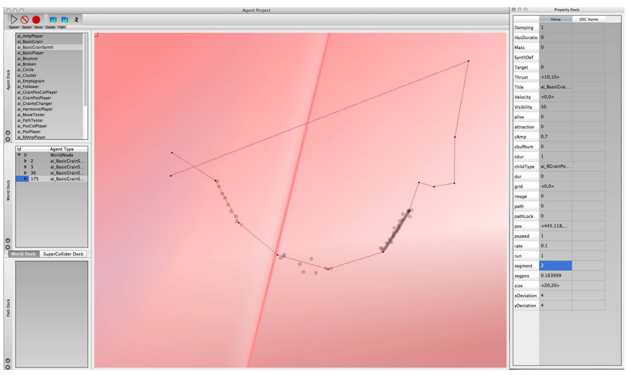
2.1 System Architecture

Agent Tool was developed using a variety of Open Source libraries and tools.

- Application developed in C++ and Qt.
- Lua provides embedded scripting.
- LibLo allows OSC communication.
- OpenCv is used for image processing.

3. THE AGENT

As previously stated, agents within the system represent script-able compositional entities. Each script exists within its own coroutine with its own lifetime and concept of time. This approach means that scripting is simple and efficient whereby the user programs behaviour from the perspective of the agent itself. For example, scripting motion becomes as simple as calling a 'move' function or setting a velocity property. A sequence of actions can be described as a list of steps to follow, rather than the using an update function



Id	Agent Type
0	WorldNode
2	ai_BasicGrainS...
3	ai_BasicGrainS...
36	ai_BasicGrainS...
175	ai_BasicGrainS...

	Value	OSC Name
Damping	1	
HasDuratio	0	
Mass	0	
SynthDef		
Target	0	
Thrust	<10,10>	
Title	ai_BasicGrai...	
Velocity	<0,0>	
Visibility	50	
alive	0	
attraction	0	
cAmp	0.7	
cbufNum	0	
cdur	1	
childType	ai_RGrainPo...	
dur	0	
grid	<0,0>	
image	0	
path	0	
pathLock	0	
pos	<445.118,...	
pspeed	1	
rate	0.1	
run	1	
segment	2	
segpos	0.163999	
size	<20,20>	
xDeviation	4	
yDeviation	4	

Figure 1. Agent Tool 's Graphical User Interface

that would be necessary in a FSM implementation . Such an approach means that agents can represent sequences of varying lengths, from entire compositions to finite musical gestures. New agents can be instantiated from within the Graphical User Interface (GUI) in real time.

In the Agent Tool Version 1.0, agents exist within a two-dimensional canvas which they can traverse as the user sees fit. Any agent within the system has a variety of *properties* which can be manipulated, extracted or set in real time. Properties can exist to store a variety of data types, floating point or integer values, boolean states, strings of text or two dimensional vectors. When an agent is created it is given a set of properties by default . These properties include positioning and velocity vectors, lifetime/duration, mass, thrust and a string representing the name of a SuperCollider synthesizer to spawn (known as a SynthDef). Upon setting the 'SynthDef' property, either in the GUI or from within the scripting engine, all control properties will be bundled into a OSC message spawning a SuperCollider node of the appropriate type. This link to synthesis is optional. Agents can exist purely as control structures. The system imposes no direct mapping strategies or constraints on the user. Mappings can be dynamically changed over time, for example if a set of conditions are met. Furthermore, the user can directly choose the mapping of properties within the GUI.

3.1 Scripting API

Vaggione [9] and Dahan [10] both argued that time within a system should not act as a shared constant, instead, tools should allow the traversal of various time scales at once as a reflection of how the composer works [11]. The Agent Tool echoes this sentiment ensuring that all events occur on individual and relative time-scales to each other.

Each agent exists within its own coroutine. A direct result of this is that each agent has its own concept of time. Consequently it is possible to construct definitive compositional events and gestures by spawning agents . Sequences of events are created through repeated spawning or by agents that themselves spawn child agents . Furthermore, this approach affords the creation of relative sequences of events.

At any given time a user can load new scripted behaviours into the system. Upon loading a script, the system monitors the state of that file so that if the script is altered by another software package, it is reloaded instantaneously. To create an agent the user has to fulfil two requirements. AI behaviours are defined using a single Lua function with a special prefix ('ai_'). These behaviour functions are then made available within the GUI for the user to spawn as agents . In effect these functions represent the 'brain' of the agent .

