An Experimental System for Real-time Interaction Between Humans and Hybrid AI Agents

John Pendlebury, Mark Humphrys, Ray Walshe
School of Computing
Dublin City University, Glasnevin
Dublin 9, Republic of Ireland
Email: jpendlebury@computing.dcu.ie

***Abstract*—There is an emerging belief among AI researchers that intelligence is the product of specialised subsystems collaborating in a type of network of the mind. Several prototype systems have been developed as part of the World-Wide-Mind project [6] that facilitate the on-line deployment of hierarchically structured, multi-author, AI agents that we call minds. These agents function in on-line user defined environments that we call worlds. In this paper we describe the latest prototype system developed as part of the World-Wide-Mind project, the XAI (Experimental Artificial Intelligence) Server. Unlike its predecessors the XAI Server is a multi-agent system (MAS) allowing minds to run concurrently in the same on-line environment, facilitating inter-mind communication, cooperation and competition.**

The XAI Server also adds real-time interaction between humans and AI agents. Along with the potential for gauging the effectiveness of AI algorithms this raises issues relating to timing and knowledge representation that previous prototypes had no need to address. The XAI Server is loosely coupled with a scheduled threading model that makes it far more scalable than its predecessors. Capable of communicating with a standard Web browser the XAI Server moves the responsibility for content generation from the server to the client via a proprietary user interface language. We demonstrate how this system works with a sample ChatWorld and four example minds. Finally we discuss our future plans for enhancing the system to facilitate 2D and 3D world authoring.

***Keywords*—Hybrid AI;Society of Mind;Multi-agent System;Human AI Interaction**

I. Introduction

Researchers in the fields of AI and cognitive science broadly agree that 'intelligence' is most likely to be the result of numerous highly specialised subsystems collaborating in what has been described [1] as a "Society of Mind".

The software paradigms necessary to create an artificial "Society of Mind" have existed for some time within the sub-field of multi-agent systems. Many tools for constructing multi-agent systems have been built [2] [3] [4]. The concept of "Society" and the relationship between individuals and the whole is modelled quite effectively within this archetype and yet the ultimate goal, that of realising intelligence, remains elusive.

We propose [5] that if Artificial Intelligence (AI) is to "scale up", it will require the collaboration of researchers from many diverse disciplines across multiple laboratories. No system, or set of tools currently exists to easily provide such a facility. [5] describes a scenario where an algorithm is represented by a software component which we call a 'mind'. The problem an algorithm addresses is represented by a second software component that we call a 'world'. Both worlds and minds are deployed as servers on the Internet. An attempt to solve a world, using a mind, can be made by a remote 'client' process.

In this 'action selection' scheme, as shown in Fig. 1, the client retrieves the 'state' of a world and sends it to the mind. In response to the state, the mind produces an 'action'. For example, in a world representing a two-dimensional maze, the state may simply consist of the Cartesian coordinates representing the mind's current position. An action could consist of the direction in which the mind wishes to move.

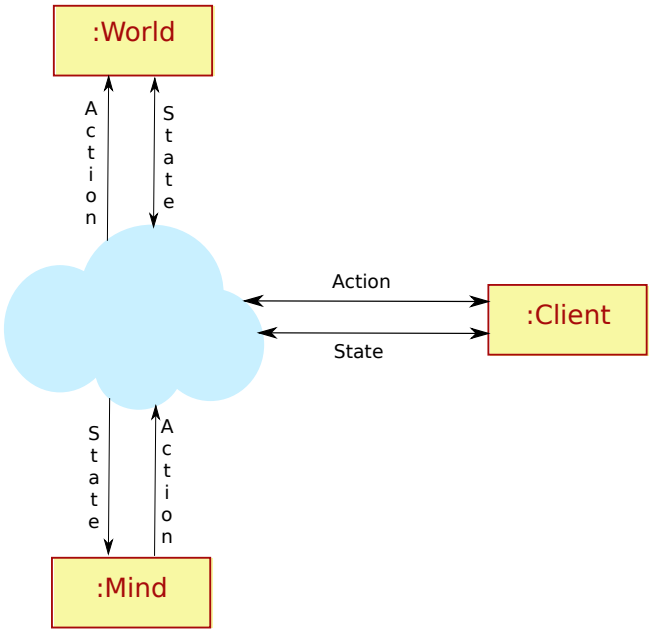

Fig. 1. Information flow between a mind and a world controlled by a client.

The previously retrieved state and the newly retrieved action are then sent back to the world, which produces a new state in response. The world is effectively being asked to produce a new state given the current state and an action. This process

This paper is supported by the Irish Research Council for Science Engineering and Technology

continues until the world indicates that it has reached a 'final state' i.e. the problem has been solved, or the time allocated by the world to solve the problem has run out.

Third party researchers can reuse these minds as components in their own larger minds, without necessarily consulting the original mind authors. These composite minds are constructed as hierarchies, with one mind at the top of the hierarchy arbitrating between the actions of minds below it in the hierarchy. To the client the hierarchical mind appears to be a normal mind, receiving states and issuing actions. Internally, when a hierarchical mind receives a state, it issues that state to each of the minds directly below it.

