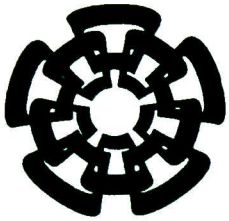
OT-683-551

DOU.-2012

XX (199684.1)



Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional
Unidad Guadalajara

Distribución de Puertos Visuales para Ambientes Virtuales

Tesis que presenta:

Alma Verónica Martínez González

para obtener el grado de:

Doctor en Ciencias

en la especialidad de:

Ingeniería Eléctrica

Directores de Tesis

Dr. Félix Francisco Ramos Corchado
Dr. Mario Angel Siller González Pico

CINVESTAV del IPN Unidad Guadalajara, Guadalajara, Jalisco,

ID. 199523-2001

Distribución de Puertos Visuales para Ambientes Virtuales

por:
Alma Verónica Martínez González
Maestra en Ciencias
Cinvestav Unidad Guadalajara 2003 - 2005

Directores de Tesis
Dr. Félix Francisco Ramos Corchado
Dr. Mario Ángel Siller González Pico

Resumen

La creación de un Ambiente Virtual (*AV*) no es una tarea fácil, debido a que incluye diferentes aspectos, tales como: el tipo de situación a representar, la interacción entre los usuarios y el sistema, el comportamiento de los objetos pertenecientes al *AV*, etc. Los procesos necesarios para la generación de un *AV* se puede conceptualizar utilizando el Modelo-Vista-Controlador (*MVC*) [1], que define tres componentes: (i) Modelo: se encarga de la gestión de la lógica del sistema (garantizar la integridad de la información) y procesar la información adquirida por el controlador, (ii) Vista: presenta los resultados del modelo en un formato adecuado que permita la interacción del usuario con el sistema, y (iii) controlador: responde a los eventos (las acciones del usuario) y ofrece como de entrada al modelo. Algunas de estas acciones pueden ser representadas a través de la vista. La mayor parte de la investigación de analizadas acerca de *AV* están relacionadas con el módulo del Modelo, debido a que controla la lógica de los cambios en el *AV*. Sin embargo, esta lógica se ha vuelto más y más compleja. Por lo tanto, es necesario revisar los requisitos que debe cumplir la Vista con el fin de presentar al usuario la correcta evolución de la *AV*. Este trabajo está relacionado con el módulo de la Vista.

La primera parte de nuestra propuesta identifica las características de la Vista (en adelante, la Vista la nombraremos como ***Puerto Visual*** o ***PV***) que permiten la interpretación de los resultados del análisis de la lógica del módulo del Modelo. Con este resultado, es posible definir un conjunto de políticas de gestión de los elementos dentro del *AV*. Nuestra propuesta también maneja los cambios del *AV* en tiempo real para que el usuario perciba una continua evolución del *AV*.

Un *VP* debe cumplir principalmente con las siguientes tareas: (i) gestionar adecuadamente los elementos de *AV*, (ii) realizar la visualización en tiempo real, (iii) permitir la participación de varios usuarios ubicados en diferentes zonas geográficas, es decir, hacer una distribución de *PVs*, y (iv) mantener una consistencia de *AV*. La distribución de los *VPs* no es una tarea trivial, ya que se trata de diferentes aspectos de la red, tales como la pérdida de paquetes, demora, la congestión en el canal de comunicación causados por el envió excesivo de mensajes, entre otros. Para hacer la distribución de los *PVs* es necesario identificar los aspectos que inciden negativamente en la tarea de los *PVs*, y así diseñar un mecanismo para su correcta administración.

La evolución de una escena está a cargo de los *PVs* que se ejecutan en los nodos de una red heterogénea. En esta propuesta, los nodos intermedios se consideran como una caja negra, se realizan mediciones de extremo a extremo. La distribución de una escena (tarea realizada por los *PVs*) involucra diferentes aspectos que van de la representación de la información sobre los cambios de la *AV* (capa de aplicación) a aspectos de la red, tales como ancho de banda y el retardo. Se ha demostrado que cuando se utiliza el paradigma de diseño de cruce de capas (Cross-

Layer Design CLD) se mejora el desempeño de estos aspectos [**2**]. Se considera que dicho paradigma pueden ser aplicado para determinar las políticas de gestión del AV, a fin de mejorar su transmisión en tiempo real.

Abstract

To create a Virtual Environment (*VE*) is not an easy task, it includes different aspects such as: type of situation to represent, the interaction among the system and users, the behavior of all objects belonging to the *VE*, etc. The processes needed for generating a *VE* can be conceptualized using Model-View-Controller (*MVC*) [**1**] which defines 3 components: (i) Model: it is in charge of managing the system's logic (ensuring information integrity) and it processes the acquired information by the Controller; (ii) View: it presents the results of the model in an adequate format allowing the interaction of the user with the system; and (iii) Controller: it responds to events (user actions) and provides them as input to the Model. Some of these actions may be represented through the View. Most of the research of *VE* deals with the Model component, mainly because it controls the logic of changes in the *VE*. However, this logic has become more and more complex. Therefore it is necessary to review the requirements that the View must fulfill in order to present the user the correct evolution of the *VE*. This work is related to the View component.

The first part of our proposal identifies View (henceforth the View will be called ***Visual Port*** or ***VP*** for short) characteristics that allow interpreting the results of the logic analysis of the Model component. With this result, it is then possible to define a set of management policies of the elements within the *VE*. Our proposal also manages the changes of the VE being executed in Real Time so that the user perceives the continuous evolution of the *VE*.

A VP must comply primarily with the following tasks: (i) properly manage the *VE* elements, (ii) perform real time visualization, (iii) permit the participation of several users located in different geographical areas, ie make a distribution of VPs, and (iv) maintain a consistency of *VE*. The distribution of VPs is not a trivial task because it involves different network aspects such as packet loss, delay, congestion in the communication channel caused by excessive messages he sent, among others. To make the distribution of *VPs* is necessary to identify the aspect that impact more negatively the task of the *VPs*, and then design a mechanism for its proper administration.

The evolution of a scene is in charge of the *VPs* which are executed in the nodes of the heterogonous network. In this proposal the intermediate nodes are considered to as a black box and end to end network measurements are rather performed. The distribution of a scene involves different aspects going from the representation of the information regarding changes of the *VE* (application layer) to network aspects such as bandwidth and delay. It has been proved that when the network cross-layer design (*CLD*) paradigm is employed a better performance of such aspects can be achieved [**2**]. It is believed that such paradigm can be applied to derive the management policies of the *VE* in order to improve its Real Time transmission.

Agradecimiento

A CONACYT por el apoyo para la terminación de mis estudios.

Índice general

Índice de figuras

Índice de tablas

Lista de Abreviaciones

Acrónimo	Definición	*Acrónimo*	Definición
α	Ambiente Virtual	o	objeto 3D
a	Avatar	P	numero de ciclos de procesamiento
ac	acciones	Pa	Paquete
AI	Área de Interés	π	estado emocional
AV	Ambiente Virtual	P_2P	Peer-To-Peer
β	objetos 3D	PA	Perfil Alto
C	Coordinador	PB	Perfil Bajo
CI	Cambio de imagen	P_C	Paquete comprimido
CLD	Cross-Layer Design	PM	Perfil Medio
CM	Consciencia Mínima	P_S	Paquete sin comprimir
Col	verificación de colisiones	PV	Puerto Visual
CT	Carga de Trabajo	PvD	Persistencia de la visión
DIS	Distributed Interactive Simulation	QoE	Quality of Experience
EAV	Evolución del Ambiente Virtual	QoS	Quality of Service
ESA	Estado del Ambiente	ρ	personalidad
Ev	Evolución - Ev(α) -> evolución del AV	RD	Recursos de Red
$ExEn$	Existencia de entidades	Rev	Respuesta del AV
FPS	First Person Shooter	rp	retardo promedio
γ	Avatares	RTD	Ranura de Tiempo Dinámica
H	promedio de nodos destino	s	solicitudes

φ	proceso	t	tiempo
k	numero de intentos de integración	T	tiempo requerido para la entrega
$kmax$	tope máximo de intentos de integración	τ	acciones validas
$LIA^{3}D$	Lenguaje de Interfaz para Animaciones en 3D	TP	Tiempo de Propagación
m	movimiento	tp_r	tiempo de procesamiento
M	Numero de mensajes	TPU	Tiempos de los Perfiles de los Usuarios
MA	Mensajes de Actualización	TR	Tiempo Real
MC	Mensajes de Consistencia	$TReq$	Tiempo Requerido
$Mphi$	Movimiento phi	TS	Tiempo de Serialización
MR	Máximo Retardo	TTA	Tiempo Total de una Acción
MVC	Modelo-Vista-Controlador	TTL	Time To Live
N	Nodo	Vac	validar acción
NS	Nodo Simple	VCL	Verificación de la consistencia Local
NTP	Network Time Protocol		

Capítulo 1 Introducción

Objetivos:

- Describir el área de investigación, así como la definición del problema, motivación y objetivos de este trabajo de investigación.
- Describir brevemente la organización de este documento.

1.1 Antecedentes

Un Ambiente Virtual (AV) es una interfaz que permite a los usuarios visualizar e interactuar en situaciones generadas y administradas por dispositivos de cómputo. Actualmente, los Ambientes Virtuales (AVs) están siendo utilizados en diversas áreas del conocimiento, tales como: medicina, educación, entrenamiento militar, desarrollo y planeación urbana [3] [4] [5] [6], entre otras. En un AV se puede representar una situación específica requerida por el área de conocimiento, describiendo dónde y cómo cambia. En este trabajo estamos interesados en AV donde existen avatares que representan humanos. En [7] se analiza el comportamiento humano bajo ciertas circunstancias que simulan situaciones de la vida real, para cumplir este objetivo se dota de cierta inteligencia artificial a los avatares quienes representan a los seres humanos, dándole la habilidad de animarse por sí mismo reaccionando a los cambios que ocurren en el AV. Lo anterior se hace con la finalidad de crear simulaciones más apegadas a la realidad.

Los procesos que permiten generar un AV pueden ser modelados utilizando el patrón de arquitectura de software nombrada Modelo-Vista-Controlador (MVC) [1]. En la **Figura 1** se muestran estos tres módulos, los cuales se definen en función del comportamiento del software a desarrollar.

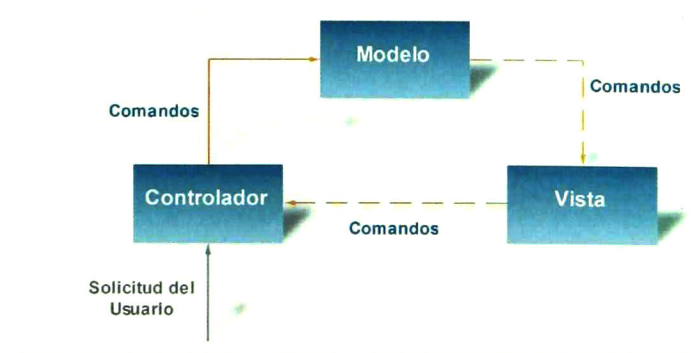


Figura 1: Esquema general del Modelo-Vista-Controlador (MVC).

En los AVs se definen los módulos de la siguiente manera:

- ***Modelo***: analiza la coherencia del comportamiento de las entidades en el AV y las consecuencias generadas por los cambios en el ambiente.
- ***Vista:*** realiza la representación de los resultados del Modelo, de tal forma que puedan ser comprendidos por el usuario.
- ***Controlador:*** recolecta las acciones del usuario.

Al clasificar los trabajos de investigación que utilizan *AVs*, la mayoría de ellos recaen en el *Modelo*, ya que este controla la lógica de los cambios en el *AV*. Sin embargo, la lógica del ambiente cada vez se vuelve más compleja, haciendo necesario un análisis de los requerimientos que debe cumplir el módulo de la *Vista*, al cual de ahora en adelante en este documento llamaremos **Puerto Visual (*PV*)**. Un *PV* debe de ser capaz de dar a conocer al usuario los cambios ocurridos en el *AV*, es decir, mostrarle la Evolución del Ambiente Virtual (*EAV*). La *EAV* se puede realizar mediante una representación gráfica en 3D, que ofrece una salida visual permitiendo a los usuarios finales comprender de manera fácil e intuitiva dichos cambios.

1.2 Definición del problema

Como parte de esta investigación, se identificaran las características que permitan interpretar los resultados del análisis lógico del *Modelo*, para determinar la capacidad que debe tener el *PV* y cumplir con los requerimientos necesarios que satisfagan al usuario final. Los resultados del *Modelo* del *MVC* se consideran no fiables, por lo cual un *PV* debe administrar eventos como: posibles colisiones entre entidades en movimiento, concurrencia de acciones para un mismo avatar, manejo de la sincronización de mensajes, entre otros aspectos no menos importantes. Las características que debe cubrir el *PV* deben ser lo suficientemente generales como para poder ser aplicadas a cualquier tipo de elemento en cualquier tipo de *AV*. El *AV* puede permitir la participación de **varios usuarios los cuales pueden encontrarse en localidades geográficas distintas**, por lo cual se deben distribuir los Puertos Visuales del *AV*.

Partiendo del concepto de que un *PV* es la interfaz de comunicación del usuario con el *AV*, la distribución de Puertos Visuales involucra diversos aspectos, como:

- La calidad de experiencia (*QoE* - Quality of experience) ofrecida al usuario.
- La administración de los elementos del *AV* para realizar una evolución consistente y en Tiempo Real.
- Los cambios en el *AV* deben ser lo más apegado a la realidad.
- La concurrencia entre los elementos y el *AV*.
- Una arquitectura adaptable tanto a las condiciones de red como a la estructura del *AV*.
- Mecanismos para reducir el impacto en la red de los mensajes de comunicación entre los *PVs* de un *AV*.
- Etc.

Estos aspectos han sido analizados de forma individual en otras investigaciones para aplicaciones como simuladores, video juegos, entre otros. Sin embargo, no se ha encontrado en el estado del arte revisado un trabajo que los considere en conjunto para su aplicación en la visualización de la evolución de un *AV* distribuido. A continuación se describen más a detalle los aspectos mencionados que son los problemas abordados en la presente tesis.

La calidad de experiencia (QoE) [8] se define como: "*La percepción del usuario de lo que está siendo presentado por un servicio de comunicación o interfaz de usuario de la aplicación. Es una medida de la aceptación general de un servicio o aplicación*". Es necesario considerar la QoE, debido a que los usuarios interactúan directamente con los PVs. Además de que el usuario final debe interpretar los resultados fácilmente. En este trabajo estamos interesados en aplicaciones donde la evolución del ambiente se realiza utilizando una representación en 3D.

La EAV distribuida debe realizarse en tiempo real (TR), para que los usuarios no perciban cambios bruscos o inconsistencias y la EAV sea *continua y suave* ("continuos and soft"). Para cumplir con este objetivo, es necesario considerar la diversificación existente entre las áreas de aplicación de los AVs, que repercute en el análisis de las características esenciales tanto para la representación de un caso sencillo como podría ser un cuarto vacío, hasta un ambiente complejo como lo es la evolución de una ciudad.

Un AV está formado por n elementos que interactúan entre sí, los cuales tienen características propias que permiten realizar animaciones diversas. Para animar a un elemento del AV, es necesario considerar las consecuencias de la animación sobre él mismo, sobre los demás elementos y además sobre el mismo escenario donde deben ocurrir los cambios. Las consecuencias pueden ser desde una colisión entre elementos, hasta la identificación de animaciones no existentes para el (los) elemento(s). Además, es necesario administrar la concurrencia entre las entidades (elementos) y la misma entidad que realiza la acción (animación). Por ejemplo, en la simulación de una situación donde están involucrados un avatar y varias cajas de diferentes tipos como cartón, madera, vidrio, etc. Una vez iniciado el ambiente, si se solicita al avatar que camine y este al hacerlo se tropieza con una o varias cajas, en este momento es necesario considerar las características propias de cada elemento involucrado para poder realizar los cambios correspondientes en todos los PVs, esto es necesario para asegurar una QoE aceptable. Es decir, los cambios se deben realizar de una forma más natural, para lo cual es necesario tomar en cuenta dos tipos de concurrencia:

1) ***Concurrencia en una entidad***: una entidad puede estar dividida en diferentes secciones, las cuales tienen limitaciones que pueden ser modificadas:

$$entidad_i = \{seccion_1, secccion_2, ..., seccion_n\}$$

$$seccion_n = \{\, limite\_rotacion\ \{min(x, y, z), max(x, y, z)\},$$
$$limite\_translacion\ \{min(x, y, z), max(y, y, z)\},$$
$$limite\_escala\ \{min(x, y, z), max(x, y, z)\}\}$$

$$Concurrencia_e \text{ -> } \{seccion_n\}$$

Si más de una acción recae en una misma sección, se debe administrar la concurrencia de $n\text{-}acciones$, ya que para representar una animación compleja, se requiere animar

diferentes secciones al mismo tiempo. Por ejemplo, el caminar de un avatar involucra modificar las secciones que representen a sus extremidades inferiores y superiores.

2) ***Concurrencia entre n-entidades***: los cambios en el AV ocurren por acciones que pueden realizarse por *n-entidades*, es decir, es posible que dos entidades requieran el mismo recurso al mismo tiempo. Por ejemplo, si dos entidades humanas desean abrir una puerta, se deben administrar la concurrencia en la puerta así como los resultados en cada entidad humana.

$$Concurrencia_{n\text{-}e} \text{ -> } \{entidad_i, entidad_{i+1}, \ldots, entidad_n\}$$

Los cambios en el ambiente pueden ser ilógicos u ocurrir a destiempo, por esta razón es necesario considerar que las solicitudes entrantes no son fiables. Para minimizar el impacto de este tipo de errores, se requiere del manejo de una consistencia mínima del AV, la cual permitirá una EAV más natural, repercutiendo en la QoE que se desea brindar al usuario final.

Como se mencionó anteriormente, el tipo de aplicaciones que nos interesan son las clasificadas como en Tiempo Real. En [**9**] se menciona que es posible clasificar las aplicaciones en tiempo real como tolerantes o intolerantes, dependiendo de si es posible tolerar la pérdida ocasional de los datos enviados. Se debe tener en cuenta que muchas aplicaciones en tiempo real son más tolerantes a la pérdida ocasional de datos que las aplicaciones que no son en tiempo real. Un tipo de aplicación en tiempo real es la *"adaptativa al retardo"* que permiten que los datos lleguen con un retardo aceptable. En los AVs donde participan n usuarios localizados en diferentes áreas geográficas y que su evolución se realiza en TR, el AV puede ser adaptable al retardo en tanto la QoE sea aceptable. Es decir que el tipo de aplicación que estudiamos en este trabajo es Tiempo Real y adaptativa al retardo.

La EAV distribuida se efectúa en diferentes PVs distribuidos y ejecutados en nodos que están intercomunicados por una infraestructura de red. Para mantener un AV consistente entre los PVs es necesario distribuir los cambios del AV entre los nodos que brindan como servicio un PV, de esta forma, los PVs toman en cuenta los cambios y realizan la representación correspondiente del AV. Para dar soporte a la distribución es necesario definir una arquitectura del sistema, las opciones consideradas son las siguientes [**10**]:

- ***Arquitecturas cliente-servidor:*** en esta aproximación, el sistema puede ser visto como un conjunto de servicios que se proporcionan a los clientes, los cuales hacen uso de dichos servicios. Los servidores y los clientes se tratan de forma diferente en estos sistemas.

- ***Arquitecturas de objetos distribuidos:*** en este caso, no hay distinción entre los servidores y clientes. El sistema puede ser visto como un conjunto de objetos que

interactúan, su localización es irrelevante. No hay distinción entre un proveedor de servicios y el usuario de estos servicios.

Con el uso de la arquitectura de objetos distribuidos, se minimizaría la dependencia hacia un PV. Es decir, los PVs pueden desconectarse voluntaria e involuntariamente del AV donde participan. Para cualquiera de estos eventos, es necesario el uso de mecanismos que permitan la adaptabilidad de la arquitectura del sistema y la transparencia de dicho evento en el AV. De esta manera, en la representación del AV se limita la dependencia hacia cualquier PV.

El manejo de la información que es definida como: *"la administración de los mensajes que se deben enviar entre los PVs que participan en la EAV"*, es un aspecto clave para evitar afectar el rendimiento efectivo de los canales de comunicación, cuando el AV es distribuido. Una de las técnicas utilizadas para el manejo de la información es la clasificación basada en prioridades de los datos a transmitir. En función de la clasificación, se establece el tipo de conexión de la fiabilidad necesaria [**11**]. De esta forma, se podría reducir el costo (en cuanto a tiempo, recursos, etc.) del envió y generar una correcta administración de los aspectos de red.

La distribución de la evolución de un AV involucra diversos aspectos desde la forma de representar la información o cambios del AV (capa de aplicación de acuerdo al modelo OSI) hasta aspectos a nivel de red (considerados en la capa de red del modelo OSI). Actualmente, existen investigaciones enfocadas en el diseño de cruce de capas (Cross-Layer Design CLD). CLD propone *"el diseño de protocolos que violan la dependencia entre las diferentes capas del modelo OSI, obteniendo una ganancia en el desempeño"* [**2**]. El paradigma CLD considera los efectos de las capas superiores hacia las capas inferiores del modelo OSI, con la finalidad de administrar la información que pasa por ellas. Lo anterior es conocido como calibración vertical entre capas. Uno de los puntos a tratar en la distribución de PV es el diseño de un mecanismo que considere el CLD para facilitar la distribución de la EAV en tiempo real, lo cual influye en la QoE que se desea brindar.