Listing 1 illustrates how a user can create a simple agent that utilizes its horizontal and vertical positioning to control stereo pan positioning and pitch respectively. Behaviour functions such as 'setp()','getp()' and 'addp()' allow an agent to modify their own state or store data.

```
function ai_HelloWorld()
        initai()
        local position = getp("pos")
        addp("freq",position:y() )
        local worldsize = getworldsize
            ()
        addp("pan", ((position:x /
            worldsize) * 2) -1    )
        addp("amp",0.8)
        setp("SynthDef","default")
        pause(1)
        die()
end
```

Listing 1. A complex "Hello World" agent

The software provides a rich and varied Lua interface allowing the crafting of a variety of agent behaviours. The API affords control over: agent relationships; agent spawning; image retrieval/writing; the mapping of controls and/or synthesis components and the amount of user control within the GUI.

3.1.1 Relationships and Interactions

Any agent within the system can have a near infinite number of child agents . At any time an agent has the ability to spawn other agents of any given type. These new agents can exist as children of the parent agent , or can be assigned to exist within the active 'world'. All agents consequently have the ability to manipulate all of their child agents by setting the child's properties. It is possible for agents to monitor the properties of all child agents . In practice this can be used to monitor the amplitude, spatial positioning and harmonic content of child agents . A parent agent can directly manipulate whole collections of agents to achieve consonance or dissonance. across the properties of collections of agents .

Whilst agents do not implicitly create audio, it is possible to utilize the hierarchical relationship within the framework to create 'control' agents which may indirectly create sonic events. One could consider these control agents as defined sonic objects or sequences.. Through this technique it is possible to realize highly complex forms of granular synthesis whereby the user interacts primarily with a agent whose job it is to simply spawn hundreds of child 'grains'. It is entirely up to the user to decide what these child grains should be, whether they should generate sonic material, or whether they themselves act control further agents . This affords recursive compositional ideas to be realised.

3.1.2 Image Audification and Sonification

The Agent Tool allows users to use static imagery, video files and live webcams as image stimuli within the system. The open nature of the system allows direct *audification* akin with systems such as MetaSynth and the ANS Synthesizer, as well as abstract *sonifications* such as those found in UPIC, IanniX and to some extent the 'Music Sketcher' [2, 3, 12, 13]. Whilst these systems are limited to static imagery, the Agent Tool allows real time stimuli and the ability for scores can be 'scanned' in via the usage of an attached web-cam.

As previously discussed, the Agent Tool does not impose a set mapping strategy. Consequently, the way in which an agent reacts to image stimuli can be scripted. For example, colour data can be read as a means of dictating the pitch and amplitude of an oscillator (akin with the *audification* approach of MetaSynth). However, unlike MetaSynth, a user of the Agent Tool can freely dictate the part of the image used to control a synthesis voice. Each voice can have completely independent synthesis algorithm. Similarly, these voices can be replaced with filters or delay units to which another sound source/performer may be rooted through. In these instances, the colour intensity at the location of the agent can be used to dictate the amplitude or wet/dry mix of an effect.

On the other hand imagery can be used purely as control data in the creation of a higher level system. For example, it is possible to use colour data to dictate whether an agent should continue to exist. The 'checkLife()' function (see listing 2) that comes with the system can be invoked at any time from a behaviour script, and kills an agent if the average colour component at its position drops below a threshold.

```
function checkLife(threshold)
        if getmeancol(0) <= threshold
            then
                die()
                return true
end
```

Listing 2. The 'checkLife' function.

Conversely, it is possible for the image data to dictate the spawning of new agents . The 'spawnLife() ' function (see Listing 3) can be used to spawn agents of the users choosing when the average colour component exceeds a threshold. An advanced form of this function allows the spawning of agents when specific colours are found within an image.

```
function spawnLife(threshold ,parentId ,
    agentType)
        if getmeancol(0) >= threshold
            then
                spawnrelative(parentId
                    ,1 ,agentType ,0 ,0)
                return true
end
```

Listing 3. The 'spawnLife' function.