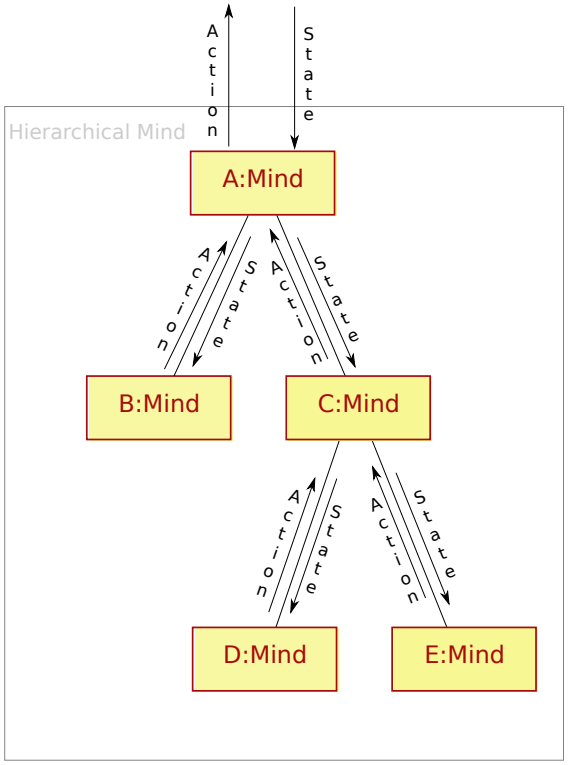

Fig. 2. A mind hierarchy consisting of five minds with potentially five unique authors. Only mind 'A' can return an action to the world via the client. It chooses between the actions produced by minds 'B' and 'C'. Mind 'C' must choose between the actions of minds 'D' and 'E', but mind 'B' produces its own action.

This concept is demonstrated [6] in a system that allows a mind and world to be deployed and accessed on-line, and subsequently in a system [7] that shows how a hierarchical mind can be created on-line by allowing one central mind to arbitrate between the actions of minds below it in a hierarchy. A later system [8] demonstrated a hybrid mind consisting of hundreds of sub-minds from hundreds of different authors.

O'Connor et al (2002) notes the speed limitations inherent in distributing minds from worlds over the Internet. Mac Fhearaí et al (2011) overcome these delays by allowing a world and mind to be executed together on a centralised server known as the World-Wide-Mind, see Fig. 3. In this architecture the functionality of the client, described in Fig. 1, is moved to the server. All user interaction is via a standard web browser. The term 'run' is used to describe the execution of a mind with a world. A user initiates a run through a Web interface provided by the World-Wide-Mind. After the run is complete, which may take anywhere from a few seconds to several minutes, the results are displayed in the browser. These results consist of the states, produced by the world, and the actions, produced by the mind, during the run.

At any point in the run, a world may output images. These can be used to represent state. This sequence of images is presented in the browser, with the results, when the run terminates. The Web interface to the system allows the user to browse the images in conjunction with the results. The World-Wide-Mind also provides the facility to generate and download a movie from any generated images.

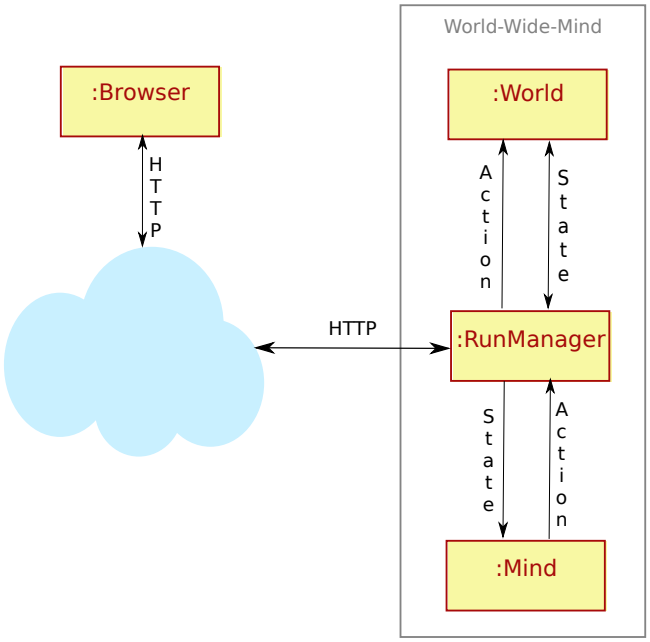

Fig. 3. The World-Wide-Mind architecture has the advantage over earlier systems that no proprietary client is required. All interaction with the system is through a standard browser.

In this paper we describe a new system, the XAI Server (See Fig. 4), which has several advantages over the systems [3][4][5][6] described earlier:

1) All of the systems described up until now, including the World-Wide-Mind have been 'single-agent' systems. That is, only one mind (agent) is run with a world at any one time. Even with hierarchical minds, consisting of many sub-minds, only the mind at the top of the hierarchy may issue an action to the world. The XAI Server allows multiple minds to run in direct competition with each other i.e. a multi-agent system. These minds may be hierarchical in nature, or be composed of a single mind. This introduces the possibility of inter-mind communication and cooperation as opposed to the intra-mind communication of the hierarchical minds described in Fig. 2. Such interaction was impossible in the systems

[3][4][5][6] described earlier.