Cuando la comunicación entre los nodos que ejecutan los PVs de un AV se realiza a través de diferentes medios de comunicación (como Ethernet o Wireless), la distribución de la EAV se vuelve una tarea no trivial, afectando la consistencia del AV desplegado en cada PV. Al transmitir por diferentes canales de comunicación, se presentan variaciones en ciertos aspectos a nivel de red como lo son: el retardo, la pérdida de paquetes, el jitter, ancho de banda, tiempo de inter-partida, etc. Dichos aspectos son considerados por la Calidad de Servicios (Quality of Service - QoS) analizados en diversas investigaciones. Existen trabajos enfocados en transmisiones en tiempo real para sistemas multimedia que analizan el impacto de los aspectos mencionados. Estas investigaciones son analizadas para verificar si cumplen con los requerimientos para ser aplicados en los AVs e incrementar el grado de aceptación de la QoE. La administración de la consistencia entre los PVs debe permitir un rango aceptable de desfase en EAV, para lo cual es necesario realizar sincronizaciones entre los PVs del AV. La consistencia entre usuarios (PVs) es necesaria para mantener un ambiente consistente con una evolución en TR.

1.3 Motivación

La motivación de este trabajo de investigación radica principalmente en los siguientes puntos:

- El tipo de aplicaciones de realidad virtual consideradas por la plataforma GeDA·3D requiere la representación gráfica de la evolución distribuida de un AV y en la revisión del estado del arte no se encontró un trabajo que describa las tareas necesarias para una visualización distribuida coherente.

  Un *PV* no está ligado a la lógica del *AV*, por lo cual este podría ser utilizado para diferentes representaciones, ya sea en una o varias áreas del conocimiento.

1.4 Objetivos

Los objetivos principales de este trabajo de investigación son los siguientes:

I. Representar coherentemente de cualquier tipo de *AV* distribuida.
II. Analizar las características necesarias de la red para poder realizar la correcta representación en 3D de los cambios en los *AVs*, considerando la variedad de elementos utilizados en ellos.
III. Administrar gráficamente la evolución de los *AVs* mediante el uso de *PVs*.
IV. Identificar las características necesarias a nivel de red que permitan una *QoE* aceptable, dichas características deben considerar tanto los aspectos a nivel de aplicación, como a nivel de red. Estos permitirán aprovechar los beneficios de un diseño por cruce de capas o *CLD*.
V. Identificar y clasificar la información necesaria para generar la *EAV*. Esta información será útil para generar los mecanismos necesarios para realizar la representación gráfica de *AVs*.
VI. Diseñar mecanismos que permitan la sincronización de los *PVs* para mantener una consistencia entre ellos. Aun cuando un *AV* sea administrado por varios *PVs* en diferentes redes heterogéneas, la sincronización de los *PVs* debe brindar un grado aceptable de consistencia entre los *PVs* del *AV*.
VII. Diseñar un mecanismo que permita la administración de los aspectos de la red. Con lo cual se permita identificar el impacto que tienen en la *EAVs*. Con la finalidad de identificar cuál de los aspectos de la red (retardo, jitter, ancho de banda, etc.) repercute más en la distribución de los *AVs*.
VIII. Analizar las características que permitan mantener un grado aceptable de independencia entre los *PVs*, con la finalidad de que aun cuando se tengan desconexiones en el *AV*, se pueda continuar con su evolución.

IX. Diseñar un algoritmo para la adaptabilidad de la arquitectura del sistema de los PVs, tomando como base tanto las características de los AVs, como las condiciones de los nodos involucrados. Dicha adaptabilidad no debe afectar la $EAVs$, sino ser transparente para el usuario final, lo cual permitiría mejorar la QoE.

1.5 Estructura del documento

Las aportaciones de esta investigación estas descritas a partir del capítulo 3. A continuación se describen brevemente cada capítulo:

- ***Capítulo 2***: ofrece un análisis de los trabajos relacionados que resultan útiles para el desarrollo de la investigación presentada en este documento.
- ***Capítulo 3:*** se detallan los elementos de un AV y su interacción. Se define la estructura de un humano virtual, en base al estándar $H\text{-}Anim$. Se definen los tipos de animaciones aplicables a un avatar humano. En base a las características encontradas en los trabajos analizados, se realizó una selección las cuales se consideran útiles para resolver el problema planteado en este trabajo. De esta manera, se proponen nuevas características que son necesarias para la representación gráfica en 3D de un Ambiente Virtual.

  Capítulo 4: debido a que un usuario interactúa directamente con el PV es necesario brindar una QoE aceptable. Las métricas de la QoE son descritas en este capítulo.

  Capítulo 5: un AV es administrado por n PVs siendo necesario un manejo de la consistencia entre cada PV. En este capítulo se describe las pruebas realizadas a los PV con la finalidad de medir su consistencia.
- ***Capítulo 6:*** la arquitectura de red es adaptativa, para minimizar los retardos y por ende la inconsistencia en el AV. Los algoritmos que permiten la auto-organización de los nodos es descrita en este capítulo.
- ***Capítulo 7:*** se realizó la formalización de la arquitectura P_2P propuesta utilizando Algebras de Procesos. Las funcionalidades de la arquitectura y su formalización por medio de procesos son descritas en este capítulo.

  Capítulo 8: este capítulo muestra las conclusiones obtenidas de este trabajo de investigación, se describen las aportaciones y el trabajo futuro.
- ***Anexo A:*** contiene el análisis de los motores 3D y de físicas, así como la elección de los motores utilizados para la implementación de la propuesta.
- ***Anexo B***: se describen brevemente las librerías creadas en Havok para el manejo de las físicas.
- ***Anexo C***: análisis de algoritmos utilizados en motores 3D y de render, útiles para la agrupación de los elementos del AV.
- ***Anexo D***: muestra los pasos necesarios para la creación de terrenos irregulares, así como la edición de objetos 3D aceptables por el motor 3D elegido.

Capítulo 2 Trabajos relacionados a la Visualización Distribuida de Ambiente Virtuales

Objetivos:

- Describir los trabajos analizados que se consideran de utilidad para esta investigación.
- Desglosar las características relevantes encontradas.

Este trabajo aborda el problema de la visualización distribuida y coherente de AVs. Uno de los objetivos primordiales es facilitar la representación de diferentes tipos de escenarios 3D. Los trabajos analizados en este estado del arte fueron clasificados considerando los siguientes tres aspectos fundamentales en la distribución de los Puertos Visuales:

Nivel de usuario: Se revisaron trabajos que analizan las métricas de QoE en los AV, para enseguida identificar cuáles deben ser incorporadas en el diseño del PV.

Funcionalidad de un Puerto Visual: El análisis de estos trabajos relacionados se realizó para determinar las características a considerar para realizar la visualización coherente y en tiempo real de la evolución de un AV.

- ***Organización y comunicación de los Puertos Visuales:*** Analizar los trabajos relacionados para identificar características que ayuden a diseñar una arquitectura para la distribución de los PVs, minimizando la dependencia entre ellos.

2.1 Nivel de Usuario

Son pocos los trabajos relacionados con la visualización distribuida de AVs que consideren la QoE. La mayoría de los trabajos se enfocan en la lógica del AV, sin considerar el grado de satisfacción de los usuarios finales. En [**12**] se describe una clasificación de las métricas que influyen en la experiencia del usuario. Las métricas mencionadas están basadas en las características de un AV que incorpora hardware especializado (como, captores de movimiento, lentes, etc.), es decir, en un AV inmerso. La experiencia del usuario es dividida en las siguientes métricas: percepción, calidad de representación, psicológica y fisiológica.

En [**13**] y [**14**] se definen métricas para la QoS y la QoE, así como su relación. Las principales métricas en QoS son: interactividad, vivacidad y consistencia. Para la QoE se consideran: la percepción cognitiva y las consecuencias del comportamiento. En [**14**] se muestran los efectos del retardo en diferentes métricas de la QoE, demostrando que entre más grande sea el retardo menor será la QoE. En los dos trabajos se utiliza un AV inmerso, enfocándose en la tele presencia.

Las características de QoE mencionadas en estos trabajos son analizadas, para identificar las más relevantes en los AV, considerando que la representación de la EAV debe ser en tiempo real en los diferentes PVs participantes.

2.2 Funcionalidad de un Puerto Visual

Existen diferentes áreas de investigación que muestran las características que deben ser consideradas para satisfacer los requerimientos de un *PV*. Nosotros estamos interesados en las características que indiquen: a) cómo realizar un *PV* (módulo de la Vista, según el patrón *MVC*), b) los tipos de elementos de un *AV*, así como sus limitantes. Esto es necesario para generar mecanismos para la adecuada administración de dichos elementos del *AV*.

Los trabajos que deben ser revisados para definir estas características son los relacionados a: Ambientes Virtuales Distribuidos, Render, Simuladores y Video Juegos.

2.2.1 Ambientes Virtuales Distribuidos

Es necesario recordar que nosotros estamos interesados en *AV* distribuidos, esto es, ambientes en los cuales participan diferentes usuarios distribuidos geográficamente y por lo tanto los *PVs* están también distribuidos geográficamente.

En un *AV* distribuido la interacción de los usuarios es un requisito básico [**15**], para dar la impresión de interacción en tiempo real entre ellos es necesario disponer de un nivel de persistencia y coherencia satisfactorios [**16**], [**17**]. Estos niveles de persistencia y coherencia pueden alcanzarse mediante la sincronización de las acciones del usuario y el comportamiento del *AV*. Dado que los cambios en el *AV* son consecuencia del envío de mensajes entre los usuarios, es necesario elegir una arquitectura de red que tome en cuenta factores como lo son el ancho de banda, el retardo de los mensajes para asegurar la consistencia y coherencia [**18**]. El caso que nos interesa es cuando la comunicación entre los *PVs* se realiza en Internet. Ahora bien, Internet permite la participación de diferentes usuarios, pero entre otras cosas, es imposible asegurar su capacidad para mantener activos los servicios requeridos y de esta manera asegurar la interacción coherente entre usuarios participantes. Esto es debido a que Internet es una red de máximo esfuerza por lo que existe una inestabilidad en su latencia y el ancho de banda es variable. Estas características hacen imposible que bajo las condiciones normales mantener un AV consistente entre los usuarios participantes. En los *AVs* es más importante la comunicación de mensajes de actualización correspondientes a las entidades virtuales (avatares y objetos 3D), y menos importante la comunicación de los mensajes de colaboración entre los usuarios. Lo anterior, es debido a que las entidades virtuales generan mensajes de actualización con mayor frecuencia que los usuarios involucrados.

En [**19**], se sugiere la distribución de los elementos que constituyen el *AV*, por que juega un papel importante en la generación del tráfico de red, lo cual repercute en la escalabilidad del sistema. Cuando es generado un gran tráfico, no es posible soportar un gran número de usuarios. De la misma manera, en este trabajo se sugiere que los parámetros de la red que deben considerarse en la distribución de un *AV* [**19**], incluyen el ancho de banda, la latencia y la potencia de cálculo.

En [20] se demuestra que para un AV es posible establecer la comunicación basándose en una estructura cliente-servidor con una visualización en tiempo real, esto sólo para un determinado número de participantes. En esta investigación los participantes generan las actualizaciones que son enviadas al servidor para ser procesadas. Sin embargo, se sigue dependiendo de las capacidades del servidor. En esta misma investigación se presenta un sistema donde las entidades son agrupadas de acuerdo a sus actualizaciones. En este sistema, se dispone de una red de servidores que administran la comunicación entre los usuarios, siendo los servidores quienes procesan el nuevo estado de los AVs.

En [21] se presenta el sistema Spline, diseñado para realizar la división de un AV, con el objetivo de disminuir la cantidad de mensajes a procesar. El AV es administrado por una de red de servidores, un servidor central administra las conexiones y desconexiones de los usuarios, lo cual crea una dependencia de este servidor central. Al enviar y recibir los paquetes se obtiene información que utiliza un mecanismo de sincronización para determinar la secuencia de ejecución de las acciones de los usuarios. Con esto, es posible realizar una distinción entre los paquetes según su importancia, de esta forma se puede saber en cuáles casos si puede ser aceptada la perdida de paquetes. Es mencionado que en los AV donde se desea la ejecución en tiempo real, se debe sacrificar un poco la consistencia del estado del AV entre los usuarios. Finalmente, cabe mencionar que el sistema Spline fue diseñado para ser ejecutado en máquinas SGI para tener una mejor representación visual y un buen desempeño computacional. Es decir, existe una dependencia del proyecto con esta arquitectura específica.

En [22], se describe una arquitectura Peer-To-Peer (Igual-a-Igual) en donde se propone un algoritmo para la agrupación de entidades virtuales, basándose en la distancia que se tiene entre los usuarios del AV. El principal inconveniente de esta propuesta es que las divisiones hechas para el AV no toman en cuenta las características de cada entidad, lo cual provoca que la evolución del mismo no se ajuste a la vida real.

En [23] se utiliza el lenguaje de programación $VRML_{97}$ para el diseño del AV. La consistencia entre los usuarios es administrada por dos servidores, provocando una dependencia de ellos al momento de generar la evolución de los AVs.

En [24] se define la relación entre el retardo y la consistencia de un sistema interactivo distribuido. La inconsistencia generada por un retardo grande es analizada en un escenario donde las acciones no tienen un final determinado, es decir, se indica avanzar, pero no avanzar durante un tiempo determinado o avanzar cierta distancia. En los AVs donde se indica la finalización de la acción (por ejemplo: avanza 3 pasos, avanza 1 metro, etc.) no sería posible considerar los retardos descritos, por lo cual las aplicaciones del sistema son limitadas.

Existen diferentes paradigmas que permiten la incorporación de elementos inteligentes, tales como algoritmos genéticos [25] [26], redes neuronales [27], ontologías [28], etc. El uso de elementos inteligentes tiene la finalidad de dar realismo a los AVs y así obtener resultados más

apegados a la realidad. Sin embargo estos trabajos están enfocados en la lógica del AV y no en su visualización.

2.2.2 Render

En [29] se muestra un sistema de renderizado distribuido de AVs. Este sistema es una aplicación distribuida basada en una arquitectura cliente-servidor, donde la evolución de un AV se distribuye entre los servidores disponibles. Sin embargo, la capacidad de cada servidor limita la cantidad de clientes concurrentes en el AV.

En [30] Strasser et al presentan un sistema de visualización interactivo basado en el perfeccionamiento progresivo y renderizado distribuido. Este sistema es capaz de navegar en múltiples conjuntos de datos de gran resolución a través del uso de la caché de imágenes. Este trabajo no es útil para los fines buscados en el trabajo de investigación expuesto en este documento, porque la salida del proceso de representación es una imagen. Con lo cual, es necesario generar varias imágenes por segundo para visualizar la evolución completa de los escenarios 3D. Otro problema en este tipo de trabajos es que existe un servidor encargado de componer la imagen final. Así, si el servidor falla, no sería posible generar la imagen final y el proceso de representación sería incompleto.

En [31], se propone un sistema de renderizado 3D que distribuye la tarea de representación a través de una plataforma multi-agente. La idea central es la representación de objetos 3D en diferentes equipos. Esta tarea se resuelve mediante un mecanismo basado en colas, donde cada agente remoto se asocia con una cola en la que almacena la imagen renderizada. Así, el agente que genera la visualización en 3D combina la información de los diferentes frentes de cada cola para producir una visualización centralizada. Aun cuando las tareas son distribuidas, un solo agente es el encargado de realizar la visualización, generando dependencia hacia él.

En [32] se implementó un sistema de renderizado remoto a través de un componente de colaboración y un componente de representación remota. La tarea de representación se distribuye en un conjunto de nodos. Las geometrías son asignadas a diversos nodos con la capacidad de procesamiento requerida, con el objetivo de mejorar el rendimiento del procesamiento de la resolución de la imagen final. El componente de representación remota es el encargado de recabar las geometrías en una sola representación generando una visualización centralizada.

En las aplicaciones que trabajan con gráficos 3D es necesario el uso de un motor 3D eficiente para mostrar los diferentes escenarios 3D y su evolución en tiempo real. Una alternativa es el uso de sistemas de renderizado paralelo. Existen dos formas de aplicar estos sistemas: a) utilizando una dispositivo con un sofisticado y potente procesador gráfico [33], b) utilizar grupos de red de computadoras con capacidad de procesamiento gráfico [34]. El principal problema a resolver en estos sistemas es la creación de algoritmos eficientes que maximicen la capacidad del

procesamiento gráfico, el almacenamiento, el manejo de la información, características de comunicación entre grupos de computadoras y los recursos de la red que está siendo utilizada. Este tipo de sistemas requieren de un alto procesamiento de gráficos, limitando su uso a dispositivos especiales, en nuestro caso es deseable el uso de dispositivos de uso frecuente con características de procesamiento grafico habitual.

El algoritmo llamado *"geometry tracking"* reduce el ancho de banda en un sistema de renderizado paralelo y distribuido, propuesto en [**35**]. La base del algoritmo es eliminar la retransmisión de datos redundantes, debido a la indexación de datos en un paquete de red. Otro trabajo en esta dirección es presentado en [**36**] donde se describe un sistema capaz de realizar una partición dinámica y coordinada de objetos 3D y 2D en regiones. Para equilibrar la carga de representación entre las computadoras y reducir al mínimo el ancho de banda necesarios para la composición de imágenes, se solapan regiones que son combinadas para generar la imagen final. La imagen final es generada por una computadora que debe distribuirla a los interesados.

En [**37**] se describen dos arquitecturas para la optimización del renderizado distribuido: Yafrid (Yeah! Un render de Libre GRID) y MagArRO (Multi Agente Approach to Rendering Optimization). Los cuales utilizan las capacidades que ofrece el uso de una malla (grid) de computadoras, con la finalidad de generar imágenes estáticas. Al generar imágenes estáticas es necesario contar con un procesamiento grafico alto para generar la cantidad de imágenes necesarias para dar el aspecto de Tiempo Real.

En [**38**] se presenta un estudio sobre la asignación de recursos, la partición de tareas y gestión de datos, los cuales se implementan en la construcción de una representación distribuida de grandes conjuntos de datos. También se describe una aplicación del algoritmo Dynamic Pixel Bucket Partition ($DPBP$). El algoritmo es utilizado para efectuar la asignación de tareas de aplicaciones de renderizado distribuido entre los nodos del sistema. Con esta aplicación los autores prueban que el renderizado puede llevarse a cabo casi en tiempo real. Este trabajo muestra que la representación distribuida casi en tiempo real puede llevarse a cabo al reducir las tareas en los nodos. La calidad del rendereo depende de los nodos del sistema, por lo cual para tener mejor calidad es necesario nodos especiales para el procesamiento de graficos.

2.2.3 Simuladores

Los simuladores son utilizados para mostrar a una audiencia algún fenómeno, para estudiar un comportamiento, entrenar a un grupo objetivo, etc. Una de las aplicaciones más importantes de la simulación es el entrenamiento militar y el médico. La implementación de algunos de estos es centralizada. En los que existen n participantes se utiliza la estructura cliente-servidor. En ciertos casos se podría llegar a tomar a un simulador como un juego de vídeo o viceversa.

Existen diversas investigaciones para hacer distribuida la implantación de simuladores, una de ellas es mostrada en [**39**], en la cual se realiza una comparación entre DIS (Distributed

Interactive Simulation) y *HLA* (High Level Architecture). *DIS* cumple con lo requerido para realizar simulaciones de tipo militar, mientras que *HLA* está diseñado para cualquier tipo de simulación. *HLA* realiza una agrupación de los miembros de acuerdo al tipo de avatar que manipula (a todos los aviones los agrupa, a los carros los agrupa). Los mensajes son enviados por UDP/IP en *DIS* y por TCP/IP en *HLA*. Tanto en *DIS* como en *HLA* se considera la latencia, pero se da un mayor enfoque en *HLA*, esto debido a que su comunicación es orientada a la conexión. La agrupación realizada en *HLA* establece comunicaciones innecesarias que generan más tráfico en la red.

Como se ha mencionado, una de las técnicas utilizadas para la reducción de mensajes es el agrupamiento de los participantes. Uno de los trabajos realizados en esta dirección es [**40**]. Donde se prueba que es factible enviar los mensajes con una frecuencia establecida, ya que de esta forma se puede detectar si un nodo sigue activo, pero se incrementa el número de mensajes en la red.

Los trabajos realizados sobre simuladores muestran que es factible la agrupación de entidades de la simulación, para reducir la cantidad de mensajes enviados entre los nodos participantes.

2.2.4 Vídeo juegos

Los vídeos juegos son un área de investigación muy activa. El desarrollo e investigación de video juegos está fuertemente influenciada por la utilización de la tecnología utilizada en los *AVs* [**41**]. Dos aspectos importantes a considerar en esta área son la facilidad de permitir la interacción de múltiples usuarios, así como el despliegue correcto de las acciones de los jugadores.

Sin embargo, dadas las características de los video juegos, la mayoría de las implementaciones de estos utilizan una arquitectura cliente-servidor [**42**] [**43**] [**44**] [**45**] [**46**]. En estos juegos, el usuario puede manipular los avatares, pero las acciones del usuario no se realizan en la aplicación de usuario [**47**], en lugar de esto, la aplicación del usuario envía mensajes de actualización al servidor y este se encarga de calcular la evolución del juego. Por lo tanto, el servidor suele representar un cuello de botella. Existen algunos juegos que utilizan una arquitectura distribuida [**48**], pero tienen un comportamiento simple y el conjunto de acciones del usuario es limitado [**49**].

