

MXP: a universal Metaverse eXchange Protocol

By Ben Lindquist (Arkowitz)

Tommi S. E. Laukkanen and I have merged the next versions of our protocols into something called Metaverse eXchange Protocol (MXP). This is a combination of the principles of CICP, my protocol for allowing external entities to place interactive content into virtual worlds; and SETP, Tommi's UDP-based protocol for efficient client->server and server->server interaction.

MXP begins with first principles, which are described below. Following the discussion of first principles is a list of terms which we consider "foundational definitions" for MXP; following that list is a description of the messages comprising MXP.

First Principles

We begin with the principle that a virtual world need only contain things which are perceivable. This is the key to interoperability. Logic, process, and internal object attributes exist outside the world. Just as knowledge and thought exist within the minds of humans and are only evidenced when they cause something perceivable to happen, such as a gesture or movement, or a sound; so the humans and programs participating in a virtual world are using the world only as a medium of expression and perception. Concepts such as internal object state, scripting, and logic belong to the participants, not the world.

Our next principle is one of layering. The communication necessary for a virtual world to provide a shared experience among participants takes place using a transport layer and a content definition layer. The transport layer provides a mechanism by which participants may make each other aware of the presence of objects, and then allow participants to transmit messages among their objects. The content definition layer provides the means for participants to publish any specific details and/or assets which define the appearance of their objects, the capabilities of their objects, the available options for interacting with their objects, and the state of any external attributes of their objects.

The layering principle allows us to limit the scope of MXP. MXP provides the transport layer and only enough of the content definition layer to enable the expression of the states of mutable, public attributes of objects. The content definition layer will specify the format and meaning of messages transmitted from object to object using MXP, and will provide a means for participants to publish and fetch content assets associated with objects. It is expected that the content definition layer will utilize XML for encoding of messages and definitions, and that assets and definitions will be fetched using HTTP. To extend the analogy to HTTP and the World Wide Web: the transport layer (MXP) is equivalent to HTTP, and the content definition layer is equivalent to MIME types, html, and other content.

We have chosen to formulate MXP as a hub-driven protocol. A piece of “server” software is expected to perform the hub function, and “client” software is expected to communicate with the hub on the behalf of users, or “participants”. All messages are transmitted from a participant to a hub, or from the hub to one or more participants. Participants do not transmit MXP messages to each other; they interact with one another via a hub. We believe that the use of a server as a messaging hub reduces network traffic compared to a peer-to-peer messaging system; furthermore, the Internet is structured in a way which promotes the use of a server with a lower-latency, higher-bandwidth Internet connection than the clients. The choice of a hub paradigm is one of the principles.

It should be noted that no simulation is performed by the MXP hub. It is simply a messaging hub which allows participants to inform each other of their objects and modifications to those objects; it is the responsibility of each participant to simulate the "spacetime" itself. There is no master simulation from the perspective of MXP. In order to allow participants to simulate smooth motion of objects from one point to another, or one orientation to another, modification messages contain a field for indication of the time in the future at which the modification should complete. Time synchronization is handled by the hub; it is expected to keep track of the difference between its local time and the local time of each participant, and convert all times to participants' respective local times when transmitting events.

It is often desirable to create a piece of server software which performs many services for its clients; for example, a “virtual world server” which performs collision detection and/or physics simulation for its clients. It may also be useful for a “virtual world server” to allow clients to create objects containing scripts which are executed within the server itself. In the world of MXP, this example server can be implemented as a hub and one or more participant programs. We refer to these server-level participant programs as “daemons”. A daemon is a participant like any other from the perspective of MXP and the hub, but from the perspective of other participants it performs a useful service. We expect that a daemon will be constructed which simulates all of its fellow participants' objects at a detailed level using their content definitions, detects collisions among them, and transmits messages to the owners of colliding objects.

Despite our intention that the MXP hub know as little as possible about the objects it is handling the messaging for, it is useful for performance reasons for the hub to store a set of public attributes for each object. This allows participants to update public attributes of their objects individually, and the hub to send a complete list of the public attributes, along with their current states, to a participant when she observes a particular object for the first time. Public attributes include things such as whether an avatar object is walking. Walking will typically mean that a walking animation, which is part of the content definition layer, is to be executed in order to properly simulate the object. The "time to complete" field applies to public attribute state changes.

Finally we have a principle of identification. Since a server acts as a communication hub for participants, the server may be expected to assign each participant a unique,

temporary identifier. Participants are then expected to provide identifiers for their objects which are unique only with respect to the set of objects they own. The participant identifier joined with the object identifier is unique for each object within the set of objects being handled by a particular hub or federation of hubs. Note that we are not precluding the use of globally unique identifiers such as UUID's; such identifiers may be very useful within the content definition layer, and could be substituted for the above identification scheme as well. The important principle is that, from the perspective of MXP, all objects are merely temporary expressions of what is perceivable. Sale or permanent exchange of an "actual" object in a permanent sense would not be "governed" by MXP, though messages enabling the exchange could be transmitted using MXP.