2) The XAI Server features the addition of direct, real-time human interaction with worlds and minds. Users are included in a run as 'proxy-minds', while their actions are captured from a standard browser, which also presents the state of the world to connected users in real-time. This raises questions in terms of knowledge representation and timing. For example, how can the state of a generic world be represented to a user? Also, how can user actions be mapped to actions that a generic world and mind can interpret? Timing is an important issue for the XAI Server. The World-Wide-Mind runs a single mind with a world and delivers the results as quickly as possible. Taking this approach with the XAI Server, the user would be unable to perceive changes in the state of the world, making real-time human interaction impossible. Instead, the XAI Server allows worlds to dictate their timing requirements i.e. how often minds and users are permitted to send actions to the world.
3) The World-Wide-Mind allows worlds to generate images on the server and output them to the client browser. This allows users to perceive the changing states of the world as actions are consumed. To reduce network traffic the XAI Server does not allow the transfer of large data packages, such as images. Also, the server-side generation of graphics raises concerns about CPU bandwidth and memory capacity. Instead, the XAI Server allows worlds to describe their user interfaces textually using a specially designed language, WUIML (World User Interface Mark-up Language). The description of the user interface is transmitted to the browser before the run begins. Then only changes to the user interface are transmitted from the server to the browser during a run. User interface events generated by the user are captured on the browser, sent back to the XAI Server and relayed to the world.

Throughout the remainder of this paper the architecture and operation of the XAI server are described. A demonstration World, ChatWorld, and several demonstration minds for ChatWorld: ChatMind, GossipMind, JabberMind and HybridChatMind, are also described.

II. XAI Server

This section outlines the key features of the XAI Server and then goes on to describe how these features are implemented.

A. Key Features

The main features of the XAI Server can be described as follows:

1) It is possible for users to access the XAI Server from a standard browser, without the need for third party software installation. This takes advantage of the ubiquitous nature of Web browsers and maximises the system's accessibility.

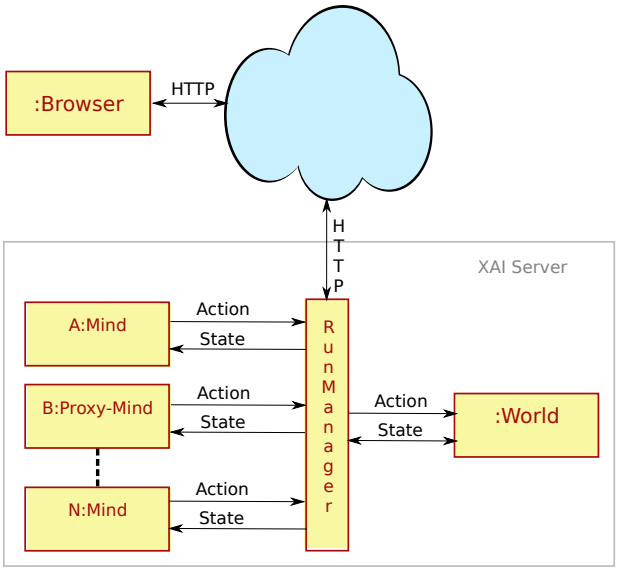

Fig. 4. Information flow between a world, several minds on the XAI Server and the client browser. Note that one of the minds involved is a 'Proxy-Mind'.

2) The XAI Server is a multi-agent system allowing multiple independent agent minds to run in competition with each other. Inter-mind communication, outside the hierarchical structure described in Fig. 2, is possible. The XAI Server does not attempt to impose a structure for inter-mind communication, but rather enables it.
3) Direct user interaction with minds is possible through the XAI Server. In fact, this is a central concept in the XAI Server architecture. One user initiates a "run" by choosing a group of minds designed for a specific world. That user then has their actions (keyboard and mouse events) mapped to actions defined by the world in question.
4) More than one user may interact with a group of minds engaged in a run. As stated above one user initiates a run. Once a run is under way its existence is visible to all users. A user may then choose to "join" a run. The run will continue for as long as at least one user remains connected to that run.

B. Architecture

In the introduction to this paper we described minds as algorithms. In this research, minds are software components which produce actions in response to states. The structure of these actions and states are defined by a world. Thus each mind is written for a specific world, as it must understand the structure of the states it receives and the actions it produces in order to interact meaningfully with the world.

1) The Action Selection Loop: A world in the context of the XAI Server is simply a container of state. A world changes this state when it receives an action from a mind. The system implemented by Mac Fhearaí et al (2011) is a single-agent system. Therefore a world knows which mind it is receiving

an action from, as only one mind is capable of producing an action. In the XAI Server several minds can interact with a world simultaneously. As there are multiple sources, actions must be qualified with an identifier for the mind that produced the action.

Central to the functionality of the XAI Server (and indeed the World-Wide-Mind) is the concept of a run. The term run refers to the process of running a mind (or minds) with a world, from the worlds initial state to some terminal state. Exactly when this terminal state is reached is determined by the world.

An overview of a run is described in Fig. 5 The RunManager begins by retrieving the state from the world and delivers it to the first mind in the sequence. This mind may produce an action, or choose to do nothing. The action, if produced, is then delivered to the world, but before a new state is retrieved the world is queried as to whether it has reached its final state. If the final state has not been reached the RunManager then delivers that state to the next mind in the sequence. Having updated all minds the RunManager sleeps before continuing the process all over again.

The duration for which the RunManager sleeps is a function of the world. Each world must implement the World interface. This interface defines several functions, one of which is the getUpdatePeriod() function. This function returns the time in milliseconds that the RunManager must sleep for.