Por otro lado, el área de los videos juegos ha hecho un estudio exhaustivo del tráfico de la red. En [**50**] se analizan juegos de red mostrando que los paquetes son generados periódicamente, lo cual permite predecir el comportamiento del tráfico entre el usuario y el servidor. Esto es porque los cambios generados por el usuario se envían al servidor en periodos de tiempo constante con la finalidad de mantener la lógica del juego. Para esto, se consideran paquetes pequeños enviados frecuentemente por UDP/IP. Los mensajes son procesados en el servidor para después enviar el nuevo estado del juego a los usuarios. El análisis muestra que para un conjunto

fijo de jugadores el tráfico se vuelve estable y es predecible. Esta propuesta crea una dependencia hacia el servidor, ya que este es quien administra los cambios en el video juego.

Otras investigaciones modelan el tráfico generado utilizando alguna distribución [43] [44] [45] [46]. Estas propuestas también analizan el caso de en una arquitectura cliente-servidor que se utiliza en diferentes tipos de video juegos. En [51] el tráfico de los clientes está separado del tráfico del servidor, mostrando que sus comportamientos son diferentes, ya que el tráfico generado por el servidor depende del número de clientes concurrentes. También el trabajo [52] analiza el tráfico haciendo una separación del mismo basada en su origen. Este trabajo muestra que el tamaño de los paquetes del servidor es más grande que los paquetes del cliente. Sin embargo, estos trabajos no son una buena alternativa a utilizar debido a que se basan en una arquitectura cliente-servidor, creando una dependencia en el servidor como componente centralizado.

C. Diot y L. Gautier en [53] detectaron que los usuarios realizan un envío constante del estado de los elementos, siendo cada usuario el encargado de procesar la información. Para ello, se utiliza un reloj global para la sincronización de mensajes, pero no se toma en cuenta la consistencia de la aplicación. Los paquetes son enviados por UDP/IP, los cuales contienen mensajes con formatos especificados en *DIS*. En [54] [44] se analiza el tráfico generado por vídeo juegos, los cuales utilizan una estructura cliente-servidor. La comunicación se realiza por medio de un canal no fiable (UDP), ya que se considera que los mensajes generados tienen una frecuencia alta. Una de las ventajas de generar mensajes constantemente es detectar si los jugadores siguen activos, la desventaja más significativa es el uso excesivo de la red que podría generar grandes retardos y/o perdida de mensajes, entre otras tantas posibilidades.

Otro mecanismo interesante de conexión de usuarios se muestra en [55], donde se utiliza un servidor de conexiones que utiliza un canal de comunicación fiable (TCP/IP). Una vez realizada la integración de los nuevos miembros en el sistema se realiza una sincronización de relojes por medio del protocolo *NTP* (Network Time Protocol - *NTP*). Los mensajes generados después de la conexión son enviados vía UDP/IP. Esta investigación, a diferencia del resto de las investigaciones de video juegos analizadas muestra la importancia de la sincronización de los usuarios para realizar los cambios solicitados por los jugadores en el orden en que fueron generados.

El juego StarCraft fue analizado en [56], el cual utiliza una estructura cliente-servidor, los mensajes son enviados por *IPX* (Internetwork Packet Exchange - *IPX*), la frecuencia en los envíos de mensajes depende del número de jugadores. Otro de los juegos analizados es QuakeWord, el cual distribuye los mensajes con una frecuencia independiente de la cantidad de jugadores involucrados. Una desventaja mostrada es cuando la frecuencia en el envió de los mensajes se incrementa o cuando es provocada por el aumento del número de jugadores, impidiendo tener una mayor escalabilidad, debido a la posible saturación de la red.

Una investigación que realiza una recopilación de las características de los vídeo juegos en red es [11]. Del cual cabe destacar el desglose de los mensajes, englobándolos en dos categorías principales:

- ***Interna:*** los mensajes no están ligados a mensajes anteriores, útil en aplicaciones que trabajan con protocolos no fiables.

- ***Externa:*** se puede utilizar información ya transmitida, lo cual es implementado en sistemas con protocolos fiables.

Otra métrica usada es la fusión de varios mensajes en un solo paquete que puede reducir el tráfico en la red. Esta fusión puede ser en un determinado tiempo o cuando se tenga cierta cantidad de mensajes. Algunos de los trabajos analizados dividen el ambiente de acuerdo a áreas de interés (auras), para que un nodo envíe mensajes a otro se deben interceptar sus auras. Las auras son delimitadas utilizando figuras geométricas (comúnmente círculos), su desventaja es que aun cuando dentro de un aura dos elementos no requieran comunicación se realizan envíos de mensajes entre dichos elementos.

2.3 Organización y comunicación de los Puertos Visuales

En esta sección se hace un análisis de las investigaciones relacionadas con algoritmos para la organización de nodos. Proponemos identificar características que permitan el diseño de un algoritmo que tome ventaja de la auto-organización. Es decir, deseamos utilizar los principios de la auto-organización para organizar los nodos donde se ejecutan los PVs de un AV. Nuestro objetivo al usar la auto-organización es minimizar los requisitos de cómputo para no afectar la visualización y la QoE del sistema global.

2.3.1 Mecanismos y protocolos

Cuando se utilizan mecanismos para la organización de usuarios, se puede ofrecer una mejor administración en la comunicación. La forma de identificar a cada integrante puede ser por medio de identificadores binarios, los cuales son otorgados mediante el código Gray [57], a su vez pueden ser organizados en árboles bajo diferentes conceptos (repetidores, distancia física, etc.).

En [58] se presenta un protocolo que utiliza un repositorio de nodos para realizar la distribución de mensajes entre ellos. El protocolo es genérico con la finalidad de ser utilizado en diversas aplicaciones. Los datos son cifrados y fraccionados para poder ser enviados. Después de ser enviados, son almacenados en un buffer por si fuera necesaria la retransmisión. Los nodos son sincronizados para colocar una marca de tiempo en los mensajes, utilizada para identificar la secuencia de llegada. Los mensajes son enviados por UDP/IP porque no es importante su pérdida,

debido a que el servidor envía constantemente actualizaciones. Cuando un protocolo necesita enviar mensajes de forma fiable se utiliza TCP/IP.

Para obtener una distribución aceptable de los mensajes es posible organizar los nodos en un árbol [**59**] de acuerdo a las distancias entre los participantes. Para esto, se considera el *TTL* (Time To Live). Ahora bien, para mantener dicha estructura, se envía constantemente el estado de los nodos. Los nodos se clasifican en: raíz (encargado del procesamiento central de la información), repetidores (utilizados para evitar la pérdida de mensajes debido a la distancia) y emisores/receptores (quieres generan el tráfico que debe llegar a la raíz). La forma de detectar si un nodo sigue activo es recibiendo periódicamente mensajes del mismo, si en algún momento se dejan de recibir dichos mensajes, se considera que el nodo dejó el grupo por lo cual se debe ajustar el árbol.

El protocolo utilizado para multitransmisiones (multicast) en internet es *MBONE* [**60**], es soportado por algunos encaminadores (routers), ya que no es obligatorio en *IPv4*, pero sí en *IPv6*. En nuestro caso este protocolo pudiera ser una opción ya que su uso reduce el trabajo de la aplicación que lo implementa. Cuando un usuario desea publicar información, envía el mensaje a la dirección del grupo deseado. Aun cuando ofrece ventajas la aplicación de dicho protocolo, existe una desventaja que puede restarle eficacia, esta desventaja se debe a que para el envío de los mensajes se utiliza el protocolo UPD/IP. De esta manera, los mensajes se envían con irregularidad (cada que un cliente lo solicite) haciendo que el remitente no tenga control o conocimiento implícito de lo que existe en la red.

La clasificación de nodos y el considerar la distancia entre un conjunto de nodos son características que se pueden utilizar para el diseño de un mecanismo para la auto-organización de los mimos.

2.4 Conclusiones

Los trabajos hasta ahora analizados para la distribución de *PVs* han sido diseñados para la representación de una situación específica. Por ejemplo, en [**21**] se visualiza el comportamiento de ciclistas, otro ejemplo son los juegos en los cuales se tienen temas en especial (futbol, estrategias, etc.) [**45**] [**46**] [**56**]. Las características de los trabajos analizados impiden generar en tiempo de ejecución una nueva representación gráfica de un *AV*, es decir, la creación de su entorno no se realiza de forma dinámica, por lo cual el ambiente no evoluciona según las necesidades que surjan en tiempo real. Se detectaron diferentes características que pueden ayudar a la correcta distribución de los *PVs*, de forma consistente entre los usuarios participantes. Dichas características son:

- El uso de una arquitectura que limite la dependencia hacia un(os) *PV(s)*.

- La organización de los PVs de acuerdo a la posición del elemento del ambiente que administra.
- La clasificación de mensajes para la elección de canales de comunicación.
- La sincronización de los usuarios participantes.
  El procesamiento de las actualizaciones en base a las entidades virtuales.
- Tomar en consideración aspectos de la red como el ancho de banda, el retardo y/o la pérdida de mensajes, etc.
- Reducir el número de mensajes entre los usuarios.
- Organizar a los nodos del sistema de acuerdo a su posición geográfica.

Como se menciona en [**21**], si se desea la ejecución en TR, se debe sacrificar un poco la consistencia del estado del ambiente entre los usuarios. Por lo tanto, es necesario analizar las características con la finalidad de encontrar un equilibrio entre la visualización y la consistencia entre los usuarios.

Es necesario analizar que métricas de la QoS y QoE son relevantes en la visualización distribuida de un AV, es decir, que métricas afectarían el desempeño de la distribución de los PVs. Esto es necesario para realizar una correcta administración de los recursos tanto de la red como del AV.

Capítulo 3 Distribución de Puertos Visuales

Objetivos:

- Definir los elementos de un Ambiente Virtual, así como la interacción entre los diferentes elementos.
- Describir la estructura de un humano virtual en base al estándar $H\text{-}Anim$.
- Describir una arquitectura P_2P utilizada para la distribución de Puertos Visuales, los cuales permiten la visualización de la evolución de un Ambiente Virtual en Tiempo Real considerando la QoS y QoE satisfactorios.

3.1 Descripción de un Ambiente Virtual

Para realizar una distribución de un PV y cumplir con los objetivos requeridos es necesario conocer tanto los elementos de un AV como la forma en que interactúan, para realizar una administración adecuada y realizar una visualización en tiempo real. A continuación se describen los elementos de un AV y su interacción.

Sea α un AV, el cual está formado por un conjunto β de objetos 3D, un conjunto γ de avatares y un escenario ε:

$$\propto = \{ \varepsilon \cup \gamma \cup \beta \}$$

donde, ε es una representación gráfica en la cual sucede la evolución de un α $(Ev\ (\propto))$. Dicha Ev está dada por los cambios provocados por solicitudes (s) que contienen acciones (ac_i) de un avatar (a_j), el cual pertenece al α en cuestión. Un avatar representa una entidad capaz de reaccionar a la situación en la que está. Siendo un avatar una entidad autónoma con características y funcionalidades definidas, se considera que el avatar es el elemento más complejo del ambiente. Una solicitud s causa la evolución de un α en un tiempo (t_x):

$$s(ac_i, a_j) \xrightarrow{t_x} Ev(\alpha)$$

donde, la notación $\xrightarrow{t_x}$ representa el tiempo en el cual se realiza la solicitud.

De la misma forma, n objetos 3D $(\omega \mid \omega \subseteq \beta)$ que pertenecen a un α pueden evolucionar (ser modificado en forma y posición) como consecuencia de las acciones de un avatar, provocando una evolución, descrita de la manera siguiente:

$$s(ac_i, a_j) \xrightarrow{t_x} Ev(\omega)$$

El resultado de una acción puede modificar la apariencia y posición del mismo avatar y/o de algún otro, provocando su evolución en el α.

$$s(ac_i, a_j) \xrightarrow{t_x} Ev(a_j)$$
$$s(ac_i, a_j) \xrightarrow{t_x} Ev(a_f) \mid s(ac_i, a_j) \xrightarrow{t_x} Ev(a_j, a_f)$$

El avatar humano es considerado la entidad más compleja para su representación y administración en un AV, al cual solamente llamaremos avatar. Cada $a_i \in \gamma$ contiene un conjunto

de acciones validas (τ_i), las cuales son representadas gráficamente tomando en cuenta la personalidad (ρ_i) y el estado emocional (π_i) del avatar, como sigue:

$$a_i = \{\tau_i, \rho_i, \pi_i\}$$
$$\tau_i = \{ac_1, ac_2, \cdots, ac_n\}$$

donde, un estado emocional π está formado por un conjunto de seis emociones básicas:

$$\pi_i = \{enojo, disgusto, miedo, alegría, trizteza, sorpresa\}$$

Cuando es generada una Ev (α), esta da como resultado un nuevo estado del ambiente (EsA) en un tiempo (t_x):

$$Ev(\alpha) \overset{t_x}{\rightarrow} EsA$$

donde, el EsA generado es el resultado de la ejecución de una o varias solicitudes que contienen acciones y el avatar que debe realizarlas:

$$\{s(ac_k, a_l) \mid s(ac_5, a_1), s(ac_1, a_2), \cdots, s(ac_j, a_i)\} \overset{t_x}{\rightarrow} EsA$$

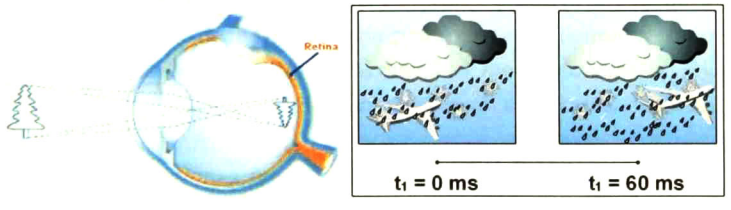
Las solicitudes son agrupadas en un proceso φ_i:

$$\varphi_i \overset{t_x}{\rightarrow} EsA$$

El tiempo de procesamiento (tp_r) de un proceso φ_i debe ser en tiempo real, es decir, la ejecución deberá estar acotada por un tiempo TR:

$$tp_r(\varphi_i) \leq TR$$

En este trabajo, TR se deriva de los siguientes conceptos relacionados con la vista humana [61]:



- ***Persistencia de la visión*** (PvD) [62]: una imagen permanece en la retina por un periodo de tiempo de 40 milisegundos.

***Movimiento phi* (*Mphi*)** [63]: el fenómeno o movimiento phi es un ilusión óptica en nuestro cerebro, que permite percibir un movimiento constante en lugar de una secuencia de imágenes. La resolución espacial y temporal de los objetos en las imágenes es óptima cuando se realizan los cambios de imagen (CI) en un periodo de 60 milisegundos:

$$\underline{\text{Si}}\ CI \gg Mphi\ \underline{\text{entonces}}$$
$$el\ movimiento\ es\ pausado$$

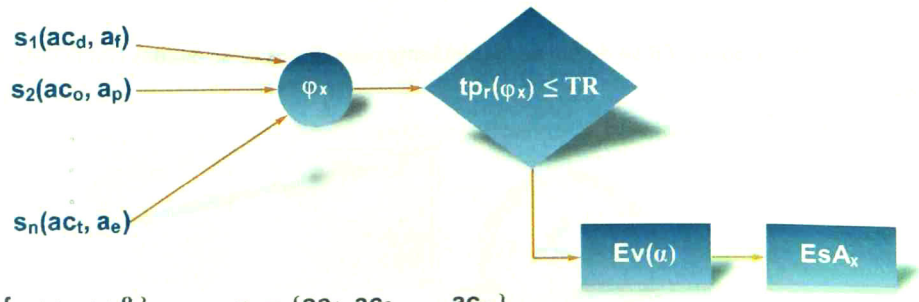
Por lo contrario,

$$\underline{\text{Si}}\ CI \ll PvD\ \underline{\text{entonces}}$$
$$el\ movimiento\ no\ es\ detectado$$

$$tp_r\ (CI) \leq 60\ milisegundos$$
$$\therefore$$
$$tp_r\ (\varphi_i) \leq 60\ milisegundos$$
$$tp_r\ (\varphi_i) \leq TR$$

Una secuencia correcta de sucesos para la evolución de un α se define como (ver **Figura 2**):

1. Se generan de 1 a n solicitudes que contienen acciones que deberán ser realizadas por algún avatar: $\{s(ac_i, a_j) \mid s(ac_1, a_3), s(ac_8, a_6) \cdots s(ac_k, a_l)\}$.
2. Las acciones son agrupadas en procesos que deben ser realizados en un tiempo menor o igual al definido como tiempo real: $tp_r\ (\varphi_i) \leq TR$.
3. La ejecución de un proceso φ_i genera la evolución del α.
4. Se genera un nuevo estado del ambiente virtual (EsA) por cada evolución del ambiente.



$\alpha = \{ \varepsilon \cup \gamma \cup \beta \}$ $\quad \tau_i = \{ ac_1, ac_2, \ldots, ac_n \}$

$\gamma = \{ a_1, a_2, \ldots, a_b \}$ $\quad a_i(\tau_1) \neq a_{i+1}(\tau_i)$

$a_i = \{ \tau, \rho, \pi \}$ $\quad ac_U(a_g) \neq ac_{U+1}(a_h)$

Figura 2: Evolución de un Ambiente Virtual.

Para permitir una correcta $Ev(\propto)$ se mencionaron diferentes características como: la ejecución de procesos en tiempo real, la animación de avatares, la secuencia de solicitudes, etc. Todas estas características son tratadas en esta disertación de tesis de doctorado.

3.2 Descripción de un avatar

Para lograr una animación coherente de un avatar es necesario que este tenga un esqueleto, el cual soporte las animaciones necesarias para el *AV*. Para la definición de un esqueleto humano se tomó como base el estándar *H-Anim* [**64**].

H-Anim es un estándar de representación de humanoides (avatares), su finalidad es permitir la manipulación de avatares en cualquier editor 3D. Estas entidades pueden ser animadas usando fotogramas claves, cinemática inversa, etc.

Se define un esqueleto humano (ver **Figura 3**) como un conjunto de segmentos (el antebrazo, la mano, el pie, etc.) los cuales están conectados entre sí por articulaciones (el codo, la muñeca, el tobillo, etc.). Para lograr una animación es necesario tener acceso a las articulaciones para cambiar sus propiedades.

El esqueleto del avatar contiene las articulaciones necesarias para realizar animaciones que permitan la representación lo más apegado al mundo real. Es posible clasificar las animaciones en dos grupos:

- ***Micro-animaciones:*** se manipulan de forma independiente las propiedades de un empalme del esqueleto. Por ejemplo, rotar el empalme de la muñeca, mover la articulación correspondiente a la rodilla.

- ***Macro-animaciones:*** representan secuencias de movimientos de los empalmes de un esqueleto. Es decir, son conjuntos de micro-animaciones que forman animaciones más complejas. Por ejemplo: caminar, saltar, saludar, etc.

- **Animaciones prediseñadas**: Un editor 3D genera animaciones prediseñadas por medio de la secuencia de cuadros que contienen micro-animaciones. Las *animaciones prediseñadas* son utilizadas hasta el momento por los *AVs* y los videos juegos, limitando las posibilidades de manipulación en tiempo real de un avatar.

- **Animaciones pre-calculadas**: este tipo de animación involucra el cálculo previo de cada movimiento que debe realizar un avatar, permitiendo tener un mejor control de los empalmes del esqueleto del avatar.

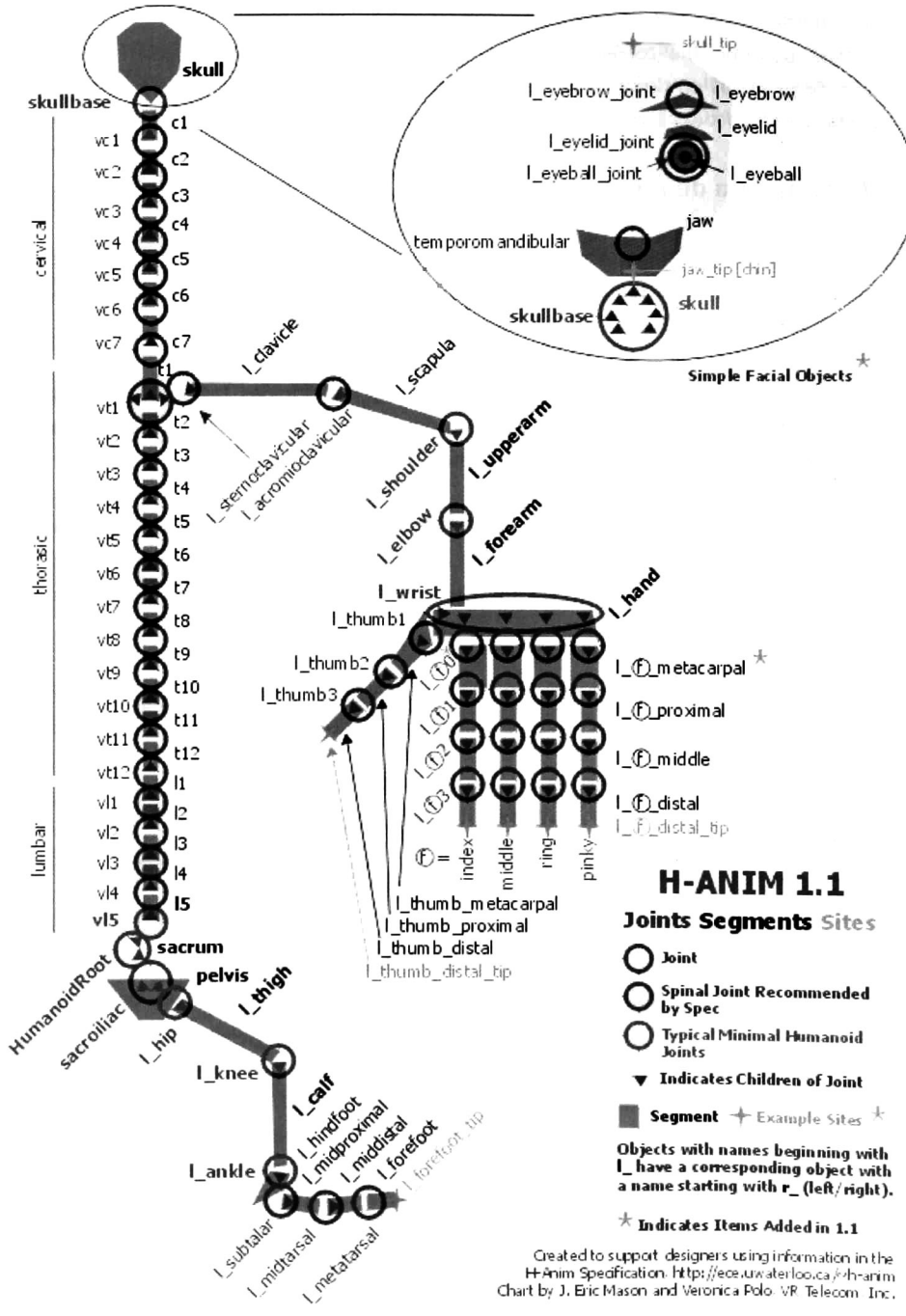


Figura 3: Esqueleto humano según el estándar *H-Anim*.