As well as extracting colour data directly from directly under the agent , it is also possible to read the image from any point that the user dictates. This feature allows for far more complex image analysis. In another example agents are programmed to be attracted to, or repulsed by certain image components. The system comes with a function that returns a vector to the closest position within a circular radius where a colour components exceeds a threshold. If the returned vector is used to influence an agent's velocity, agents will be attracted to or repelled from specific colours. In combination with a web-cam it is consequently possible to use real word items and environments to enforce motions and interactions with agents .

It is also possible for agents to draw colour data onto an image surface within the system. This affords agents a means of communicating via the image. One agent can deposit information in the image, whilst another can read and react to the changing image. Image functions allow the composer to create self generating sonic landscapes. They can interact both through the Agent interface and the live image interface. Another example, is the possibility of graphically 'scoring' image stimuli on paper and scanning into the system to create repeatable simulations and performances.

3.1.3 Tracking

Through the API one agent can track another agent . Agents can perform queries such as 'Is agent within a circular radius?' or 'Find the closest agent?'. Consequently, it is possible to create agents that are attracted to or repelled by others. The software also features an example behaviour that implements Reynold's famous 'Boids' flocking algorithm [14] whereby an agent 's position and velocity is dictated by rules of *separation*,*alignment* and *cohesion*. In combination with real-time user input (by user interface, image stimuli or external controller) the user can interact with this simulation in real-time. Furthermore, the user has the ability to adapt the provided agent function, mapping agent properties to sonic properties as they see fit.

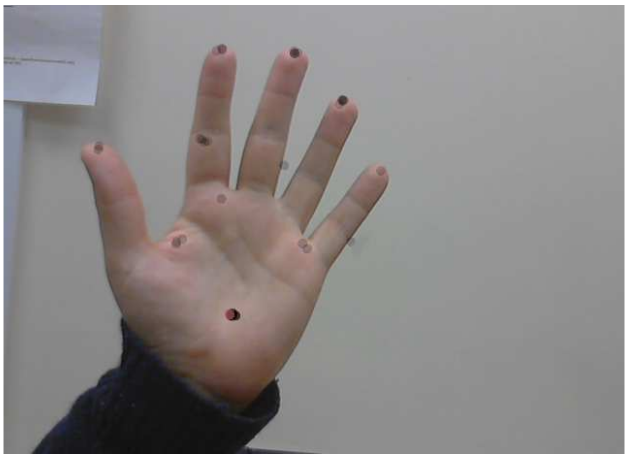


Figure 2. Agent Tool using a web-camera to perform a basic form of hand tracking

3.1.4 External Controller Support

Every agent property can be set via OSC. This enables external control from a wide range of software and hardware controllers. Future iterations of the system will allow direct input, and consequently script-able interaction with both Leap Motion and Kinect controllers.

4. AGENT BASED INTERACTIONS

Agent Tool allows users to design (and consequently compose) materials that are entirely relative to one another. Rather than instantiating or controlling items, synthesis or manipulation at a global level, it is possible to sequence events and relationships that are relative to one another. Consequently, this allows for a much more dynamic and 'reactionary' approach to composition and performance in allowing the user to vary when agents are instantiated in performance, be it in front of an audience or within the studio. The system enables the crafting of materials and interactions based purely upon defined relationships and logic which can be realised in a non-linear fashion.

As agents are abstract entities and can exist merely as controllers or objects that directly control synthesis, the amount of interaction that the user has can vary depending on the given context. Similarly, as there are no implicit mappings within the framework, OSC can be emitted in a format that the user so wishes from within the environment. Furthermore, these mappings can vary over time and can potentially react to elements within the environment. It is therefore possible to create hybrid forms of instruments that are both *reactive* and *generative*. Crucially, the multi-tasking framework the system is built upon allows interactions to be crafted and executed efficiently.

5. MUSICAL CONTEXTS

Two vastly different acousmatic works have been composed by the author using Agent Tool , *Liten Röst* (2014) and *Mikro Studie A+B* (2014). Each of these works represents a different set of compositional methodologies and implementations. *Liten Röst* on the one hand is a *texture carried* work exploring the purity of the female voice. *Mikro Studie A+B* on the other hand is a study in flocking microsound and is completely *gesture carried* [15]. The usage of Agent Tool varied substantially across these two works, both in terms of agent and interaction. In both instances however, agents were used to spatialize materials of several configurations of speakers.