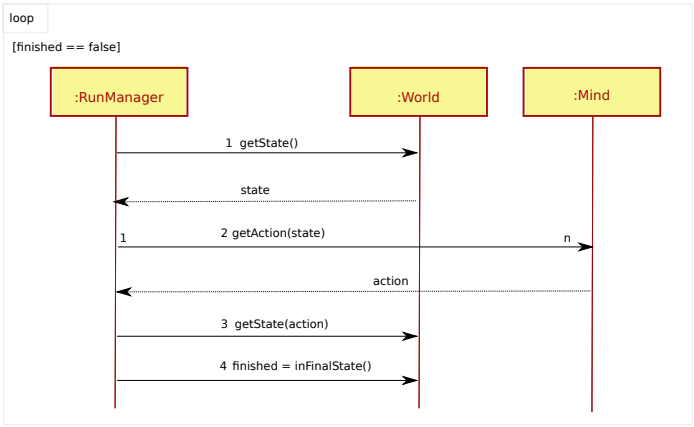

Fig. 5. A run orchestrated by the RunManager.

2) Threading Model: Roth (2007) explains that the naive approach to server design i.e. utilising one dedicated thread per connection, is not optimal. Under this scheme threads spend a significant amount of their life waiting. He notes that a one-to-one relationship between the number of worker threads and the number of client connections limits scalability.

This situation is analogous to our own environment. In the previous section we noted that the RunManager sleeps periodically based on the timing requirements of the world. If we dedicate one thread to each run this would limit the number of concurrent runs to the number of worker threads. As with any resource the number of worker threads is a limited one. Hence scalability becomes compromised.

To increase scalability the RunManager utilises a scheduled thread pool. Each run is scheduled according to the requirements dictated by the world involved in the run. In place of a continuous loop we now have a situation where a run is periodically serviced by a worker thread. When a thread has finished servicing a run it returns to the thread pool where it may be utilised to service another run, perhaps even the same run. The key point is that we no longer have a one-to-one relationship between the number of worker threads and the number of runs.

3) Low level User Interactions: Unlike the actions of minds, user actions will arrive asynchronously. However, from our architectural point of view we can treat a user as simply another mind in the sequence of minds. To facilitate this the concept of an asynchronous mind is introduced, see Fig. 6. An asynchronous mind is a mind with an associated action queue, which acts as a proxy for a user within the system. The action queue performs a buffer function between the mind and user until the mind can be queried.

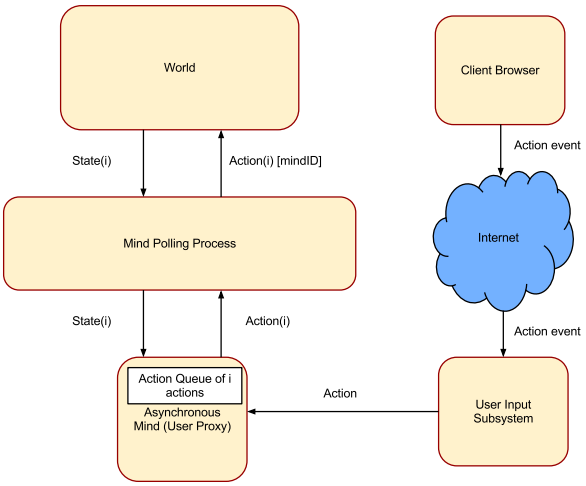

Fig. 6. A user proxy mind delivering i actions to a world.

The user input subsystem receives an action some time after it is performed by a user. This action is stored in the action queue associated with the user's proxy asynchronous mind. The action is processed when the proxy mind is queried by the RunManager.

C. Representing the State of a Generic World in WUIML

In order for a user to be able to access the system via a standard browser without having to install any third-party software the state of a world must be delivered to the users browser in a language that browsers understand, namely Hypertext Markup Language (HTML). Alternatively, the state of the world can be delivered to the browser in a proprietary format and, transformed into HTML within the browser.

To allow the XAI Server to be easily extended and enhanced in the future we would like the interface between the browser and the system to be loosely coupled. This means that we cannot afford to tie the development to a client specific

language such as HTML. A better approach would be to allow worlds to represent their user interface in a client independent form that could be easily transformed into any client-side representation. This would necessitate the client taking responsibility for performing such transformations.

There are many client independent user interface definition languages in existence. UIML [10], XUL [11] are just two examples. Both these languages have an associated learning curve and are far more complex than is necessary for our current development needs. The World-Wide-Mind [9] maintains a low barrier to entry for researchers, allowing moderately skilled programmers to successfully build minds and worlds for the system. We wish to emulate and build on this success. To facilitate this we developed our own client independent user interface definition language, WUIML.

WUIML is an acronym for World User Interface Markup Language. Although currently in development, it is mature enough to allow development of the demonstration world, ChatWorld. WUIML is an XML based markup language allowing it to be readily transformed into a client specific language. It defines several tags and attributes that allow elaborate layouts of user interface artefacts such as text areas, text inputs and buttons.

1) WUIML Windows: WUIML Windows are the basic elements of the WUIML language. A window defines the width and height of the worlds user interface and contains all other elements, directly or indirectly. Window tags only contain WUIML screen tags directly. A WUIML snippet containing a screen element contained within a window element is shown in Listing 1.

```
<window id='window_01'
        width=800
        height=600>
<screen id='screen_01'>
<!-- More elements are placed here -->
</screen>
</window>
```

Listing 1. A window element containing a screen element.