En las siguientes secciones se plantea y se describe nuestra solución propuesta al problema de visualización y representación gráfica distribuida de un AV 3D de cualquier tipo. Dicha visualización tiene que ser llevada a cabo por un conjunto de PVs que son ejecutados en nodos interconectados en su caso general mediante redes heterogéneas. La visualización llevada a cabo por los PVs debe realizarse considerando una QoE aceptable, esto debido a que los PVs son el medio de interacción del usuario con el AV.

Con base en lo citado en [**41**], es necesario especificar las características necesarias para llevar a cabo la visualización de la evolución de los AVs, los cuales son utilizados para la simulación de diversas situaciones de la vida real. Además, identificar los elementos que provocan la evolución del ambiente, así como el flujo de información, para lo cual se diseñó una arquitectura de referencia. Dicha arquitectura de referencia contempla desde el usuario final hasta la parte de la visualización de un AV.

3.3 Arquitectura de referencia QoE-PV

En [**65**] se menciona que para diseñar una arquitectura de software es posible tomar en cuenta varios conceptos, los cuales son:

- ***Patrón arquitectónico:*** es un conjunto de restricciones de los elementos e interacciones de las partes que forman una arquitectura. Un patrón arquitectónico no es una arquitectura, sino algo que transmite una imagen útil del sistema que impone restricciones útiles sobre la arquitectura y, a su vez, en el sistema.

- ***Modelo de referencia:*** división de funciones junto con el flujo de datos entre las piezas. Un modelo de referencia es una descomposición estándar de un problema. Derivados de la experiencia, los modelos de referencia son una característica de los dominios de madurez. Ejemplos: componentes estándar de un compilador, un sistema de gestión de bases de datos, etc.

- ***Arquitectura de referencia:*** Mientras que en un modelo de referencia se divide la funcionalidad, una arquitectura de referencia representa el mapeo de esa funcionalidad en una descomposición del sistema.

Los modelos de referencia, los patrones de arquitectura, y las arquitecturas de referencia no son arquitecturas de software, sino que son conceptos útiles el diseño de arquitecturas de las mismas. La relación entre estos elementos de diseño se muestra en la **Figura 4** (rectángulos en color azul).

En la **Figura 4** se muestran los pasos realizados (rectángulos en color naranja) para el diseño de la Arquitectura P_2P propuesta en la siguiente sección. Para ello, se tomó como base el patrón de arquitectura MVC para el diseño de la Arquitectura de referencia QoE-PV. El patrón

MVC contempla los módulos en los cuales un sistema puede ser dividido, pero no las expectativas del usuario final, es decir, la *QoE*.

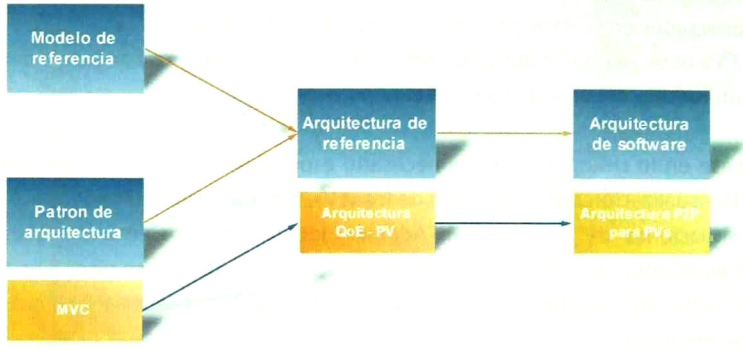


Figura 4: Conceptos considerados para el diseño de una arquitectura *P2P*.

La arquitectura de referencia *QoE-PV*, considera las expectativas del usuario final y su relación con la visualización del sistema, en nuestro caso el *PV*. De esta forma los sistemas que puedan compartir un mismo *PV* pero no la lógica del sistema pueden diseñar una mejor arquitectura de software. Un ejemplo son los IDE de programación como Visual Studio que a través de la misma interfaz permite implementar sistemas en diferentes lenguajes de programación (C#, C++, ASP, etc.). El IDE se encarga de identificar el lenguaje de programación utilizado para realizar la compilación del sistema.

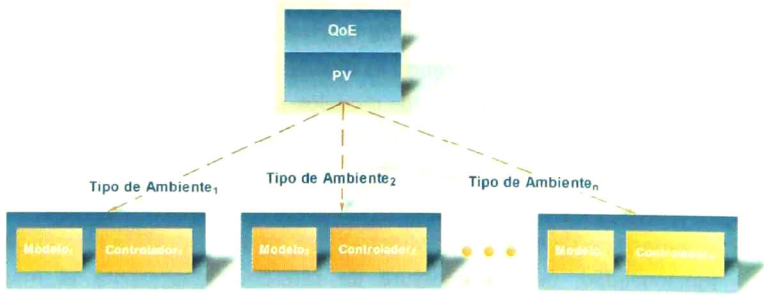


Figura 5: Arquitectura de referencia QoE - PV.

La **Figura 5** muestra la descomposición de *QoE – PV*, la cual permite compartir el *PV* entre diversas lógicas de *AVs*. De esta forma, es posible delimitar las funcionalidades que debe cumplir el *PV*, para lo cual se diseñó una arquitectura de software, descrita en la siguiente sección.

3.4 Arquitectura Peer-To-Peer para Puertos Visuales

Para llevar a cabo la representación de la EAV del AV se diseñó una arquitectura Peer-To-Peer (P_2P). En [**10**] se define los sistemas P2P de la siguiente manera: *"Los sistemas P_2P son sistemas descentralizados donde los cálculos pueden realizarse en cualquier nodo de la red y, al menos en un principio, no se hace distinción entre clientes y servidores"*. La arquitectura P_2P que se propone en esta sección, está dividida en capas como se muestra en la **Figura 6**. Esta separación facilita independizar el funcionamiento de las capas para poder realizar tareas en paralelo y agilizar la evolución de la representación de un AV.

La división en capas se realizó tomando en cuenta que los PVs de un AV deben mantener una consistencia local y global. Es decir, es necesario considerar las repercusiones de las solicitudes tanto localmente, como el impacto en el resto de los PVs del AV. Además es muy importante analizar la forma conveniente en que las solicitudes deben de ser enviadas al resto de los PVs del AV. Cada capa realiza una tarea específica como se menciona a continuación:

- ***Capa 3: Procesamiento local:*** despliega gráficamente en 3D las solicitudes del usuario, dando como resultado sus efectos sobre el AV, manteniendo una consistencia mínima local.

- ***Capa 2: Verificación de consistencia:*** es la encargada de administrar la consistencia entre los usuarios, es decir, verifica que todos los PVs tengan el mismo estado del AV.

- ***Capa 1: Distribución de actualizaciones:*** crea los mensajes a distribuir, decidiendo si serán enviados por un medio fiable o no fiable.

Figura 6: Arquitectura P_2P para la Visualización de un Ambiente Virtual.

Estas capas satisfacen los requerimientos para distribuir la EAV, entre diversos PVs. El uso de una arquitectura P_2P facilita una mayor escalabilidad y minimiza la dependencia entre los distintos PVs que participan en el AV distribuido.

De las secciones anteriores ahora, sabemos que la QoE no sólo involucra la capa de procesamiento local, sino además debe de ser considerado todo el proceso de actualización de la EAV, desde la interpretación de las solicitudes y la correcta ejecución de las acciones hasta la adecuada forma del manejo de la información. Esto quiere decir que cuando se requiere distribuir un mensaje, se deben tener en cuenta los recursos de la red para minimizar el costo de su envío. Los recursos de la red (RD) son descritos en [**18**], los cuales sintetizamos en la **Figura 7**.

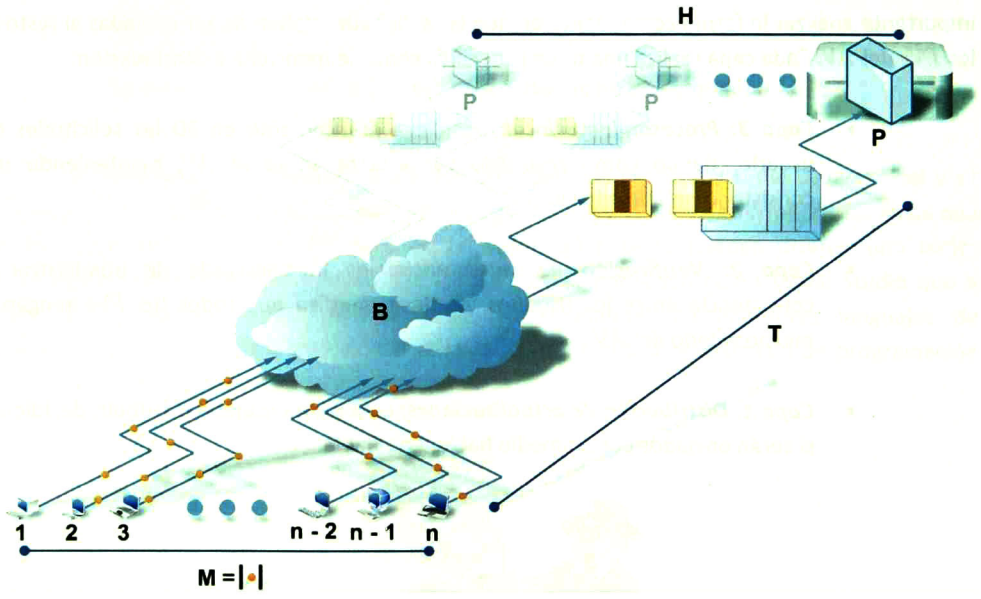


Figura 7: Recursos presentes en la red.

- M = número de mensajes transmitidos.
- H = promedio de nodos destino para cada mensaje.
- B = promedio del ancho de banda requerido para la comunicación.
- T = tiempo requerido para la entrega de paquetes. Entre mayor sea el tiempo, menor será el retardo.
- P número de ciclos de procesamiento requeridos para recibir y procesar los mensajes.

En [**18**] también se menciona la dependencia entre los recursos de red:

Si M aumenta entonces

B aumenta
Si H aumenta entonces
M y B aumentan
Si M disminuye entonces
B y P disminuyen y T aumenta

Cuando el parámetro M es modificado constantemente, son afectados negativamente varios recursos, es por eso que se considera que se tiene que tomar un enfoque distinto en la administración de los mensajes generados que se desean enviar, es decir, se debe proponer un nuevo mecanismo para el manejo de la información. Al considerar los recursos presentes en la red, se puede tener un mejor manejo de la información generada por los PVs y realizar así un CLD adecuado permitiendo que la QoE se mantenga en un rango aceptable para el usuario. El diseño del CLD puede permitir una visualización en tiempo real en cada PV del AV. Las siguientes secciones describen más a detalle la funcionalidad de cada capa de la arquitectura propuesta.

3.4.1 Capa 3: Procesamiento local

La capa de *Procesamiento Local* tiene como finalidad mostrarle al usuario, por medio de un PV, la evolución del AV en 3D. Las solicitudes generadas por el usuario son procesadas por el esta capa, al recibir una solicitud se identifica el avatar y la acción que debe realizar dicho avatar, posteriormente la acción es mostrada gráficamente al usuario. La ejecución de la acción podría generar un resultado erróneo, por lo cual se evalúa si es necesario crear una notificación para el usuario (ver
Figura 8).

Figura 8: Capa 3: Procesamiento Local.

La interacción entre el usuario y el PV es por medio de las solicitudes estructuradas en base al Lenguaje de Interfaz para Animaciones en 3D ($LIA{\cdot}3D$) [66]. Estas solicitudes son interpretadas por el PV para generar los cambios en el AV. Se analizan los siguientes tres elementos: el avatar que realizará el cambio, la acción y una marca de tiempo que indica en qué momento debe ser realizada la acción. Por ejemplo:

<accion id=1 tiempo = 1259707742640>
<caminar>3</caminar>
</accion>

Al desglosar las solicitudes, estas son clasificadas en base al avatar que contienen y son encoladas de acuerdo a la marca de tiempo (ver **Figura 9**). El orden es tomado en cuenta por una ranura de tiempo dinámica (RTD) que indica cuales y cuantas acciones se ejecutarán en un proceso.

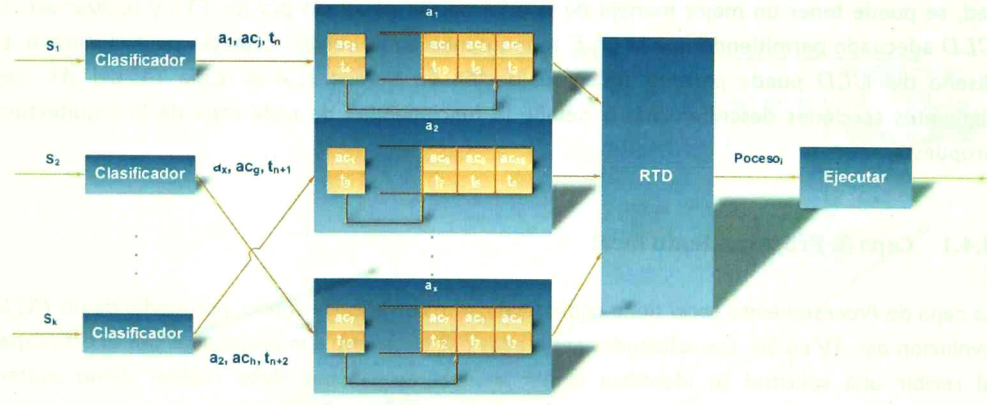


Figura 9: Tratamiento de solicitudes.

Al ordenar las acciones de cada avatar, se obtiene correctamente la secuencia de cambios en el AV, dando como resultado la evolución deseada. La finalidad de manejar una RTD es administrar la carga de trabajo del PV, lo cual repercute en la visualización en tiempo real y en la QoE. La RTD está administrada de acuerdo a un incremento de tiempo como sigue:

$$RTD = marca\ de\ tiempo\ (mt) + incremento\ de\ tiempo\ (\Delta t)$$

En base al tamaño de la RTD_x, son agrupadas un número determinado de acciones, generando un proceso (φ_i).

$$| \varphi_i | \approx | RTD_x |$$

El tamaño de la RTD_x es modificado dependiendo de la carga de trabajo (CT_x) del PV. Cuando la CT_x impide obtener una QoE aceptable provocada por una lenta visualización, la RTD_x disminuye dejando menos acciones en el proceso (φ_i). Es decir:

Si

$$CT_x\ (PV)\ > tp_r\ (\varphi_i)$$

o

$$CT_x\ (PV)\ > TR$$

entonces

$$|\ RTD_x|\ y\ |\varphi_i|\ disminuyen$$

La cantidad de acciones contenidas en un proceso varia al modificar el tamaño de RTD_x, como se muestra en la **Figura 10** en RTD_1 son seleccionadas cuatro acciones de diferentes avatares (a_1, a_2, a_{x-1}, a_x) para ser ejecutadas en el proceso φ_i, mientras que en la RTD_4 sólo es seleccionada una acción perteneciente al avatar a_x que será ejecutada en el proceso φ_{i+3}.

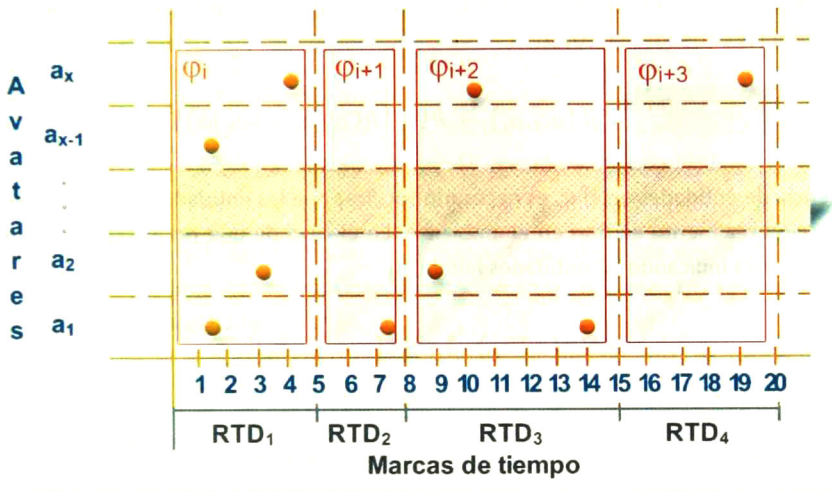


Figura 10: Ranura de tiempo dinámica.

El valor inicial de Δt es de 60 milisegundos. Cuando el tiempo de procesamiento de un proceso sobrepasa los 60 milisegundos, se toma el tiempo exedido para ser decrementado a Δt

Si $tp(\varphi_i) > 60$ *milisegundos entonces*

$$decremento\ de\ \Delta t_{i+1} = \Delta t_i - (tp_r(\varphi_i) - TR)$$

De lo contrario

$$incremento\ de\ \Delta t_{i+1} = \Delta t_i + (TR - tp_r(\varphi_i))$$

Cuando es ejecutado un proceso, se realiza un manejo de la concurrencia entre los avatares o entre las partes de un avatar, generando bloqueos momentáneos del (los) avatar(es) o la(s) parte(s) que se requieren modificar. Para lo cual, existe un gestor de conflictos capaz de administrar tanto las entidades, como las partes internas de una entidad.

Las acciones (ac) ejecutadas tienen una consecuencia en el ambiente, generando un nuevo estado del ambiente virtual (EsA).

$$\{ac_k\ (a_l) \mid ac_5(a_1), ac_1\ (a_2), \cdots, ac_j(a_i)\} \overset{t_x}{\rightarrow} EsA$$

En cada EsA generado se verifica la consistencia mínima local $(VCL\ (EsA))$, es decir, validar cualquiera de los siguientes eventos:

Colisiones entre los elementos del α, es decir, esta tarea debe ser realizada cuando dos entidades se traslapen en un tiempo t_x, provocando una respuesta del ambiente que contendrá las entidades en cuestión.

$$Col\ (ac_i, a_j) \overset{t_x}{\rightarrow} REv\ \left(RCol(a_j, a_j \mid o_h \mid \alpha)\right)$$

- Existencia de entidades en él α, es necesario verificar que las entidades que deben realizar una acción realmente existan en el ambiente. En el caso de que no existan, se generará una respuesta indicando las entidades faltantes.

$$ExEn(a_j \mid o_h) \overset{t_x}{\rightarrow} REv\ (\neg\ a_j \mid \neg o_h)$$

Acciones (ac) válidas para el α, cuándo se desea realizar una acción con cierta entidad es necesario verificar que la acción sea válida (conocida) para dicha entidad, si no es válida, se genera una respuesta indicando que dicha entidad no conoce la acción solicitada.

$$VAc\ (ac_i, a_j \mid o_h) \overset{t_x}{\rightarrow} REv\ \left(\neg\ (ac_i, a_j \mid o_h)\right)$$

De esta forma, el α se mantiene consistente a nivel local. Al realizar la VCL pueden surgir notificaciones que son estructuradas de acuerdo a los eventos descritos en $LIA{\cdot}3D$. Por ejemplo:

```
<evento>
        <colision>
                <objeto 1> 1 </ objeto 1>
                <objeto 2> 4 </ objeto 2>
        </colision>
</evento>
```

Con las características mencionadas es posible modificar la secuencia de pasos de una correcta evolución del AV, la cual fue descrita en la sección 0. Ahora se consideran: la RTD, la notificación de eventos y la verificación de la consistencia mínima local VCL. La secuencia sería la siguiente (ver **Figura 11**):

1. El PV recibe de 1 a n solicitudes que contienen acciones que deben ser ejecutadas por algún avatar. $\{s(ac_i, a_j) \mid s(ac_1, a_3), s(ac_8, a_6) \cdots s(ac_k, a_l)\}$

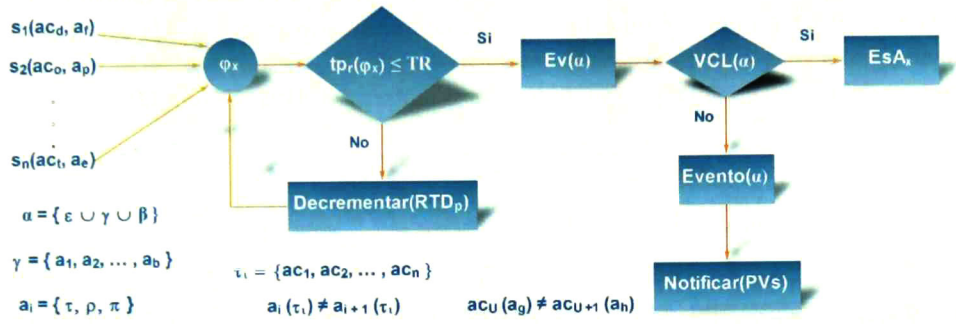


Figura 11: Evolución del AV considerando una VCL.