Foundational Definitions

Bubble

The overall goal of MXP is to enable multiple participants to experience a shared, simulated, spacetime. Mathematically a spacetime is a four-dimensional continuum. We constrain this continuum such that the time dimension only moves forward and is constantly incremented by some sort of clock synchronization among the participants. We also may constrain this continuum such that there are a finite number of positions between two given points rather than an infinite number. This means it is not necessarily a continuum in the formal sense of the word. We refer to a spacetime defined by a single coordinate system as a "bubble".

Multiple bubbles may exist, and need not exist within the same coordinate system. A bubble may exist within a region of another bubble. A bubble may also border another bubble. Bubbles may also overlap.

Participant

Participants are entities which are able to perceive aspects of a bubble and/or have an effect on aspects of the bubble. Participants may be humans or programs. Participants perceive and affect bubbles through the use of objects, not directly. In order to participate in a bubble, an entity must cause an object to exist within the bubble.

Object

An object is an atomic unit of content. An object's effect on a spacetime may be comprised of multiple types and occurrences of content, but is treated such that either the entire object exists within a given bubble, or none of it exists. If a bubble is assumed to have a boundary, and some object overlaps that boundary, the object can still be expressed in terms of the coordinate system of that bubble and as such is considered to exist fully within that bubble.

Injection commands place objects into bubbles and are executed against individual objects. Modify and ejection commands, which modify objects and cause them to be removed from a bubble, respectively, may be executed against individual objects or groups of objects.

Every object affecting a given bubble belongs to a participant in that bubble. If a program issues an injection command, placing an object into a bubble, that program is a participant in that bubble. If the program is a human interface, the human using the human interface is considered a participant.

Content

Content is anything which can be perceived in the bubble. Content only exists as a part of an object. All participation in a bubble involves content. This includes images or textures, three-dimensional shapes, audio, fog, and light; it also includes changes occurring to any of these. Content may be injected into a bubble, modified within a bubble, ejected from a bubble, or perceived within a bubble. Injection, modification, or ejection of content is performed via the protocol, by use of a command; perception of content is indicated via the protocol by receipt of an event. Injection, modification, and ejection of content all result in perception events.

Land and avatars are objects like any others as far as MXP is concerned. In a typical real-world usage of MXP it is expected that the first participant to join a particular bubble will inject a land object into the bubble. This first participant may be a program which is developed specifically to place land into bubbles.

Awareness

Objects have awareness bounds. However they are defined geometrically, the awareness bounds indicate a region of space within a bubble. If this region of space is affected by other objects, via injection, modification, or ejection commands, events representing those effects will be sent to the participant responsible for the object. Objects may be defined with an empty awareness bounds, indicating that no events should be received.

Upon issuing an injection command for an object, a participant will receive events which correspond to all objects which were already affecting the bubble within the awareness bounds of the new object.

A more complete discussion of foundational definitions is available at <http://bubblecloud.org>, in the "Domain Model" section of the wiki.

Messages

We divide the messages which comprise MXP into two sets: commands, which are issued from a participant to a server; and events, which are issued from a server to a participant. Descriptions of these messages follows.

Commands

JOIN: this command informs a server that one wishes to join a bubble; it includes one's local time

INJECT: place an object into a bubble; includes the initial location, orientation, rough physical bounds, awareness radius, content description version, object id, initial attribute/state list, and url to the definition of the content

EJECT: remove one's object from a bubble; includes the object id

MODIFY: modify aspects of one's object; includes object id, new position, new orientation, new rough physical bounds, new awareness bounds, attribute/state list, and time to take effect

LEAVE: eject all one's objects

INTERACT: send a message to someone else's object; includes id of the other participant, id of their object, optional id of one's object which is interacting, and XML payload to be delivered

Events

OBSERVATION: an object has been observed by one of your objects; includes participant id and object id of observed object, location, orientation, rough physical bounds, content description version, participant id of owner, object id, url to content definition, and current attribute state

MODIFICATION: an object which you have already observed has been modified

INTERACTION: another participant has sent a message to one of your objects

EJECTION: an object which you have previously observed has been ejected

Current message implementation work is available at <http://bubblecloud.org>.

Conclusion

Once we have finalized the complete and minimal set of commands which comprise MXP, we will specify two implementations of the protocol: a text-based implementation for use over TCP sockets, and a binary implementation for use over UDP. We will also develop a reference implementation of an MXP hub, daemon, and human-useable client. We hope to release the protocol standard under the auspices of an official standards body, and plan to release the reference implementations to the public domain.