2) WUIML Screens: Only one screen may be visible to the user at any one time. By default the first screen declared is automatically visible. Subsequent screens, and any elements contained in those screens, are hidden by default. Thus, user interface elements can be hidden until needed. Screens are automatically sized to match the width and height of the window. Each screen tag must contain at least one layer tag. See Listing 2.

```
<window id='window_01'
        width=800
        height=600>
<screen id='screen_01'>
<!--Visible elements are placed here -->
</screen>
<screen id='screen_02'>
<!-- Hidden elements are placed here-->
</screen>
</window>
```

Listing 2. A window element containing two screen elements.

3) WUIML Layers: Layers allow stacking of WUIML elements in the z-direction i.e. each layer is displayed above the previous one. See Listing 3. This allows for some quite complex user interface designs to be achieved. Layers contain only panel elements. Layers are the width and height of their containing screen.

```
<window id='window_01'
        width=800
        height=600>
<screen id='screen_01'>
<layer id='layer_01'>
<!--
 Elements displayed at lowest z order 0
-->
</layer>
<layer id='layer_02'>
<!--
 Elements displayed at next z order 1
-->
</layer>
</screen>
</window>
```

Listing 3. A screen element containing two layer elements.

4) WUIML Panels: Panels are the WUIML elements which allow elements to be positioned. They have a width and height specified as a percentage of their containers dimensions and can be aligned vertically and horizontally. They are centred by default. Listing 4 shows a panel which is 50% the width and height of its container. It is horizontally centered by default, but vertically aligned to the top of its container.

```
<window id='window_01'
        width=800
        height=600>
<screen id='screen_01'>
<layer id='layer_01'>
<panel width=50 height=50 valign=top>
<!--
 Controls placed here.
-->
</panel>
</layer>
</screen>
</window>
```

Listing 4. A panel aligned to the top of its containing layer.

5) WUIML Controls: The remaining WUIML tags define controls, such as text areas, text inputs and buttons. All controls must be contained within a panel. Listing 5 shows a text input which is 90% the width and height of its containing panel.

```
<window id='window_01'
        width=800
        height=600>
<screen id='screen_01'>
<layer id='layer_01'>
<panel width=50 height=50 valign=top>
<input id='input_01'
        width=90 height=90/>
</panel>
</layer>
</screen>
</window>
```

Listing 5. A panel containing a text input.

D. The WUIML API

In order to speed the development of world user interfaces an API (Application Programming Interface) has been developed. Written in Java this interface allows world developers to define the user interface for their world. Below is example Java code showing how ChatWorld defines its user interface.

```
1
2  WUIMLWindow win = new WUIMLWindow(800,600);
3  WUIMLScreen screen = new WUIMLScreen();
4  WUIMLLayer layer = new WUIMLLayer();
5
6  WUIMLPanel panelMain = new WUIMLPanel(100,100);
7  WUIMLPanel panelMainTop = new WUIMLPanel(100,75,
8          WUIMLAlign.CENTER,WUIMLAlign.TOP);
9  WUIMLPanel panelMainBottom = new WUIMLPanel(100,25,
10         WUIMLAlign.CENTER,WUIMLAlign.BOTTOM);
11 WUIMLPanel panelMainBottomLeft = new WUIMLPanel(100,25,
12         WUIMLAlign.LEFT,WUIMLAlign.CENTER);
13 WUIMLPanel panelMainBottomRight = new WUIMLPanel(100,25,
14         WUIMLAlign.RIGHT,WUIMLAlign.CENTER);
15
16 WUIMLTextArea text = new WUIMLTextArea(50,50,true);
17 WUIMLTextInput input = new WUIMLTextInput(50,
18         WUIMLAlign.CENTER,WUIMLAlign.BOTTOM);
19
20 WUIMLButton sendButton = new WUIMLButton("Send");
21 WUIMLFieldMap fieldMap = new WUIMLFieldMap();
22 fieldMap.put("msg", input);
23 sendButton.map( onclick , say ,fieldMap);
24
25 panelMain.addChild(panelMainTop);
26 panelMain.addChild(panelMainBottom);
27
28 panelMainTop.addChild(text);
29 panelMainBottom.addChild(panelMainBottomLeft);
30 panelMainBottom.addChild(panelMainBottomRight);
31 panelMainBottomLeft.addChild(input);
32 panelMainBottomRight.addChild(sendButton);
33
34 layer.addChild(panelMain);
35 screen.addChild(layer);
36 win.addChild(screen);
37 WUIMLDocument doc = new WUIMLDocument();
38 doc.addChild(win);
```

Listing 6. WUIML API code to create the ChatWorld user interface.

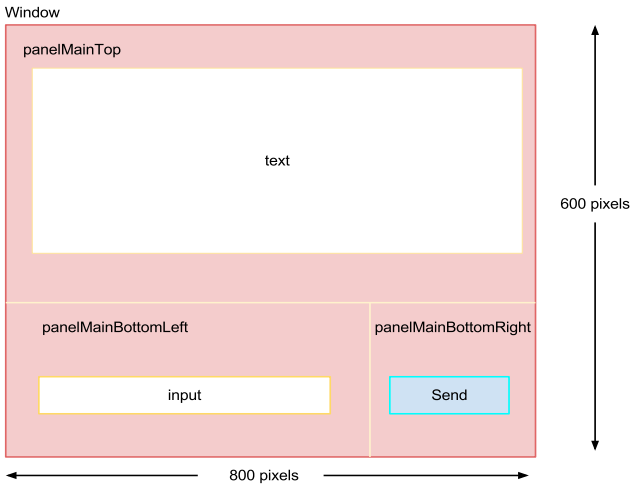

Fig. 7. The layout resulting from the code in Listing 6.