2. Las acciones son agrupadas en un proceso φ_i que debe ser ejecutado en un periodo de tiempo que permita realizar una visualización en tiempo real. $tp_r(\varphi_i) \leq TR$. Para lo cual, se evalúa lo siguiente:
   a. Si el proceso φ_i no es realizado en un tiempo real, se decrementa el tiempo excedido a la RDT_p actual.

3. Se verifica la consistencia mínima local ($VCL(\alpha)$), si ocurre alguno de los eventos descritos (colisión, inexistencia de avatares, acciones no validas) se realizan los siguientes pasos:
   a. Generar una respuesta con el evento correspondiente. $\{REv\ (RCol(a_j, a_j \mid o_h \mid \alpha)) \mid REv\ (\neg\, a_j \mid \neg o_h) \mid REv\ (\neg\, (ac_i, a_j \mid o_h))\}$
   b. Enviar el evento a los PVs involucrados. $Notificar(PVs)$

4. Un nuevo estado del ambiente (EsA) es generado para cada proceso φ_i.

De forma paralela a las tareas realizadas por la capa de Procesamiento Local, las solicitudes realizadas por un usuario son enviadas a los integrantes del α, para que sean tomadas en cuenta en tiempo y forma. La comunicación entre los PVs del α se realiza a través de dos tipos de mensajes: mensajes de actualización y mensajes de consistencia. Un mensaje de actualización contiene la misma estructura de las solicitudes, mientras que un mensaje de consistencia contiene la descripción del ambiente (el cual será descrito en la siguiente sección).

3.4.2 Capa 2: Verificación de Consistencia

Cuando se tiene más de un PV en la EAV, es necesario mantener una consistencia entre ellos. Esta tarea es realizada por la capa de *Verificación de Consistencia* (ver **Figura 12**), que tiene

como entrada principal mensajes de consistencia con el estado actual del AV de los diferentes PVs.

El cambio en las posiciones de los avatares puede ocasionar un cambio en las AIs. Por lo cual, la Capa 2 analiza si es necesario generar nuevas AIs, al detectar algún cambio esta capa envía a la Capa 1 Distribución de Actualizaciones las actualizaciones en las AIs para cuando se requiera un envió se realice a los PVs involucrados. Aunado a esto, se generan mensajes de consistencia que son enviados para verificar la consistencia en el AV (ver **Figura 12**).

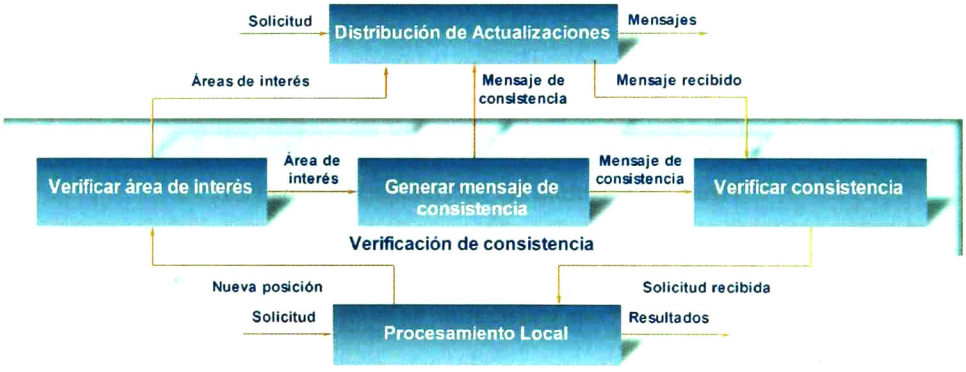


Figura 12: Capa 2: Verificación de consistencia.

Los elementos de un AV están agrupados en áreas de interés (AI) delimitadas bajo dos conceptos, descritos a continuación:

- ***Área de visión*** *(ver* **Figura 13.*a*)***:*** es definida por las características de cada avatar, se indica el alcance de visión de los mismos, identificando los elementos del AV que se pueden detectar. De esta forma, se evita el envío y/o el procesamiento de mensajes que no son perceptibles por cada avatar.

- ***Limitaciones dadas por el ambiente*** *(ver* **Figura 13.*b*)***:*** en un escenario es posible encontrar objetos que limitan el área de visión de los avatares, haciendo innecesario establecer una comunicación entre ellos.

a) **b)**

Figura 13: a) Área de vision de un avatar, **b)** Limitaciones del AV.

En base a estos dos conceptos, se pueden identificar: (i) las Áreas de Interés de los avatares, las cuales crean divisiones en el AV (ver **Figura 14**), (ii) los nodos que ejecutan el PV que pertenece a cada AI. La agrupación de nodos reduce la cantidad de mensajes que se envían entre los nodos de los PVs del AV. En la **Figura 14** se ejemplifica que los nodos que ejecutan un PV pueden estar interconectados por redes heterogéneas. Por lo cual, los nodos de un AI pueden no estar en un mismo tipo de red, lo cual debe ser transparente para el usuario.

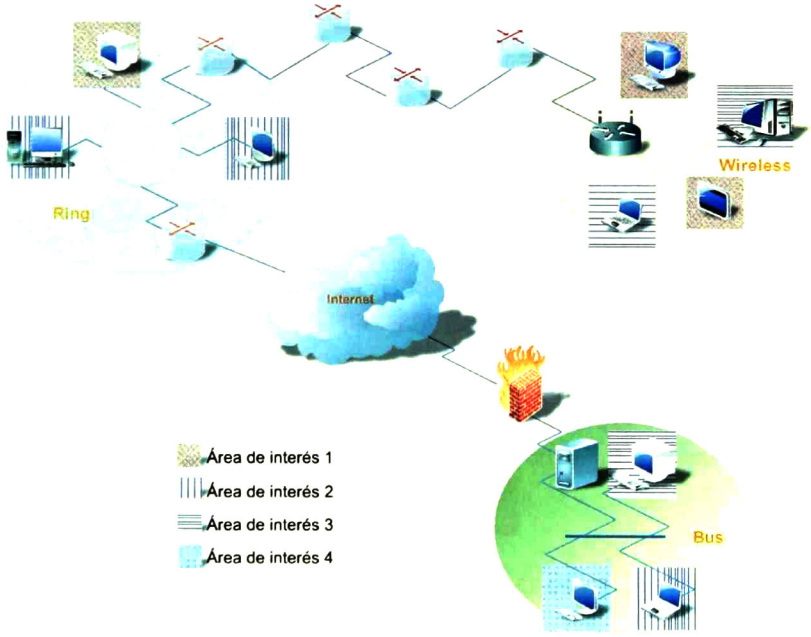


Figura 14 : Áreas de interés.

En cada AI existe un nodo coordinador (C) encargado de cotejar los estados de los nodos simples (NS) del AI. Un AI al menos tiene un coordinador y un NS. Cada nodo, ya sea coordinador o nodo simple brinda como servicio un PV que interactúa en el AV. Con lo cual:

$$C = \{PV_{\alpha}, PV_{\varepsilon}, \cdots, PV_{\mu}\};\ C_{\beta} \subseteq C$$
$$NS = \{PV_{\chi} | PV_{\chi} \neq PV_{\eta} \in C\};\ NS_{\varepsilon} \subseteq NS$$
$$|C| \leq |NS|$$
$$AI_{\iota} = \{C_{\beta}, NS_{\varepsilon}\}\ y\ AI = \{AI_{\iota} | \iota \in \aleph\}$$
$$\therefore\ \alpha\{IA\}$$

Una vez identificadas las áreas de interés se asignan roles a cada PV. El coordinador es elegido tomando en cuenta la distancia promedio entre los nodos (ver **Figura 15**), por medio del campo TTL de los mensajes, que indica el número de saltos en la red que realiza el mensaje. Esto permite dos características: a) disminuir el retardo en la entrega, b) reducir la pérdida de mensajes, debido a la cercanía de los nodos.

Como ejemplo, en la **Figura 15** se muestran las distancias entre los nodos del *área de interés 1*, se calcula la distancia promedio para cada nodo, siendo el nodo 3 el de menor distancia promedio del área, por lo cual este se toma como coordinador del *área de interés 1*.

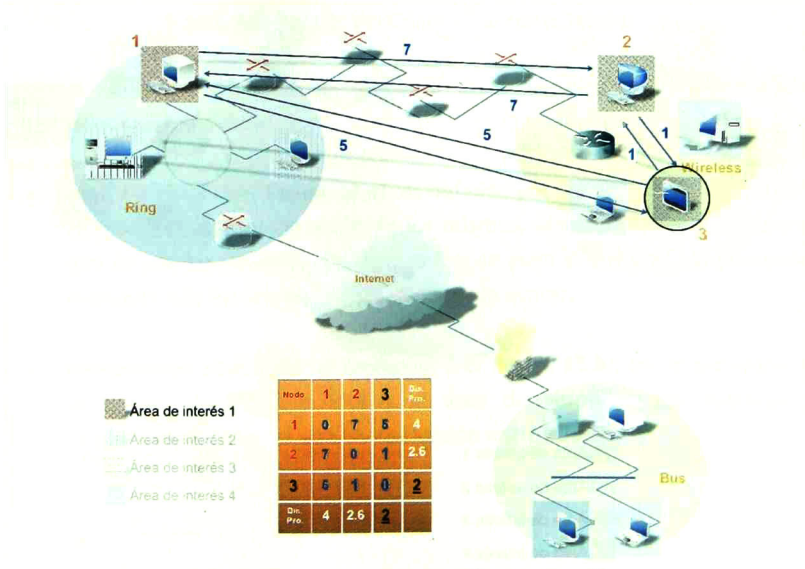


Figura 15 : Distancia promedio entre nodos.

La interacción entre los PVs se realiza entre el nodo simple y el coordinador (ver **Figura 16.a**) de cada AI. De esta forma, se llega a un acuerdo del estado actual del área de interés en cuestión y entre coordinadores (ver **Figura 16.b**) para validar el estado general del AV.

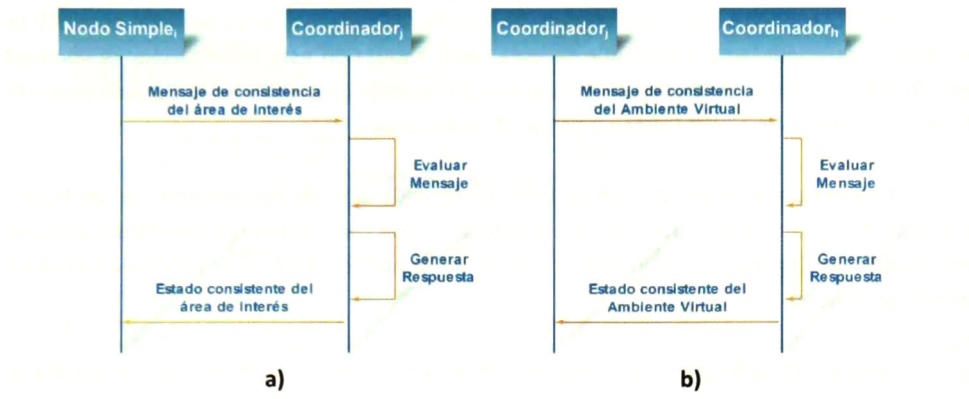


Figura 16: a) Consistencia en un área de interés, **b)** Consistencia en el AV.

Debido a que los nodos establecen una comunicación $P2P$, se considera suficiente analizar el comportamiento de dos nodos del AV. El resto de los nodos tendrán un comportamiento similar. Se describe la comunicación entre los PVs del AV de la siguiente manera:

Cuando dos nodos f y g participan en la evolución de α, se asume que se puede establecer una conexión entre f y g. Los nodos de un área de interés forman un grafo $G = (V, E)$ donde $(f, g) \in E$, si y solo si permanecen a la misma área de interés (AI). Es decir, $AI(f, g) = 1$ sii f y g pertenecen a la misma área de interés, de otra forma $AI(f, g) = 0$. De esta manera, el α está formado por al menos un grafo. Lo anterior se define como sigue:

$$G\alpha = \sum_{i=1}^{n} GAI_i$$

donde, n es el número de áreas de interés en el α.

En cada grafo de un área de interés al menos existen dos nodos, los cuales mantienen dos tipos de comunicación. Una es utilizada para los mensajes de actualización (MA) y la otra es empleada para los mensajes de consistencia (MC). De esta forma, un AI contiene al menos dos nodos que tienen como servicio un PV del AV, definido como:

$$GAI = \sum_{i=2}^{x} PV_i \qquad \begin{matrix} PV_i \in AV \\ x \geq 2 \end{matrix}$$

El canal de comunicación requerido para MA no es necesariamente fiable. Sin embargo, los MCs requieren un canal fiable. Se asume que al analizar el comportamiento de una canal fiable es suficiente, ya que se considera la comunicación en un medio no fiable como un sub-caso de un medio fiable [**67**].

El MC contiene tres características principales de cada avatar: su posición en el $AV(x, y, z)$, estado emocional y la acción que se está ejecutando. Con esta información, se compara el estado del usuario con el estado del coordinador. El mensaje de consistencia se puede enviar si: (***i***) Transcurre un tiempo t_x, (***ii***) Existen n cantidad de acciones

Cuando un mensaje de consistencia es recibido por un coordinador, se verifican dos aspectos. Primero, que los nodos integrantes del área a su cargo estén representando un correcto estado del AV. Segundo, si algún nodo requiere un cambio de área, se le notifica a cual área de interés pertenece.

Si el coordinador es el que requiere de un cambio de área de interés, se solicita a los integrantes del AI que uno de ellos sea el nuevo coordinador. Así, el nuevo coordinador se comunicará con los demás nodos para notificarles el cambio. El nuevo coordinador será el que contenga la segunda menor distancia promedio del AI.

Cuando un AI contiene demasiados nodos como para continuar con un solo coordinador (debido a la sobrecarga de trabajo), se solicita a los nodos que alguno de ellos de soporte a una parte del área de interés. La elección del segundo coordinador se realiza de la misma forma, es decir, en base a la distancia promedio hacia los demás nodos. Cuando uno de los usuarios se desconecta o cambia de AI, se re-calcula si los nodos pueden ser administrados por un solo coordinador. En el Capítulo 6, se muestran a detalle los algoritmos necesarios para la elección del coordinador de un AI.

3.4.3 Capa 1: Distribución de actualizaciones

Esta capa se encarga de establecer la comunicación entre los nodos que ejecutan los PVs (ver **Figura 17**). Otra tarea es identificar el tipo de mensaje para determinar el nodo(s) destino(s). Cuando el PV recibe una solicitud (mensaje de actualización), analiza si fue recibida a tiempo para así ser enviada a la Capa 2, la cual hace la función de canal de comunicación entre la Capa 1 y la Capa 3. Cuando la Capa 1 recibe algún mensaje proveniente de la Capa de Verificación de Consistencia, se analiza el tipo de mensaje para seleccionar el tipo de canal (fiable o no fiable) que requiere el mensaje. Antes de ser enviado el mensaje es estimado el retardo que sufrirá para conocer si llegará a tiempo para realizar una visualización en tiempo real en los PVs restantes (**Figura 17**).

Figura 17: Capa 1: Distribución de actualizaciones.

Como se ha mencionado, la $Ev(\alpha)$ puede estar dada por n usuarios que participan por medio de un PV, el cual es soportado por un nodo, de esta forma cada usuario U_i tiene asignado un nodo N_i, ya sea coordinador C_β o nodo simple NS_ε:

$$U = \{U_x, U_y, \cdots, U_Z\};\ U_i \subseteq U$$
$$N_i = \{C_\beta | NS_\varepsilon\} = U_i$$

La evolución de un $AV(Ev(\alpha))$ está dada por un conjunto de usuarios (U) que interactúan con el AV en un determinado tiempo (t_x):

$$U \xrightarrow{t_x} Ev(\alpha)$$

En la comunicación en una arquitectura $P2P$ se realizan conexiones de igual a igual entre los PVs participantes, reduciendo la dependencia hacia alguno de ellos. Sin embargo, en este caso se incrementaría innecesariamente la cantidad de mensajes entre los nodos, lo cual se evita con la agrupación de PVs en áreas de interés. Es posible que un PV_i envíe una acción a un PV_j en un tiempo t_x de la siguiente manera:

$$PV_i\left(Enviar\left(s\left(ac_r, a_f\right)\right)\right) \xrightarrow{t_x} PV_j\left(Recibir\left(s\left(ac_r, a_f\right)\right)\right)$$

Se considera que un PV sólo puede manipular un avatar en un tiempo (t_x). De esta forma, se asigna un avatar (a_i) a un PV en un tiempo (t_x):

$$a_i \xrightarrow{t_x} PV_h$$

Las modificaciones (solicitudes) que se realicen al avatar a_i asignado al PV_h son enviadas a los PVs del AI, a través de mensajes de actualización. Es importante considerar el retardo de los mensajes con el objetivo de continuar brindando una representación gráfica en TR y mantener una QoE aceptable. Un retardo grande o una alta tasa de pérdidas de mensajes repercuten en la consistencia entre los PVs provocando que el AV no sea consistente entre los PVs. Se asume que

entre el PV origen y el PV destino es posible establecer un canal de comunicación, el cual es fiable.

Es necesario delimitar un máximo retardo (MR) en la comunicación entre los PVs para el manejo de la información. En base a [**68**], se definen tres tipos de usuarios, clasificados de acuerdo al tiempo de conexión de los usuarios. Cada perfil de usuario tiene asignado un retardo considerado como aceptable (ver **Tabla 1**).

Tabla 1: Perfiles de usuarios.

Perfil	Tiempo de conexión (min.)	MR (ms.)
Bajo (PB)	PB =< 1	Retardo <= MR <=339
Medio (PM)	PM > 1 y PM =< 9	Retardo <= MR <=176
Alto (PA)	PA >10	Retardo <= MR <=232

Tomando en cuenta un límite de tiempo razonable para cumplir con la restricción de TR y a los tiempos de los perfiles de los usuarios (TPU), se puede administrar el tiempo total de una acción (TTA), es decir, el tiempo que deberá tomar una solicitud desde su origen hasta su destino. Es decir:

$$TTA \leq TR + MR$$

Al conocer los tiempos que debe tomar la acción, se puede administrar mejor los mensajes que son enviados por la red y de esta forma estimar si es posible cumplir con dichos tiempos. Para estimar el TTA, se modelo la comunicación P_2P utilizando una cadena simple de Markov (**Figura 18**).

En la cadena de Markov el estado q_1 representa la espera de solicitudes. Sea η la probabilidad de ocurrencia de las solicitudes. Se asume que las solicitudes siguen una distribución de Poisson, debido a que ocurren de forma independientemente, aleatoriamente y una después de otra [**69**] [**70**] [**71**]. El número de solicitudes que ocurren en un periodo de tiempo dependen solo del tamaño del periodo de tiempo. Sea κ la probabilidad para conocer la probabilidad de que el retardo estimado este en el periodo MR correspondiente. Se asume que el retardo end-to-end sigue una distribución de Pareto. En [**72**] se muestra que el retardo end-to-end sigue de una manera apropiada una distribución de Pareto. El objetivo de utilizar una distribución de Pareto es para predecir la CDF del retardo end-to-end en cualquier comunicación entre los PVs. Se considera que la pequeña diferencia entre la real CDF y la distribución de Pareto del retardo end-to-end no es relevante para la distribución de los PVs. En otras palabras, la redundancia en la compresión de mensajes (causada por un error de estimación) puede solo contribuir a la reducción del retardo end-to-end y a disminuir la congestión de la red. El estado q_2 representa el estado en el cual los PVs estiman el retardo end-to-end. En base al resultado de la estimación se elige el estado q_3 (el retardo estimado menor o igual al MR) o q_3 (el retardo estimado es mayor a MR). El estado

q_5 representa el estado en el cual las solicitudes son comprimidas con el objetivo de cumplir con MR. Finalmente las solicitudes son enviadas en el estado q_6.

Un modelo alterno y simplificado al proceso del manejo de la información de la capa de distribución de la arquitectura propuesta sería una máquina de estados **Figura 18 b.** En dicho modelo, descartando el comportamiento estadístico de distribución del puerto visual, prevalecería el flujo y los estados de la cadena de Markov de la **Figura 18 a.** Este proceso del manejo de la información de la capa de distribución es validado de manera experimental en el en la sección 5.1.

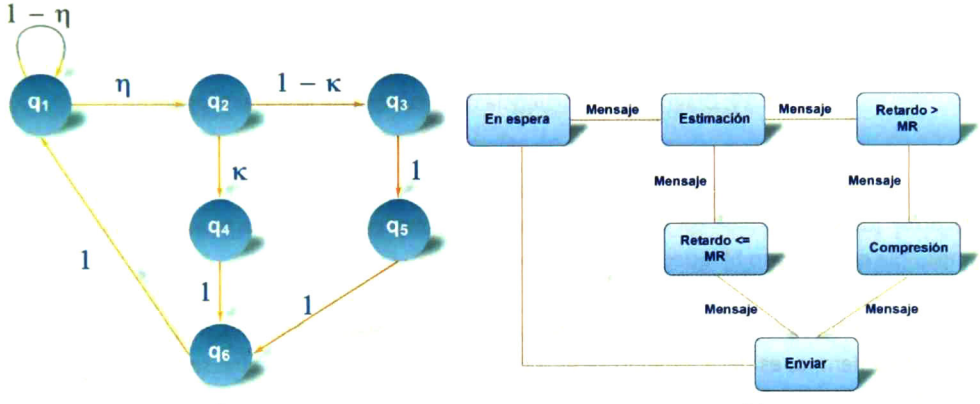


Figura 18: a) Cadena de Markov propuesta. b) Maquina de estados en notación UML.