In the context of *Liten Röst* Agent Tool was used to design a collection of instruments utilizing different forms of granular synthesis that should then be explored through real time interactions with the system. The author subsequently interacted with a collection of control agents that each in turn spawned and subsequently controlled a variety of synthesis parameters of child grains. A suite of different child grains were designed, each using different forms of logic to control their behaviour. For example, one of the control agents dictated different flocking attributes of its children whereas another controlled what attributes of real time video could be analyzed and synthesized.

The scripting of spatialization within these works became highly intuitive via the parent child relationships of the system. In both contexts control agents were used to represent single speakers and the distance of child (or non-child) agents from these controls would represent the amplitude of that source in that speaker. Consequently it became easy to reposition and script the motion of these speakers across time.

As a composer, utilizing hundreds of different agent instances, each with their own unique parameter mappings within the same framework, offered a highly efficient workflow yielding very complex sonic results. With all of the agents within the system reacting to one another, the instruments became highly 'learnable through usage. Once an agent or set of agents were learnt through practice, it then became possible to score sections of the work within the scripting API.

The scripting of complex sonic relationships and passages consequently became the focus of the work *Mikro Studie A+B* with real time interaction being strictly prohibited. The flexibility of the system was not without its drawbacks however. Often being confronted with endless possibilities, the scripting engine itself somewhat limited compositional activity. It was only through the design of real time interactions that the system became a highly efficient means of sonic exploration and scoring. Akin with designing instruments in other audio programming environments such as Pure Data and MaxMsp, each new agent that is crafted requires interaction and exploration for its behaviours to be learnt by a user. When these agents are put in a context whereby thousands exist at any one given time, the slightest change to the compositional or control logic can have huge implications upon the materials produced.

6. CONCLUSIONS

The form of co-operative multitasking that the system utilizes allows relationships, mappings and interactions to be crafted from the perspective of agent within the system. The scripting framework consequently allows these systems to adapt and react to user input in a variety of forms. Furthermore, the reactions to user input can be scripted to evolve over time to suite the needs of a compose or performer. From simple scripting users can instantiate highly complex sonic materials and interfaces to suite a variety of contexts. Future development on the system will be focused on direct integration of a suite of commercial motion controllers as a means of enabling greater flexibility for performers.

7. REFERENCES

[1] S. Pearse, "The agent tool," http://www.theagenttool.co.uk, accessed: 2014-07-14.

[2] C. Roads, *The Computer Music Tutorial*. Cambridge: MIT Press, 1994.

[3] G. Marino, M.-H. Serra, and J. M. Raczinski, "The upic system: Origins and innovations," *Perspectives of New Music*, vol. Vol. 31, No.1, pp. 258–269, 1993.

[4] Coduys and Ferry, "Iannix: Aesthetical/symbollic visualisations for hypermedia composition," 2009.

[5] H. Lohner, *The UPIC system: a user's report*. Computer Music Journal, 1986, vol. 2, no. 10, pp. 42–49.

[6] ——, *Interview with Iannis Xenakis*. Computer Music Journal, 1986, vol. 10, no. 4.

[7] P. Manning, *Electronic and Computer Music*. Oxford Oxfordshire: Oxford University Press, 2004.

[8] B. Dawson, "Micro-threads for game object ai," *Game Programming Gems*, vol. 2, 2001.

[9] H. Vaggione, "Some ontological remarks about music composition processes," *Computer Music Journal*, vol. 25, no. 1, pp. 54–61, 2001.

[10] K. Dahan, "Domaines formels et representations dans la composition et l'analyse des musiques electroacoustiques," Ph.D. dissertation, CICM, Universit e Paris VIII, France, 2005.

[11] J. Thiebaut, P. Healley, and N. Kinns, "Drawing electroacoustic music," in *International Computer Music Conference*. International Computer Music Association, 2008, pp. 261–264.

[12] J. Thiebaut, "Sketching music: Representation and composition," Ph.D. dissertation, Queen Mary, University of London, 2010.

[13] I. Xenakis, *Formalized Music: Thought and Mathematics in Composition*. Bloomington: Indiana University Press, 1971.

[14] C. Reynolds, "Flocks, herds, and schools: A distributed behavioral model, in computer graphics," *SIGGRAPH '87 Conference Proceedings*, vol. 21, no. 4, pp. 25 – 34, 1987.

[15] D. Smalley, "Spectromorphology: explaining soundshapes," *Organised sound*, vol. 2, no. 2, pp. 107–126, 1997.