This simple, easy-to-use API obviates the need for developers to learn a proprietary syntax, such as WUIML. Instead even those moderately familiar with Java programming may use the API to create quite sophisticated user interfaces.

Fig. 7 shows the user interface resulting from the code shown in Listing 6. Listing 7 below contains the WUIML generated from the code in Listing 6. It is this WUIML that is sent to and interpreted by the browser to produce the user interface for ChatWorld.

```
1  <wuiml>
2  <window id='window_01' width='800'
3          height='600'>
4  <screen id='screen_01'>
5  <layer id='layer_01'>
6  <panel id='panel_01' width='100' height='100'
7          align='center'
8          valign='center'>
9  <panel id='panel_02' width='100' height='80'
10         align='center'
11         valign='top'>
12 <input id='input_01'
13         width='50'
14         align='center'
15         valign='center'/>
16 </panel>
17 <panel id='panel_03' width='100' height='20'
18         align='center'
19         valign='bottom'>
20 <panel id='panel_04'
21         width='80'
22         height='100'>
23 <input id='input_02' width='50'
24         align='center'
25         valign='center'/>
26 </panel>
27 <panel id='panel_05'
28         width='20'
29         height='100'>
30 <button
31  id='button_01'
32  text='send'
33  onclick='send:msg:input_02'/>
34 </panel>
35 </panel>
36 </panel>
37 </layer>
38 </screen>
39 </window>
40 </wuiml>
```

Listing 7. The WUIML behind the layout shown in Fig. 7.

E. Delivering the user interface to and receiving events from the browser.

Fig. 7 shows the resulting output for the code in Listing 6. Listing 7 illustrates the WUIML used to produce this output. How this user interface gets delivered to the browser and how events, such as button clicks, get returned to the running world is described in this section.

1) Delivering the User Interface to the Browser: Each world created by a user is required to implement the World interface. One of the methods the World interface defines is the 'getRunProfile' method. This method returns an instance of the RunProfile class, which is a wrapper class for, among other things, a WUIML document.

When a user chooses a number of minds and a world to run, that information is sent to the XAI Server as a run request

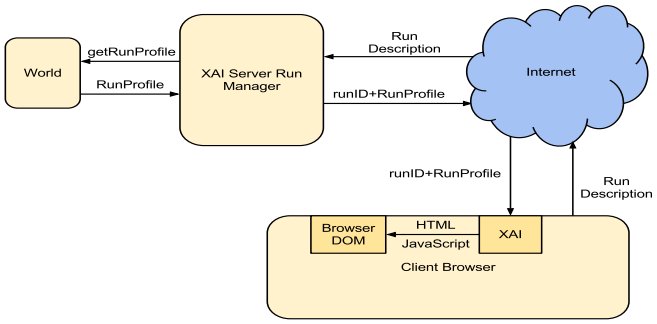

Fig. 8. User interface creation on the client.

description, see Fig. 8. The XAI Server will create instances of the world and the minds to run, before returning the RunProfile for the world and a run id to the browser. On the browser a JavaScript library called xai.js interprets the WUIML and produces HTML. It also provides JavaScript code to act as event listeners. When one of those event listeners fires it produces an action event as shown in Fig. 6.

2) Receiving Events From the Browser: Listing 8 repeats a section of code from Listing 6 that describes how the action event is generated.

```
WUIMLButton sendButton = new WUIMLButton("Send");
WUIMLFieldMap fieldMap = new WUIMLFieldMap();
fieldMap.put("msg", input);
sendButton.map( onclick , say ,fieldMap);
```

Listing 8. Mapping WUIML elements to action events.

The WUIMLFieldMap maps an action event field name to a control. In this case the action event field named ‘msg’ is mapped to the input control. The call to the map method of the Button class maps the onclick event of the button to the action event called ‘say’ with the parameter names ‘msg’ equal to the value of the input. This will produce the action event shown in Listing 9.

```
<action mindid="4"
        id="say"
        params="msg:Hello There!"/>
```

Listing 9. An XML representation of an action.

The Action shown here can be interpreted as, the mind with the id equal to “4” performed an action called “say” with a parameter called “msg” that was equal to “Hello There!”. The action and params attribute are only meaningful to the world that created them, which is correct as the XAI Server only acts as a conduit for this information. It does not attempt to interpret the information. This is utilised in our ChatWorld implementation described later.

3) Efficient client server communication: All of the interactions presented so far between the browser and the XAI Server have involved the browser initiating communication to the XAI Server. Browsers communicate with servers using HTTP (HyperText Transport Protocol). A HTTP connection is made to the server. The browser sends a request, waits for a response and then drops the connection. Setting up and tearing down such connections is a time intensive process and HTTP 1.1 [12] specifies that a client should only maintain at most two connections to a server at any one time. These issues with this model of communication are addressed in the XAI Server.

Having the browser initiate communication to the XAI Server for run setup is acceptable because it happens infrequently (only once in the life cycle of the run), but, user interface events will happen frequently and there is a need to send frequent updates to the user interface elements on the browser so that state changes are detected by the user. The ChatWorld implementation ensures updates happen once every second, but worlds requiring more frequent updating are easily conceivable. For example, any world wishing to simulate character interactions in 2D or 3D environments would need tens of updates every second. This would quickly become unmanageable using the conventional HTTP request response mechanism as setting up a HTTP connection and tearing it down is a time intensive process.