La cadena propuesta es no absorbente, ergódica y finita. Contiene los siguientes estados:

Tabla 2: Estados de la cadena de Markov propuesta.

Estado	Significado
q_1	Estado en espera de mensajes.
q_2	Estimación del retardo end-to-end de los mensajes.
q_3	Retardo estimado mayor a *MR*.
q_4	Retardo estimado menor o igual a *MR*.
q_5	Compresión del mensaje con un retardo estimado mayor a *MR*.
q_6	Envió de mensajes.

$$no\ absorvente \rightarrow \forall\, i, i \in S\ q_{ii} \neq 1$$
$$finita \rightarrow \forall i, j \in S\ v_{ij} = P\,[X_n = j \mid X_0 = i]\ n > 0$$

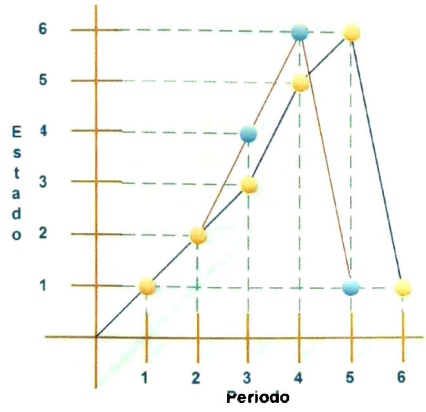


Figura 19: Diagrama de secuencia de la cadena de Markov propuesta.

El diagrama de secuencia (ver **Figura 19**) de la cadena propuesta muestra que es posible recorrer los seis estados en cinco o seis periodos, probando de una manera evidente que es imposible que un caso bloqueo en los estados.

La situación más importante en este caso es saber cuándo el mensaje a enviar será recibido a tiempo, es decir, cuándo se va a llegar al estado q_4, para lo cual se tienen las siguientes probabilidades:

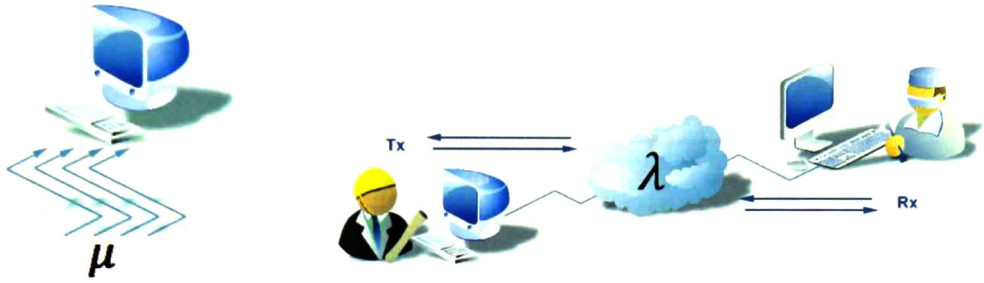
$$P_{14}^{(2)} = P\,(X(2) = 4|X(0) = 1) = P_{12} \cdot P_{24} = \eta \cdot \kappa$$
$$P_{14}^{(3)} = P\,(X(3) = 4|X(0) = 1) = P_{11}\ \ P_{12}\ \cdot P_{24} = (1-\eta)\ \cdot \eta \cdot \kappa$$

Desde el estado inicial (q_1) se puede llegar en al menos en dos o tres periodos al estado q_4. La matriz de transición es la siguiente:

$$Matriz\ de\ Transicion\ P = \begin{bmatrix} 1-\eta & \eta & 0 & 0 & 0 & 0 \\ 0 & 0 & 1-\kappa & \kappa & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{aligned} Enviado &\rightarrow \frac{\kappa}{\eta} \\ En\ tiempo &\rightarrow \kappa \\ Sobre\ tiempo &\rightarrow 1-\kappa \\ Envios &\rightarrow \eta \\ En\ espera &\rightarrow 1-\eta \end{aligned}$$

Las solicitudes son caracterizadas por una distribución de Poisson por que se consideran aleatorias e independientes entre ellas (η). El retardo en la comunicación P_2P está dada por una distribución de Pareto (ver **Figura 20 a**) en base a [**72**].



a) Solicitudes recibidas. **b)** Solicitudes enviadas.

Figura 20 : Solicitudes de acciones.

Siendo κ los mensajes enviados, caracterizados por una distribución de Pareto y $1 - \kappa$ los mensajes que sobre pasan el tiempo de envío (ver **Figura 20 b**), los cuales se comprimen. La finalidad es realizar la compresión de los paquetes (Pa) a enviar cuando el tiempo para el envío sobrepase el tiempo requerido ($TReq$) (ver **Figura 21**), modelado en la transición $q_2 \rightarrow q_3$, de la cadena propuesta.

Si TReq(Pa) > TTA entonces
Com(P)

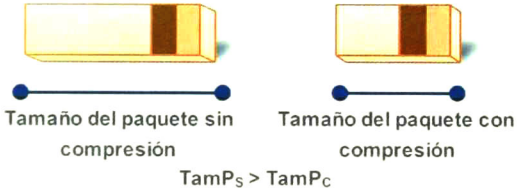


Figura 21: Tamaño de los paquetes enviados.

El tiempo de serialización (TS) (ver **Figura 22**) y el tiempo de propagación (TP) de un paquete comprimido (P_C) es menor que el tiempo del paquete sin comprimir (P_S). Es decir:

$$TS(P_S) + TP(P_S) > TS(P_C) + TP(P_C)$$
$$TArribo(P_S) = 2TS(P_S) + TP(P_S)$$
$$TArribo(P_C) = TCom(P_C) + 2TS(P_C) + TP(P_C) + TDes(P_C)$$
$$TArribo(P_C) \leq TTA$$
$$\frac{TArribo(P_S)}{TArribo(P_C)} \approx 1$$

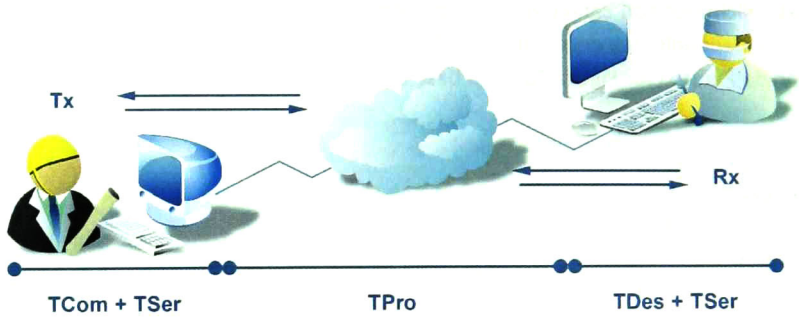


Figura 22: Tiempos considerados para los paquetes.

La compresión de los mensajes no se realiza todo el tiempo, debido a que podría restarle recursos a la capa de procesamiento para realizar la visualización en TR. La utilización de la cadena propuesta ayuda a obtener un mejor manejo de la información en el AV. Con las características hasta el momento descritas es posible obtener los siguientes valores de los recursos en la red:

3.5 Conclusiones

La descripción de un AV permite identificar los elementos y su interacción que da como resultado la evolución del AV. Los elementos son manipulados a través de solicitudes hechas por los usuarios, dichas solicitudes contienen al menos el identificador de un avatar y el identificador de la acción que debe realizar el avatar en cuestión.

Se describió la estructura del esqueleto de un humano virtual (avatar) que permite realizar animaciones lo más apegado a los movimientos de un humano. La estructura es descrita detalladamente en el estándar $H\text{-}Anim$ [**64**].

Una vez descritos los elementos de un AV, se realizó el diseño de una arquitectura de referencia llamada $QoE\text{-}PV$, la cual incorpora la QoE en los AV. Siguiendo esta arquitectura es posible diseñar AVs que compartan PVs, de esta forma las áreas de conocimiento sólo necesitarían diseñar la lógica del AV, módulo del Modelo según el patrón de arquitectura MVC.

En base a la arquitectura de referencia $QoE\text{-}PV$ se diseñó una arquitectura P_2P para los PVs. Los PVs realizan una visualización en tiempo real de la evolución del AV. La definición de Tiempo Real se realizó considerando características de la vista humana para detectar un movimiento fluido. La visualización en tiempo real se debe realizar en los PVs que participen en un AV, es decir se deben distribuir los PV de un AV.

La distribución de los PV con lleva a la administración de: consistencia global, mensajes de actualización, mensajes de consistencia, retardo, nodos que dan como servicio un PV. Una mala

administración disminuiría el grado de aceptación de la QoE, así como también afectaría negativamente la visualización en Tiempo Real de la evolución del AV.

Capítulo 4 Calidad de Experiencia en Puertos Visuales

Objetivo:

- Desglosar las métricas objetivas y subjetivas de la Calidad de Experiencia que garantizan un grado de aceptación por parte del usuario final.

Los *PVs* tienen una interacción directa con el usuario final, debido a que muestran de forma gráfica la evolución del *AV*. Por lo cual, es necesario considerar las características requeridas para estos con el fin de asegurar una calidad de experiencia (*QoE*) aceptable.

La *QoE* en los *AVs* ha sido poco analizada como se indicó en la sección del estado del arte. Uno de los trabajos más representativos que trata este problema es presentado en [**12**]. Este trabajo describe una clasificación de métricas que influyen en la *QoE* en los *AVs* y consideran un *AV* inmerso. En este capítulo se mencionan además todas las métricas con la finalidad de que sean consideradas para un trabajo futuro donde se implemente sobre un *AV* inmerso. Dado que nuestro trabajo no trata con ambientes inmersos, algunas de las métricas no serán útiles en esta investigación.

Las principales métricas que influyen en la *QoE* son (**Figura 23**): la Calidad de Servicios (*QoS*) y la experiencia del usuario. En [**12**] para la *QoS* se consideran diferentes parámetros involucrados en cualquier aplicación de red, así es necesario identificar cuáles son importantes en los *AV*. En la sección 4.2 se mencionan los parámetros que deben ser considerados para un *AV* en el que participan usuarios situados en diferentes áreas geográficas. En la experiencia del usuario se consideran métricas subjetivas que dependen de diferentes factores relacionados con el usuario final, como: estado de ánimo, nivel de conocimiento en la manipulación del *AV*, edad, etc.

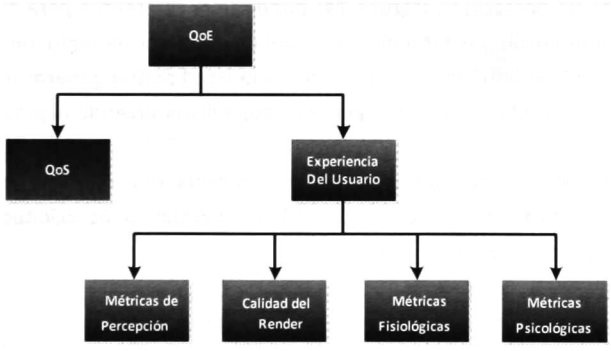


Figura 23: *QoE* en los *AVs*.

En las siguientes secciones se describen a grandes rasgos las métricas de la *QoS* como de las métricas de la Experiencia del usuario utilizadas en los ambientes inmersos.

4.1 Experiencia del usuario

Las métricas englobadas en la Experiencia del usuario están enfocadas en medir el impacto que tiene el ambiente en el usuario, para saber el grado de aceptación de la *QoE*. Una de las métricas es la percepción, en la cual se considera el nivel de dificultad para administrar el

ambiente (en nuestro caso el PV), ya que debe ser sencilla, evitando la utilización de tecnicismos entre las diferentes áreas de conocimiento involucradas en la EAV. Para facilitar la comprensión de la evolución del AV, se realiza un desplegado grafico (calidad del render), que muestra al usuario una visualización con alta calidad lo más apegado a la realidad. En ocasiones el AV podría generar fatiga provocando que el usuario no desee participar en la evolución del AV, este tipo de métricas pertenecen al conjunto de métricas fisiológicas. Otra situación por la cual la QoE no es aceptable, es cuando el AV causa estrés al usuario, métrica considerada en las métricas psicológicas.

Los resultados de las métricas de la experiencia del usuario depende del tipo de usuario final, es decir, es muy probable que un adulto conteste de forma diferente a un joven, debido a que podrían no tener el mismo nivel de conocimientos para manipular el AV. Lo mismo pasa con personas de diferentes áreas del conocimiento, ya que un abogado podría no tener la misma agilidad para manipular el AV que un informático, para lo cual es necesario permitir la manipulación del AV de una forma sencilla.

4.1.1 Métricas de percepción

Las métricas de percepción (**Figura 24**) pueden ser diferentes para cada usuario, porque dependen de las preferencias y estado de ánimo del usuario. Por ejemplo, en un AV que requiere de mucha actividad del usuario, en un usuario de corta edad podría generar entusiasmo, mientras que a un usuario de avanzada edad podría generar fatiga disminuyendo el grado de satisfacción.

La impresión de colaboración entre usuarios o entre el usuario y los objetos (elementos del AV) puede ser calificada por el usuario, si el PV no responde a las solicitudes del usuario, dará la impresión de no tener colaboración con el AV.

En la **Figura 24** se muestran las métricas consideradas en la percepción. La percepción engloba métricas como:

- **Satisfacción**: mide si el ambiente cumple con las expectativas del usuario, en las cuales se pueden considerar: fiabilidad de resultados, coherencia en la lógica del ambiente, etc.

  Intuitiva: indica si el usuario cree que el ambiente es fácil de administrar, o si los resultados mostrados son fáciles de interpretar.

  Preferencias: evalúa si el usuario considera que las formas de manipulación del ambiente son suficientes.

- **Tele-presencia**: indica el nivel en que el usuario se siente parte del ambiente.

- Entre otras.

Las métricas de percepción tienen como finalidad evaluar el nivel de satisfacción del usuario, dicha satisfacción es medida a través de conocer como el usuario se siente al interactuar con el ambiente, es por eso que los resultados de la evaluación depende de cada usuario.

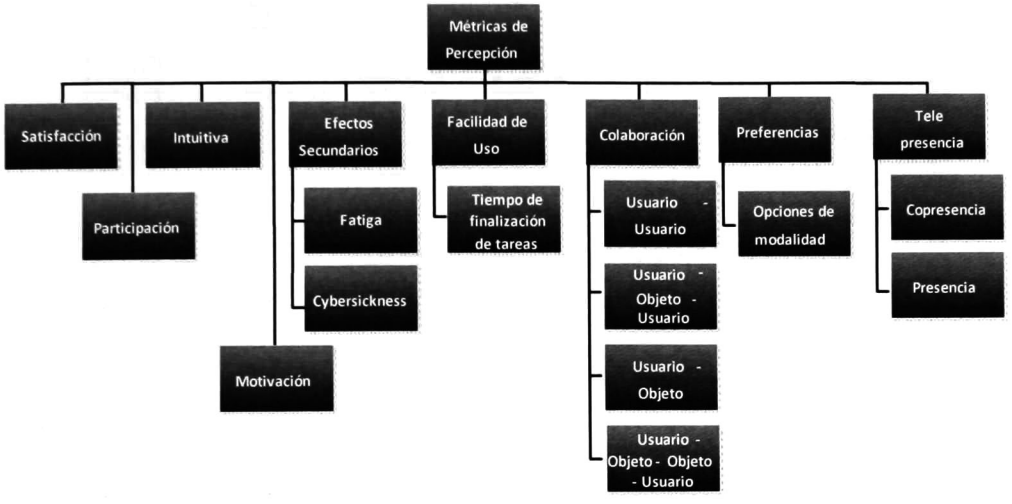


Figura 24: Métricas de Percepción.

La tele presencia es utilizada en AV inmersos, esta es la encargada de hacer sentir al usuario que forma parte del AV. Por ejemplo, si el AV representa a una granja, este debe ser capaz de representar gráficamente árboles, plantas, animales, así como lo sonidos, para que el usuario sienta que forma parte de la situación.

La tele presencia no es aplicable en este trabajo, debido a que hasta el momento está enfocado en la visualización del AV. Si se calificara este aspecto la evaluación seria baja, por considerarse aspecto fundamental en la tele presencia.

Para realizar la medición de reacciones secundarias es necesario considerar la aplicación de algún(as) pruebas psicotécnicas que indiquen el nivel de fatiga del usuario. Para lo cual se requiere de la colaboración del área de la psicología. Las preferencias dependen tanto del usuario como del área de conocimiento en la que se utiliza el AV, con ello, se requiere un estudio más amplio para definir las características necesarias de una manera más adecuada.

4.1.2 Calidad del Render

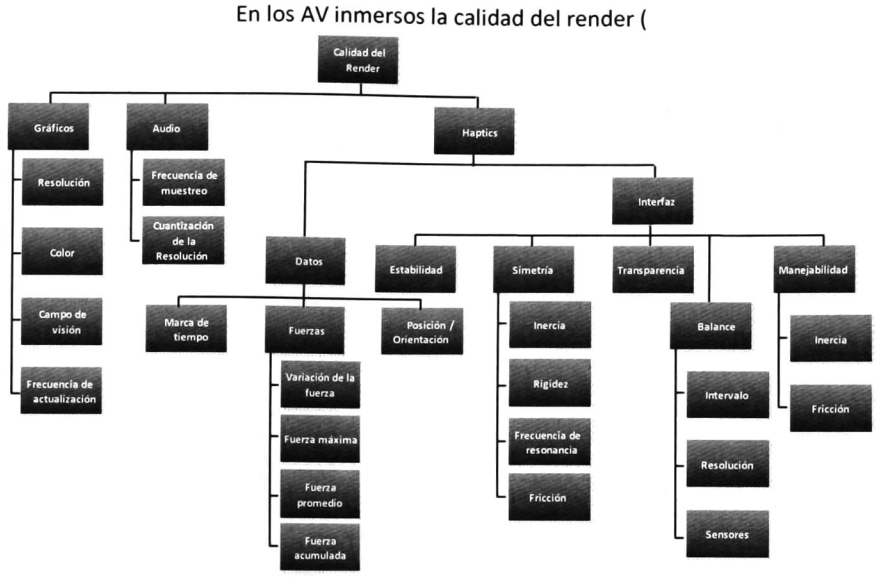


Figura 25) es dividida en: gráficos, audio y haptics. En el área de los haptics se considera desde la manejabilidad hasta el tiempo de respuesta del hardware, por permitir la iteración del usuario con el AV inmerso.

La tele presencia involucra el audio, el cual debe tener una frecuencia de muestro óptima para sincronizar los gráficos con el sonido. Por ejemplo, si el usuario solicita cerrar una puerta, tanto la representación 3D como el sonido deben realizarse de manera sincronizada en el tiempo. El sonido en los *AVs* puede manejarse desde una forma sencilla, como es sólo reproducir el sonido, hasta considerar el tipo de material que se representa en el *AV*.

En un *AV* administrado por diversos *PVs*, el manejo del sonido se vuelve más complejo. Por ejemplo, en un caso que involucre dos avatares ubicados en diferentes habitaciones de una casa, si un avatar habla, el segundo avatar podría no escucharlo, porque influyen diferentes factores como: el material de las habitaciones, la distancia, los decibeles del sonido emitido, etc.

Este trabajo se enfoca únicamente a los gráficos, los cuales consideran aspectos como son: la resolución, el color, el campo de visión y la frecuencia de actualización. Las métricas consideradas en los Haptics son propias para los *AV* inmersos, los cuales hasta el momento no forman parte de nuestra área de investigación. Las métricas Haptics miden la interacción del usuario con el ambiente a través de los dispositivos haptics, si estas métricas se aplicaran directamente al usuario, entonces se estaría evaluando al mismo usuario. Un análisis más

detallado del audio y haptics indicaría si las métricas desglosadas son suficientes para un AV inmerso.

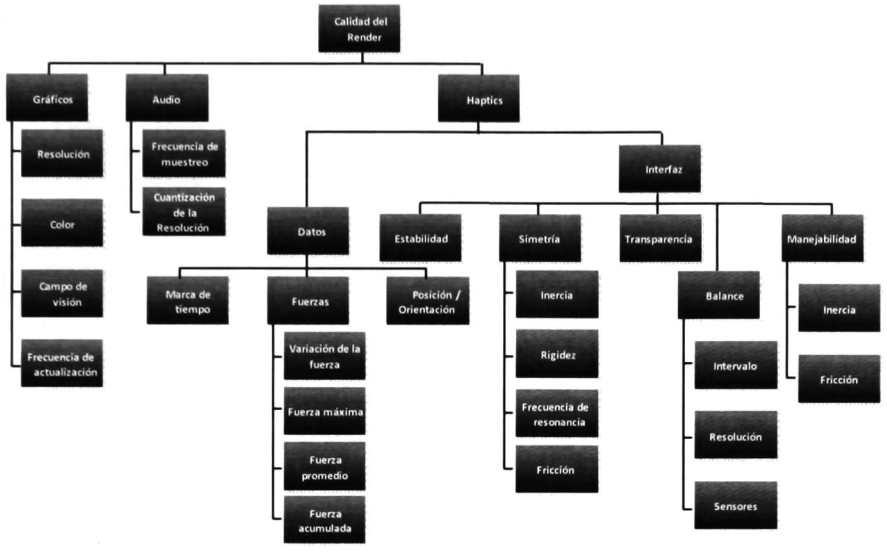


Figura 25: Calidad del Render.

4.1.3 Métricas Fisiológicas

Existen diferentes parámetros biológicos considerados en las métricas fisiológicas (**Figura 26**). Las pruebas utilizadas deben ser diseñadas por especialistas médicos quienes deberán interpretar correctamente los cambios en los parámetros medidos.