The solution to the faster update problem is a full duplex communication channel between the browser and the XAI Server. Fortunately a mechanism exists to provide this in the form of CometD [13].

CometD operates over HTTP and transparently provides a full duplex communication channel from client to server. How this is achieved is beyond the scope of this document. Interested readers are directed to the CometD documentation [13].

Even though a mechanism exists for full duplex communication channels the user may wish to have more than one run progressing from within the browser simultaneously. An account must be taken of the two connection limit imposed by HTTP 1.1. If a CometD connection is opened to the server for each run a limit of two connections per client browser is possible. Even though not all browsers enforce the two connection limit the system must allow for the situation to occur. Instead of creating a new CometD connection each time the user initiates a run, a single CometD connection channel is used to multiplex the run event actions and UI updates over this channel.

Within the JavaScript library, xai.js a global CometD connection object is stored. When a user first initiates a run, the CometD connection is opened. Internally the XAI Server maintains a list of run ids assigned to each client. Clients are identified by their CometD client id. A run id is returned to the client in response to each run request. All information to and from the server in relation to this run after this point must contain this run id.

CometD allows the XAI Server to detect when a client disconnects by firing a disconnect event. The XAI Server detects this event and removes a user from the list of users connected to a run. The users proxy mind is also removed from the run and the associated world is notified through the

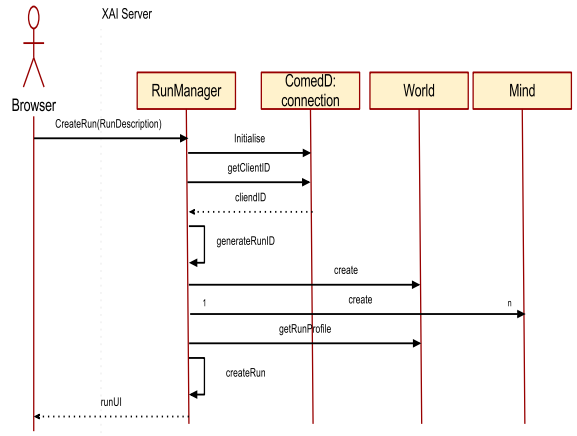

Fig. 9. Sequence of events for run creation

'removeMind' method defined in the World interface. If a run looses all connected users then it is removed.

Once a run has been initiated (See Fig. 9) users can request to join the run (See Fig. 10). When this occurs the client id for that user is added to the list of clients involved in the run. A new AsynchronousMind is set up for the client and the world is notified that a new mind has been added through the 'addMind(int id)' method defined in the World interface.

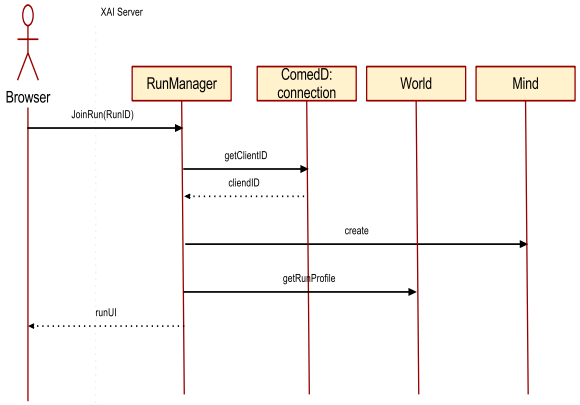

Fig. 10. Sequence of events for joining a run

III. Demonstration Worlds and Minds

This section describes the demonstration world developed for the XAI Server and the worlds interaction with the four demonstration minds: ChatMind, GossipMind, JabberMind and HybridChatMind.

A. ChatWorld

The XAI Server does not try to impose a structure on communication between the world and minds. However, some structure is necessary to allow instances of AsynchronousMind to format actions to the world.

As described in Listing 9 an action has an identifier and parameters. The identifier we used in that example was "say". In reality this identifier is an integer defined by ChatWorld. The parameter list is a JSON (JavaScript Object Notation) [16] object. This is a structure that is easy to parse and generate. Fig. 11 shows a representation of a typical Action object that ChatWorld would receive from a mind designed to interoperate with it.

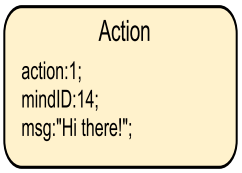

Fig. 11. A typical action object received by ChatWorld.

All action objects in the XAI Server will have an action field and a mindID field. The only field unique to ChatWorld actions is 'msg'. Although actions have some format imposed, the state of a world requires no special format. Coincidentally ChatWorld imposes a JSON format on its state. The ChatWorld state object contains an array of the last ten messages sent by minds. An example of this is given in Fig. 12.

Fig. 12. A ChatWorld state object.

The state above in Fig. 12 shows the first five comments in a run including two minds and one user.

B. ChatWorld Minds

The minds designed for ChatWorld are capable of producing Actions and understanding States as described previously.

1) ChatMind: When ChatMind receives a state from an instance of ChatWorld it looks at the last comment and identifies the mind responsible for that comment. An internal list of strings is scanned and one is found with the greatest number of words in common with the comment. A string is produced consisting of the id of the mind that made the last comment and the new string.

2) JabberMind: JabberMind simply produces a random string from a list of internal strings each time it is asked for an action.