Las métricas fisiológicas (ver **Figura 26**) tienen como finalidad medir aspectos físicos para detectar de una manera fiable la reacción del usuario al interactuar con el ambiente, es por ello que se realizan pruebas como:

- **Actividad cerebral**: una característica que puede saber es si el usuario tiene un efecto emocional positivo o negativo.
- **Ritmo cardiaco**: Puede indicar que situaciones en el ambiente le causa estrés o fatiga al usuario.
- **Frecuencia de respiración**: indica la actividad física que tiene el usuario.
- Entre otras.

Figura 26: Métricas Fisiológicas.

4.1.4 Métricas Psicológicas

El estrés, las fobias, entre otros son analizados por las métricas psicológicas (**Figura 27**). Para evaluar estas métricas se requiere de un análisis que identifique las cualidades a interpretar para la evaluación de cada parámetro e identificar al mismo tiempo la situación que provoca una variación en la información. Estas métricas están ligadas con la personalidad del usuario final.

Figura 27: Métricas Psicológicas.

4.1.5 Métricas consideradas en los Puertos Visuales

La elaboración de pruebas que abarquen las métricas anteriores no es una tarea sencilla, debido a que se requiere de la intervención de especialistas como psicólogos, médicos, entre otros, para diseñar pruebas que arrojen resultados óptimos para la medición de la QoE en los PVs. Además, los resultados obtenidos de las pruebas también deben ser valorados por los especialistas, para que la interpretación de dichos resultados sea confiable.

Las métricas consideradas en la experiencia del usuario para este trabajo, son las mostradas en la **Figura 28**. Se eligieron métricas en las cuales no fuera requerido un especialista, las métricas de audio no se consideran debido a que se desea realizar un análisis más detallado que involucre la propagación del sonido en los *PVs* de un *AV*. Así será posible identificar las características necesarias.

Una primera aproximación para evaluar las métricas subjetivas, sería realizar cuestionarios donde las respuestas utilizaran la escala de Likert, la realización de las preguntas podrían ser sencillas (Preguntas del tipo: ¿Está usted satisfecho con la visualización del Ambiente Virtual?), sin embargo el diseñar preguntas que estén dirigidas a diferentes tipos de personas, mostrarían un resultado más confiable.

Si el *AV* es utilizado por un área de conocimiento específico, las pruebas subjetivas contendrían aspectos que valoraran características fundamentales para el área de conocimiento en cuestión.

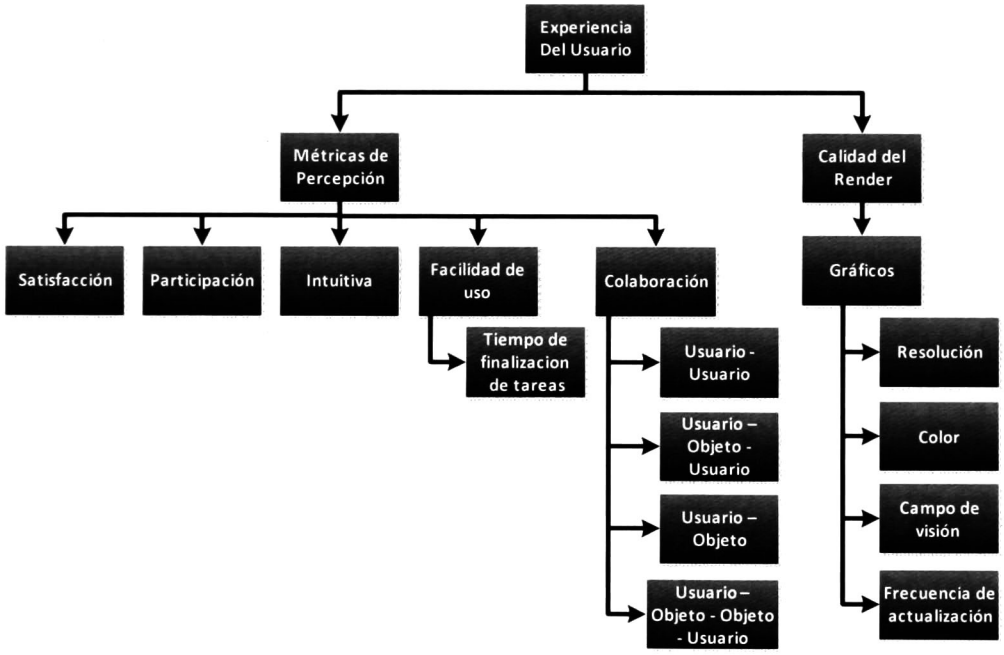


Figura 28: Métricas consideradas en este trabajo de investigación.

4.2 Calidad de Servicios

En **[13]** y **[14]** se describen diferentes métricas para evaluar la *QoS* (**Figura 29**) que influyen en los *AV* distribuidos. Las métricas principales son:

Interactividad: mide el grado de participación de los usuarios para modificar el AV. Para lo cual, son necesarias las siguientes métricas: *a) Velocidad:* rapidez con la que se realizan las solicitudes en el AV. Está relacionado con el retardo de extremo a extremo. *b) Amplitud:* representa las opciones que tiene el usuario para modificar el AV. *c) Asignación:* es la forma de interacción entre el usuario y el AV, es decir, el grado de intuición que ofrece el AV al usuario para interpretar los cambios.

- ***Consistencia:*** cuando en un AV participan usuarios que están en diferentes áreas geográficas, se debe considerar el envió de los estados locales para realizar una sincronización, lo que inevitablemente incurre en una inconsistencia debido a la existencia de retardos de propagación, perdidas, etc. Se identifican dos tipos de consistencia: *a) Espacial:* es la sincronización de estados entre todos los usuarios del AV. *b) Temporal:* se refiere a la consistencia en cada usuario del AV.

Viveza: agrupa los canales sensoriales que se ofrecen al usuario final. Está dividida en: *a) Amplitud:* es el número de canales en el AV (visuales, auditivos, textuales, etc.). *b) Profundidad:* es la resolución de cada canal sensorial (la relación señal a ruido de pico (Peak Signal to Noise Ratio $PSNR$), resolución 3D, resolución audio/video, etc.)

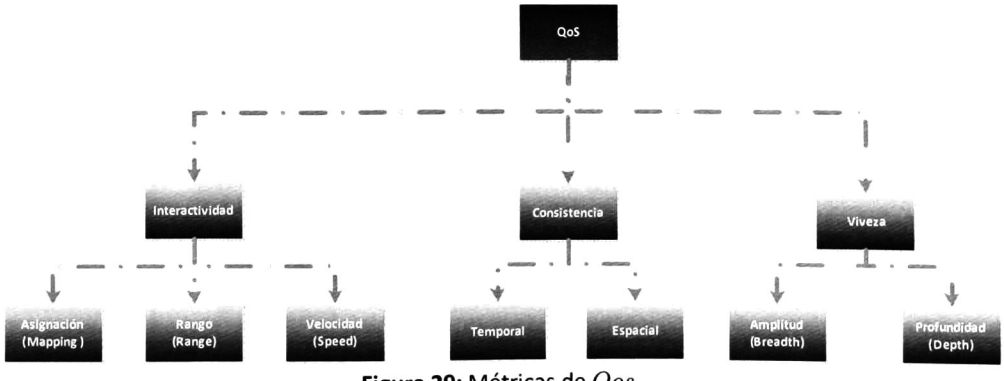


Figura 29: Métricas de Qos.

4.3 Físicas en los Ambientes Virtuales

Hoy en día, los AVs requieren que la representación de una situación sea lo más apegado a la vida real. Por lo cual, los PVs deben tener la capacidad de simular las físicas del mundo real. La agregación de físicas aumenta el grado de la QoE del usuario final. Las físicas son consideradas en la calidad de render, en específico en los gráficos (ver **Figura 30**).

Como se definió anteriormente, el AV está formado por los siguientes elementos: escenario, objetos 3D y avatares. Para cada elemento se definieron las características físicas que permitieran realizar animaciones (cambios) en el AV de una manera lo más natural posible.

Las físicas mínimas requeridas para cada elemento del AV se muestran en la **Figura 31**. El parámetro más común son las articulaciones en los avatares. Las articulaciones permiten definir una estructura (generalmente el sistema óseo), para tener una mejor manipulación de cada sección del avatar. Al definir un estructura como la mencionada en la sección 3.2, es posible generar micro y macro animaciones, aunado a esto, con las físicas el movimiento es más suave y natural.

Contar con una estructura en los avatares permite hacer distinción entre los cuerpos rígidos como podrían ser los objetos 3D y los avatares humanos o de otro tipo. Los objetos 3D son elementos del AV que requieren de un estímulo externo para modificar su comportamiento o su forma. Por ejemplo, si un avatar cae de una altura considerable, su estructura debe reaccionar de acuerdo a las físicas del mundo real, dando realismo a la evolución del ambiente, esto tiene repercusiones en el grado de la QoE brindada.

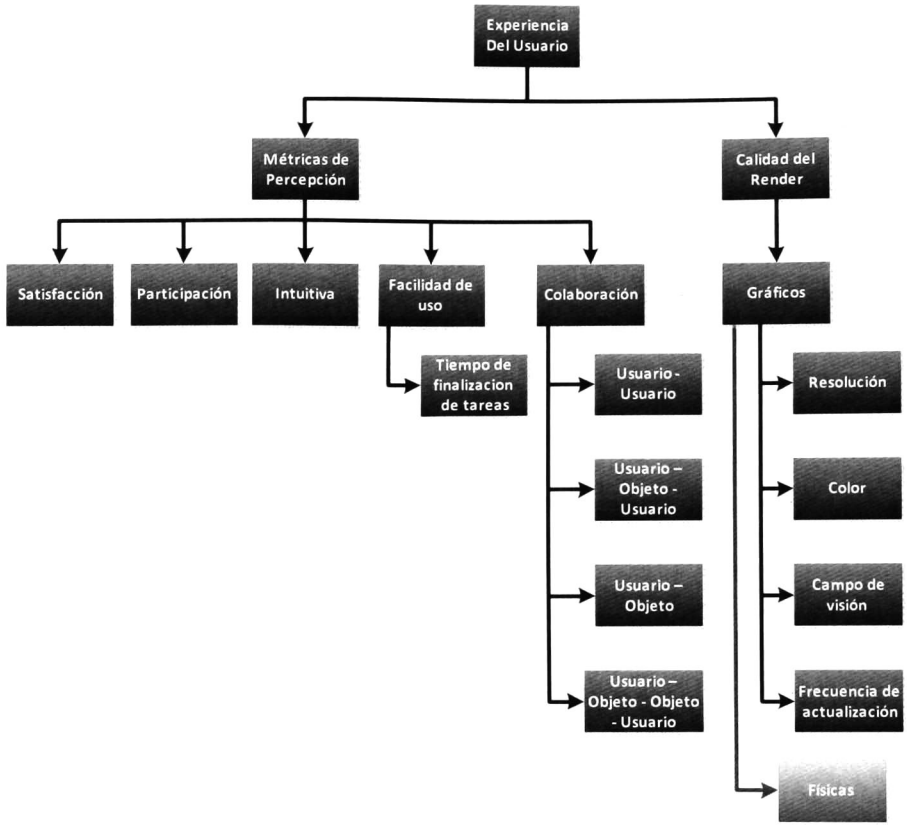


Figura 30: Incorporacion de métricas de Físicas.

La incorporación de las métricas clasificadas en las físicas (ver **Figura 31**), permitirán realizar una manipulación de los elementos del ambiente de una forma más natural, es por ello que se consideran características como: la masa, el tamaño, la inercia, etc. Las métricas de la física tienen como finalidad el evaluar a los elementos de una ambiente considerando su similitud con su correspondiente en el mundo real.

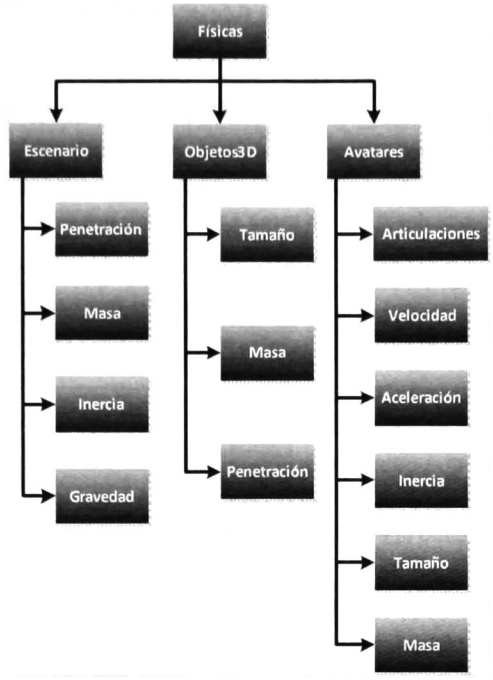


Figura 31: Fisicas en un AV.

4.4 Relación entre QoS y la Experiencia del Usuario

Las métricas consideradas en la Experiencia del Usuario son evaluadas subjetivamente debido a que dependen del usuario final, mientras que en la QoS son objetivamente ya que dependen de las funcionalidades del AV, como el retardo que es afectado por el ancho de banda disponible. En este trabajo de investigación se realizó una relación entre estas con la finalidad de mostrar que características cubren cada métrica y evaluar métricas de la experiencia de los usuarios de forma objetiva, a través de métricas de la QoS.

La relación entre la QoS y la Experiencia del Usuario está dada por las características de la arquitectura $P2P$ de los PVs (**Figura 32**). Las características de la arquitectura $P2P$ propuesta permiten realizar una relación entre la QoS y la Experiencia del Usuario. Estas están numeradas en la **Figura 32** y su descripción se encuentra en la **Tabla 3**. Se considera que al administrar las características de relación (**Tabla 3**) se brinda una QoE aceptable. De esta forma se puede realizar la evaluación de las métricas de Experiencia del Usuario de una forma objetiva.

Posteriormente los resultados de las pruebas objetivas, pueden ser cotejados con resultados de pruebas subjetivas, pruebas que serían diseñadas y analizadas por los expertos

adecuados, con la finalidad de comprobar si es suficiente realizar pruebas objetivas para evaluar la experiencia del usuario.

Las métricas como el audio o las psicológicas se pueden aplicar en los PV, siendo necesario detallar las pruebas a realizar. Dichas pruebas deben ir orientadas a las características que se desean incorporar en los PVs. Debe tenerse en cuenta al momento del diseño de las pruebas que un PV no contempla la lógica del AV, esta tarea es realizada por el modulo del Modelo (basándose en el patrón de arquitectura MVC, **Figura 1**).

Si en un AV se solicita realizar una animación de correr, las pruebas subjetivas realizadas al PV deben valorar: ¿Cómo corre el avatar?, ¿La acción de correr es de forma natural?, ¿El desplazamiento con cada zancada o paso es coherente?

El motivo del por qué debe correr o qué ocasionó que se solicitara correr, no deben ser consideras como parte del PV, porque son tareas de la parte lógica del AV. Es decir, se debe evaluar el cómo se lleva a cabo la evolución del AV y no por qué evoluciona de cierta manera.

Tabla 3: Caracteristicas de relación entre la QoS y la Experiencia del usuario.

	Características de relación	QoS – Experiencia del usuario	Descripción
1)	Consistencia mínima local	Consistencia: Temporal / Métricas de percepción: Satisfacción	Cada PV administra las solicitudes del usuario, para ser ejecutadas según el orden de creación.
2)	Mensajes de consistencia	Consistencia: Espacial / Métricas de percepción: Participación	El AV se mantiene consistente, debido a que cada PV envía el estado local del AV en mensajes de consistencia. Los coordinadores son los encargados de detectar cualquier anomalía.
3)	Representación en 3D	Interactividad: Asignación / Métricas de percepción: Intuitiva	La administración del AV, puede darse de forma sencilla debido a la representación 3D del AV.
4)	Tiempo Real utilizando la cadena de Markov propuesta	Interactividad: Velocidad / Métricas de percepción: Tiempo de finalización de tareas	La frecuencia de cambios en los PVs está definida en base a la vista humana, para dar la impresión de que los cambios son continuos. El retardo del envío de los mensajes es estimado por la cadena de Markov propuesta.

5)	Micro y macro animaciones	Interactividad / Métricas de percepción Rango / Colaboración	El avatar es el elemento más complejo de un AV, debido a su complejidad estructural. Para minimizar el impacto de su administración se definen micro y macro animaciones, para facilitar la colaboración entre los elementos y el usuario.
6)	Visualización de la evolución del AV	Viveza / Calidad del render Amplitud / Gráficos	La representación gráfica del AV permite un análisis fácil de las consecuencias de los cambios en el AV.
7)	Representación en 3D y manejo de físicas	Viveza / Calidad del render Profundidad / Gráficos	Una representación en 3D y la aplicación de físicas, brinda una evolución más apegada a la realidad.

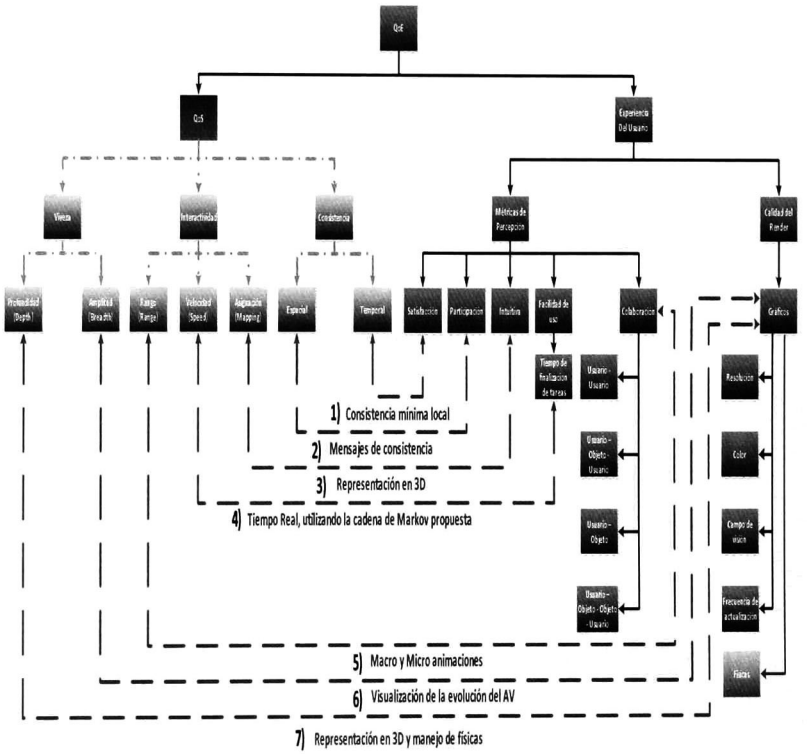


Figura 32: *QoS* - Experiencia del Usuario.

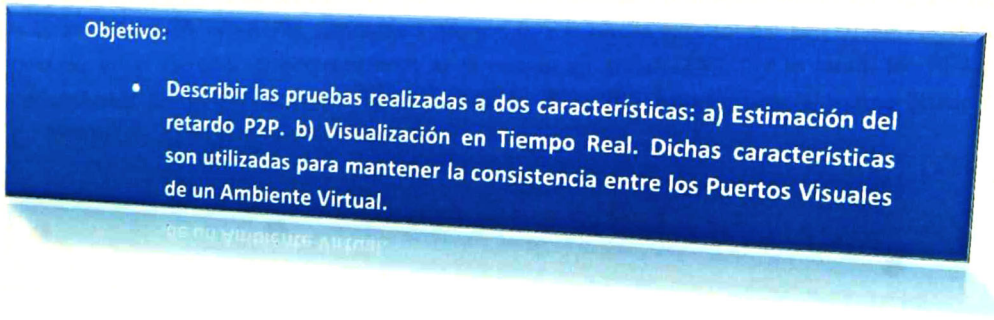
Las métricas fisiológicas requieren de un análisis más detallado, por ejemplo; ¿cómo se detectaría que la frecuencia de respiración ha cambiado solo por la visualización realizada por el PV y no por la lógica del AV?. En este caso, un especialista podría indicar que tipo de actividad física cambia con el trabajo realizado por un PV. Una aproximación sería analizar las pruebas de QoE que se aplican al área de multimedia en especial al video.

4.5 Conclusiones

Un PV es el medio de comunicación del usuario final con el AV, es importante considerar su punto de vista, es decir, el grado de aceptación que tiene el usuario del AV, esto es posible mediante la evaluación de la QoE. Hasta el momento, en la literatura revisada se han analizado las métricas consideradas en un AV inmerso, que incluyen métricas que no son aplicables a un PV. Las métricas clasificadas para un PV sólo evalúan aspectos importantes para la visualización en tiempo real. Es importante considerar la naturalidad para mostrar los cambios en el AV. Esta fue la razón por la cual se decidió agregar la métrica de las físicas y sus características.

Un punto importante para un futuro trabajo es la realización de pruebas subjetivas, diseñadas en conjunto con especialistas en cada área involucrada en el diseño de cada AV específico. Estas pruebas deben de ser tomadas en cuenta para priorizar las métricas y así incrementar el grado de aceptación de la QoE.

Capítulo 5 Consistencia entre Puertos Visuales



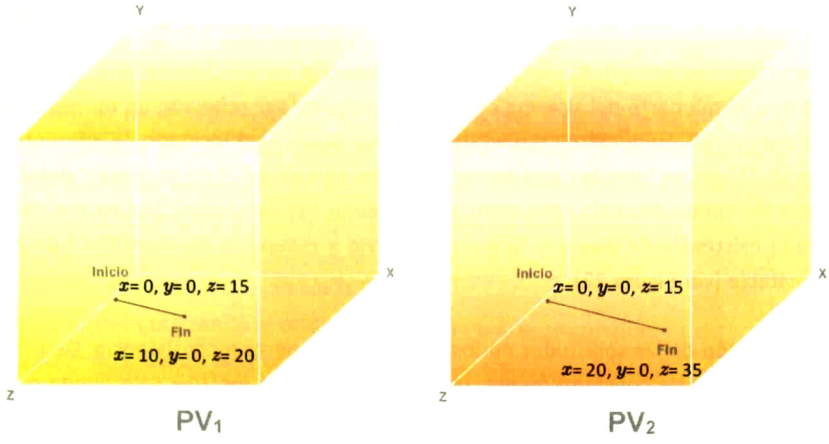


Figura 33: Acciones no finitas.