3) GossipMind: GossipMind chooses a random number between 1 and 10. If the number is greater than 5 it simply repeats the last comment made. Otherwise it picks a string from a list stored internally.

4) HybridChatMind: HybridChatMind is a hierarchical mind. It stores three instances of minds internally, one of each type of mind described here. When asked to produce an action HybridChatMind chooses randomly between its three internal minds and returns whatever that mind returns.

IV. Conclusion

Current development on the XAI Server has highlighted several areas for future development and improvement of the system:

1) WUIML will be extended to support 2DCanvas and 3DCanvas elements. These can be implemented on the existing browser framework. HTML5 [14] supports a canvas element and WebGL [15] adds support for native 3D rendering within browsers. This will allow researchers to develop worlds that generate 2D and 3D geometric representations of their states. It will also help to provide a richer user experience.
2) We envisage that, over time, a large reservoir of worlds and minds will develop naturally as the XAI Server matures and gathers a strong user community.
3) The addition of third party libraries will be extremely useful. 2D and 3D physics engines would allow worlds to produce more interesting simulations. Audio would add a completely new dimension to the user experience, and would be valuable in simulating real-world environments.
4) All communication between the browser and XAI Server is over HTTP currently. HTTP offers reliability, but at the cost of bandwidth. However, if updates from the server to the client are very frequent (as they would need to be in a 3D simulation) it may be acceptable to lose information on the basis that the information will be retransmitted later. A facility to allow lossy communication would reduce network bandwidth requirements. Another possibility which would reduce network bandwidth requirements would be a form of fast compression.
5) The scheduling scheme currently in use does not place any restriction on the time taken for minds to produce actions. This could potentially allow minds to monopolise threads. A more sophisticated scheduling scheme is necessary where a worker thread servicing a mind can be pre-empted should the mind attempt to monopolise the thread.
6) Large scale testing of the system will help define upper boundaries for both the number of minds, the number of users in a single run and the number of runs on a single server instance.

With the ChatWorld example we demonstrated that the XAI Server can emulate the functionality of a simple ChatBot. Minds consisting of a single component and minds consisting of a hierarchical structure have been demonstrated. The XAI Server system also has the ability to run several minds in a world concurrently.

We have shown how WUIML can be used to deliver "world defined" user interfaces to a standard browser and how such a user interface can be used to allow users to interact in real-time with minds and worlds executing on the XAI Server. Multiple user interactions have also been shown to be feasible.

References

[1] Minsky, M. "The Society of Mind", Simon and Schuster, 1985.
[2] Gasser, L. Braganza C. Herman N. "MACE: 'A flexible testbed for distributed AI research.', In "Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence", Addison-Wesley, 1999.
[3] Wooldridge, M. Vandekerckhove, D. ,"MYWORLD: An Agent-Orientated TestBed for Distributed Artificial Intelligence" Proceedings of the 1993 Workshop on Cooperating Knowledge Based Systems, 1993.
[4] Reticular Systems Inc. ,"AgentBuilder: An Integrated Toolkit for Constructing Intelligent Software Agents" White Paper, April, 2012.
[5] Humphrys, M.,"The World-Wide-Mind: Draft Proposal" Dublin City University, School of Computing, Technical Report no. CA-0301, Feb 2001.
[6] Walshe, R. Humphrys, M. First Implementation of the World-Wide-Mind, poster in Advances in Artificial Life: Proceedings of the 6th European Conference on Artificial Life (ECAL 01), Prague, Czech Republic, Sept 2001.
[7] O'Connor, D. Humphrys, M. The implementation of a Distributed Hierarchical Mind on the Internet using the World-Wide-Mind, Dublin City University , School of Computing, Technical Report no. CA-0302.
[8] O'Leary, C. Humphrys, M. Walshe, R. Constructing an animat mind using 505 sub-minds from 234 different authors, Proc. 8th Int. Conf. on Simulation of Adaptive Behavior (SAB-04), July 2004, Los Angeles, CA.
[9] Mac Fhearaí, O. Humphrys, M. Walshe, R. A High-Speed Architecture For Building Hybrid Minds, poster presented at 3rd International Conference on Agents and Artificial Intelligence (ICART 2011), Rome, Italy, 28-30 Jan 2011.
[10] OASIS. (2007) UIML Working Draft. [Online]. Available: http://www.oasis-open.org/committees/download.php/28457/uiml-4.0-cd01.pdf
[11] Mozilla Developer Network. (2012) XUL Documentation. [Online]. Available: https://developer.mozilla.org/en/XUL
[12] The Internet Society. (1999) Request for Comments: 2616. [Online]. Available: http://www.w3.org/Protocols/rfc2616/rfc2616.html
[13] Bordet, S. (2011) The CometD Reference Book. [Online]. Available: http://docs.cometd.org/reference/
[14] W3C (2012) W3C Working Draft. [Online]. Available: http://www.w3.org/TR/html5/
[15] Khronos Group (2012) WebGL Specification. [Online]. Available: http://www.khronos.org/registry/webgl/specs/latest/
[16] Ecma International (2011) ECMAScript Language Specification. [Online]. Available: http://www.ecma-international.org/publications/files/ecma-st/ECMA-262.pdf
[17] Roth, G. (2007) Architecture of a Highly Scalable NIO-Based Server. [Online]. Available:http://today.java.net/pub/a/today/2007/02/13/architecture-of-highly-scalable-nio-server.html