Tabla 4: Acciones no finitas.

	Puerto Visual	Solicitudes	Posición
Inicio	PV_1	$s_1(a_1, avanza) \xrightarrow{t_0} PV_1$	x 0 y 0 z 15
Fin	PV_1	$s_2(a_1, alto) \xrightarrow{t_3} PV_1$	x 10 y 0 z 20
Inicio	PV_2	$s_1(a_1, avanza) \xrightarrow{t_1} PV_2$	x 0 y 0 z 15
Fin	PV_2	$s_2(a_1, alto) \xrightarrow{t_6} PV_2$	x 20 y 0 z 35

Las acciones finitas (ver **Figura 34**) garantizan que en un determinado tiempo los PVs tendrán el mismo estado del AV. En la **Tabla 5**, se describe que solo es necesario el envió de una solicitud para realizar una animación. El PV_1 ejecuta y envía en t_0 la acción de caminar 3 pasos con el avatar a_1, lo cual es ejecutado en t_1 por el PV_2. Los tiempos y posiciones finales son: t_3, x = 15, y = 0, z = 10 en PV_1 y en PV_2 son t_4, x = 15, y = 0, z = 10. De esta forma, aun cuando existe un desfase en el tiempo, la posición final es la misma en ambos PVs. Por lo tanto, los PVs son consistentes en un determinado tiempo, es decir, PV_1 y PV_2 representarán el mismo Estado del Ambiente (EsA) en tiempos distintos.

$$EsA\ (PV_i) \xleftrightarrow{t_x + \Delta t} EsA\ (PV_{i+n})$$

Para iniciar este capítulo, se define la consistencia en un sistema distribuido como: "en todo nodo que forma parte del sistema distribuido, debe de ser reflejado un mismo estado en un tiempo determinado". En nuestro caso, el sistema distribuido está formado de diferentes *PVs* que conforman el *AV*. Ahora bien, la consistencia entre los *PVs* es difícil de garantizar debido a que en este sistema las condiciones de consistencia que deben ser satisfechas incluyen el factor visual, además de la existencia de diversos tiempos de envío y recepción de mensajes que generan un retardo inevitable (ver **Figura 22**).

La ejecución de las solicitudes en los *PVs* podría generar inconsistencia. Debido a que las solicitudes son recibidas constantemente, la consistencia entre los *PVs* sólo puede ser aceptada en un tiempo t_n. En un tiempo t_{n+1} podría no realizase alguna(s) solicitud(es) por diversas razones: perdida de comunicación, sobrecarga de trabajo, retardo que impida la ejecución de la acción, etc. Es por eso que la consistencia debe ser administrada durante toda la evolución del *AV*.

En la sección 3.4.2 es descrita la secuencia de acuerdos (coordinador – coordinador, nodo simple – coordinador) que realizan los nodos para la administración de la consistencia tanto a nivel de Áreas de Interés como a nivel de *AV*.

En las solicitudes generadas por los usuarios de los *PVs* se consideran acciones con inicio y fin (acciones finitas). Es decir, los avatares quienes son los que realizan las acciones saben las restricciones para iniciar una acción y también conocen cuándo finalizar la acción, por lo que no se requiere de una segunda acción para finalizar la primera. Por ejemplo: se solicita al avatar a_1 camine 3 pasos, cada uno de los *PVs* involucrados sabe cuándo deben de iniciar y cuando terminar dicha acción. De esta forma, se evita que el avatar en los diferentes *PVs* continúe con una acción que pudiera generar inconsistencia.

Las acciones no finitas (ver **Figura 33**) dificultan la adecuada administración de los avatares. Esto es debido a que cuando un *PV* envía una solicitud para finalizar una acción de este tipo, esta llega con un retardo provocado por el tiempo de envío y recepción de los mensajes, es por ello que la duración de las acciones no finitas puede variar de un *PV* a otro. Como se muestra en la **Tabla 4**, el PV_1 realiza la acción de avanzar en t_0 con el avatar a_1, le solicita al PV_2 que mueva al avatar a_1, lo cual se realizan en t_1 en el PV_2. Para finalizar esa acción, el PV_1 genera una solicitud de alto a a_1 realizada en t_3. PV_1 envía a PV_2 la solicitud que contiene la acción de alto, la cual será aplicada al avatar a_1 en t_6. Los tiempos de finalización y las posiciones son diferentes: t_3, x= 10, y=0, z=20 en PV_1 y t_6, x= 20, y=0, z=35 en PV_2. El retardo en la segunda solicitud genera un desplazamiento mayor en el PV_2, por lo cual, los *PVs* involucrados son inconsistentes.

5.1 Estimación del retardo P2P

La finalidad de la cadena de Markov propuesta (ver **Figura 18**) es estimar el retardo que sufrirán los mensajes en la comunicación *P2P*. Para realizar la validación de la cadena, se desarrolló un prototipo de la capa de distribución de actualizaciones. Dicho prototipo tiene las funciones básicas para la comunicación entre los nodos, las cuales son:

- Establecer una comunicación vía TCP y UDP.
- Sincronización utilizando el protocolo *NTP*.
- Creación de cadenas aleatorias.
- Tamaño aleatorio de las cadenas.
- Compresión con el algoritmo de Huffman.
- Estimación del retardo utilizando la distribución de Pareto.

En la **Figura 35** se muestra el escenario utilizado para la validación de la cadena de Markov propuesta.

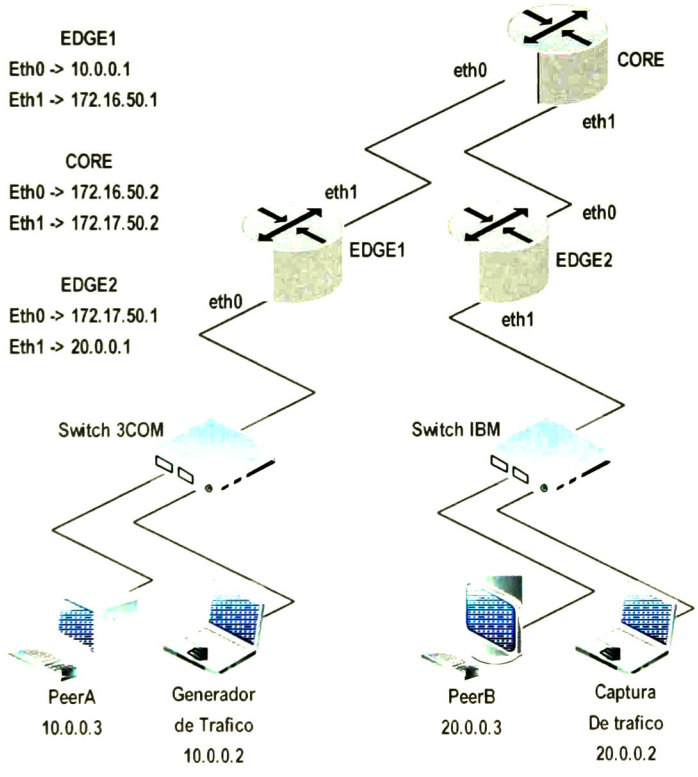


Figura 35: Escenario para la validación del proceso del manejo de la información.

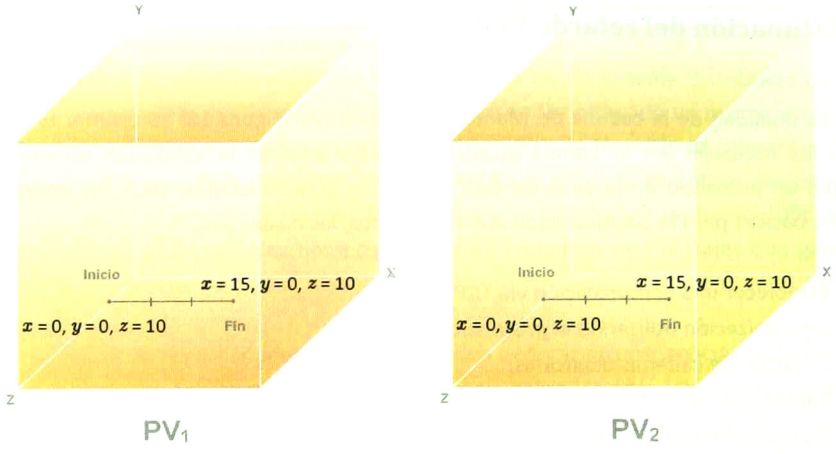


Figura 34: Acciones finitas.

Tabla 5: Acciones finitas.

	Puerto Visual	Solicitudes	Posición	
Inicio	PV_1	$s_1(a_1, camina\ 3\ pasos) \xrightarrow{t_0} PV_1$	x	0
			y	0
			z	10
Fin	PV_1	t_3	x	15
			y	0
			z	10
Inicio	PV_2	$s_1(a_1, camina\ 3\ pasos) \xrightarrow{t_1} PV_2$	x	0
			y	0
			z	10
Fin	PV_2	t_4	x	15
			y	0
			z	10

Para comprobar de forma objetiva la consistencia entre los PV, se definen dos características escenciales:

Estimación del retardo P2P: el TPU es estimado mediante una cadena de Markov, la cual indica si la solicitud llegará en un tiempo TPU.

- ***Visualización en tiempo real:*** se verifica que las solicitudes enviadas entre los PVs se realicen en un determinado tiempo, dado por el TTA.

Estas características son descritas en las siguientes secciones, así como los escenarios implementados para las pruebas y los resultados obtenidos.

Tabla 8: Configuración del nodo EDGE2.

EDGE2	
BGPD.CONF	**ZEBRA.CONF**
hostname BGP_Edge2	hostname Edge2
password zebra	password zebra
	enable password zebra
router bgp 3	
bgp router-id 3.3.3.3	interface lo
network 20.0.0.0/24	description Edge2 loopback
network 172.17.50.0/24	
neighbor 172.17.50.2 remote-as 2	interface eth0
	description wan
log stdout	ip address 172.17.50.1/24
	interface eth1
	description lan
	ip address 20.0.0.1/24
	ip route 0.0.0.0/0 172.17.50.2
	interface eth2
	shutdown
	interface eth3
	shutdown
	line vty

Se estimó el parámetro de Hurst del tráfico cruzado generado en la red, con la finalidad de mostrar que es auto-similar, se ha demostrado que el tráfico de internet sigue este patrón. Para que el tráfico sea auto-similar, el parámetro de Hurst debe estar en el siguiente rango:

$$\widehat{H} \in [0.5, 1.0)$$

El tráfico cruzado fue generado utilizando la aplicación llamada TrafGen [76], la cual brinda la facilidad de generar tráfico según la distribución de Pareto [77]. Con base en [78], cuando el tráfico es generado conforme a una distribución de Pareto, este contiene un parámetro de Hurst [79] aceptable, lo cual indica que es auto-similar. La función de densidad de la distribución de Pareto es:

$$f(x) = \begin{cases} \frac{ab^a}{x^{a+1}} & si\ x \geq b \\ 0 & si\ x < b \end{cases}$$

El tráfico cruzado está formado por 20 fuentes UDP con los siguientes parámetros para la distribución de Pareto:

Tabla 9: Parámetros para el tráfico cruzado.

Parámetro	Valor
Tamaño del paquete	1024

Los nodos EDGE1, EDGE2 y CORE están implementados en el sistema operativo Red Hat 7.2 [**73**]. Los nodos utilizan el software Zebra [**74**] para la administración de las tablas de ruteo. La configuración en cada uno de los nodos se realizó con el protocolo de ruteo BGP (Border Gateway Protocol) [**75**] que permite el intercambio de tablas de rutas entre los routers externos de cada sistema autónomo, esto en el archivo BGPD.CONF. En el archivo ZEBRA.CONF se configuraron las tarjetas de red. Las configuraciones de los nodos se muestran en las siguientes tablas.

Tabla 6: Configuración del nodo EDGE1.

EDGE1	
BGPD.CONF	**ZEBRA.CONF**
hostname BGP_Edge1	hostname Edge1
password zebra	password zebra
	enable password zebra
router bgp 1	interface lo
bgp router-id 1.1.1.1	description Edge1 loopback
network 10.0.0.0/24	interface eth0
network 172.16.50.0/24	description wan
neighbor 172.16.50.2 remote-as 2	ip address 10.0.0.1/24
	interface eth1
	description lan
	ip address 172.16.50.1/24
	ip route 0.0.0.0/0 172.16.50.2
	interface eth2
	shutdown

Tabla 7: Configuración del nodo CORE.

CORE	
BGPD.CONF	**ZEBRA.CONF**
hostname BGP_Core	hostname Core
password zebra	password zebra
log stdout	enable password zebra
router bgp 2	interface lo
bgp router-id 2.2.2.2	description Core loopback
	interface eth0
network 172.16.50.0/24	description wan
neighbor 172.16.50.1 remote-as 1	ip address 172.16.50.2/24
network 172.17.50.0/24	
neighbor 172.17.50.1 remote-as 3	interface eth1
	description wan
	ip address 172.17.50.2/24
	ip route 0.0.0.0/0 172.17.50.1
	ip route 0.0.0.0/0 172.16.50.1
	interface eth2
	shutdown
	line vty

Una vez que se obtuvieron las características del tráfico cruzado, se validó que efectivamente el retardo End-To-End siguiera una distribución de Pareto, para lo cual se implementó el protocolo Network Time Protocol (*NTP*), que se encuentra descrito en RFC 778 [81], RFC 891 [82], RFC 956 [83], RFC 958 [84] y RFC 1305 [85]. Dicho protocolo fue utilizado para la sincronización de los nodos.

Las características de los mensajes generados, tales como el tamaño y la frecuencia, fueron generadas de forma aleatoria, para poder simular parte del comportamiento de la capa de distribución de actualizaciones, que forma parte de la arquitectura propuesta en la sección 3.4. Antes de ser enviadas las actualizaciones, es calculada una marca de tiempo en milisegundos en base a la fecha de la época [86], y a su vez es colocada en el mensaje a enviar. Al ser recibida la solicitud por el segundo nodo, se calcula una nueva marca de tiempo que indique el tiempo de llegada del mensaje (solicitud). Las marcas de tiempo tanto de envió como de recepción son almacenados en un archivo utilizado para el cálculo del retardo entre los nodos.

Se estimó la distribución de los retardos obtenidos, utilizando pruebas de bondad [87] para el cálculo de los parámetros en cada distribución probada. Uno de los algoritmos para la prueba de bondad es Anderson-Darling [88] (ver **Tabla 11**), por medio del cual se comprobó que los retardos si seguían una distribución de Pareto (ver **Figura 37** y **Tabla 12**).

Tabla 11: Parámetros de la prueba de bondad de Anderson-Darling.

Tamaño de la muestra	47092				
α	0.2	0.1	0.05	0.02	0.01
Valor crítico	1.3749	1.9286	2.5018	3.2892	3.9074
¿Rechazar?	No	No	No	No	No

La fórmula de Anderson-Darling determina si un conjunto de datos ordenados, vienen de una distribución con función acumulativa [89], la fórmula para saberlo es la siguiente:

$$A^2 = -n - \left(\frac{1}{n}\right) \sum_{1} [(2i-1)\ln(p_{(i)}) + (2n+1-2i)\ln\{1 - p_{(i)}\}]$$

Tabla 12: Parámetros para la distribución de Pareto.

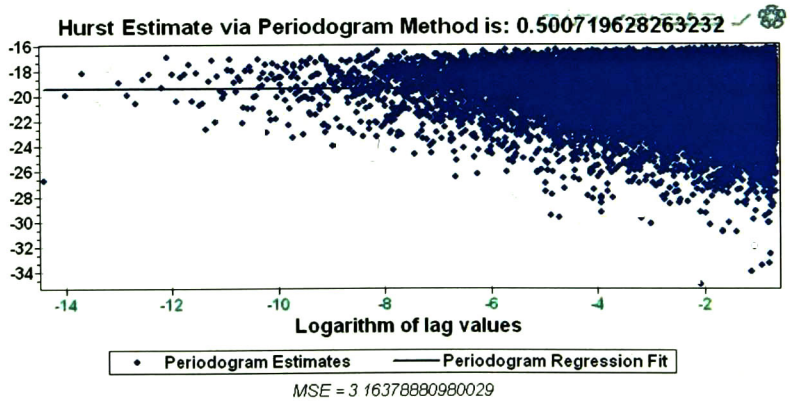
Parámetro	Valor
α	0.49268
β	1

β	10
α	1.5

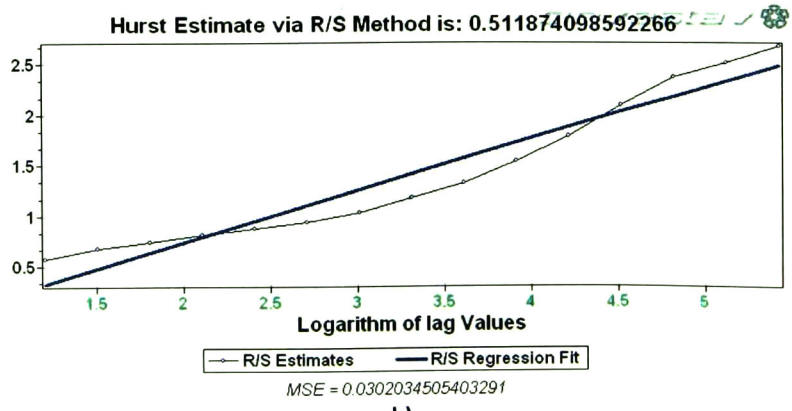
La **Figura 36.a** muestra que por medio de periodograma el parámetro de Hurst tiene un valor de 0.5007, mientras que en la **Figura 36.b** el método R/S da un parámetro de Hurst con valor de 0.5118. Se obtuvo el parámetro de Hurst a través de la herramienta nombrada SelQoS [**80**], los valores son mostrados en la **Tabla 10**.

Tabla 10: Parámetro de Hurst.

Método	Parámetro de Hurts
Periodograma	0.5007
R/S	0.5118



a)



b)

Figura 36: Estimación del parámetro de Hurst.

Tabla 13: Algoritmo de $Huffman$.

Entradas:	El alfabeto $A = \{a_1, a_2, \cdots, a_n\}$, que es un conjunto de símbolos de tamaño n. El conjunto $W = \{w_1, w_2, \cdots, w_n\}$, que es el conjunto de pesos (positivos) de los símbolos (normalmente proporcionales a probabilidades), es decir, $w_i = peso\ (a_i), 1 \leq i \leq n$.
Salida:	El código $C(A, W) = \{c_1, c_2, \cdots, c_n\}$, que es el conjunto de elementos del código (binario), donde c_i es la palabra del código para $a_i, 1 \leq i \leq n$.
Objetivo:	Sea $L(C) = \sum_{i=1}^{n} w_i \times longitud(c_i)$ la longitud del camino ponderado del código C.
Condición:	$L(C) \leq L\ (T)$ para cualquier código $T(A, W)$.

La validación del proceso del manejo de la información de la capa de distribución de la arquitectura propuesta se realiza mediante la implementación de la máquina de estados, la cual realiza la estimación del retardo en el estado q_2 y la si es necesaria la compresión en el estado q_5. Cuando se llega al estado q_3 se toma el ultimo retardo de los mensajes de sincronización del protocolo NTP, para estimar el retardo del mensaje a enviar. Si se estima que el retardo será mayor al rango de tiempo definido como MR, los mensajes son comprimidos. De esta forma se monitoreo el retardo antes y después de la utilización de la máquina de estados. En la **Tabla 14** se muestran los resultados obtenidos, al utilizar la estimación y la compresión de los mensajes se redujo en un 36.82% el retardo promedio de una muestra de 47, 369 mensajes enviados.

Tabla 14: Resultados de aplicación la estimación del retardo y la compresión.

Sin la estimación y la compresión		Con la estimación y la compresión	
Tamaño de la muestra	47369	Tamaño de la muestra	47369
Retardo Promedio	74.4436	Retardo Promedio	47.0324

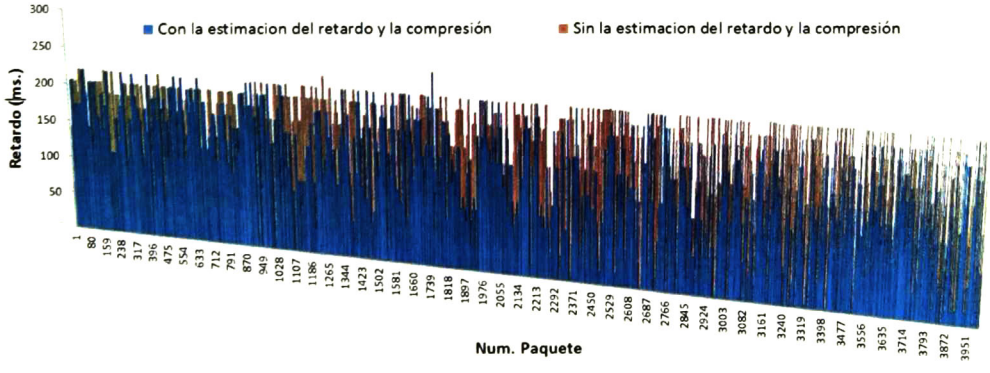


Figura 38: Retardo de los mensajes.

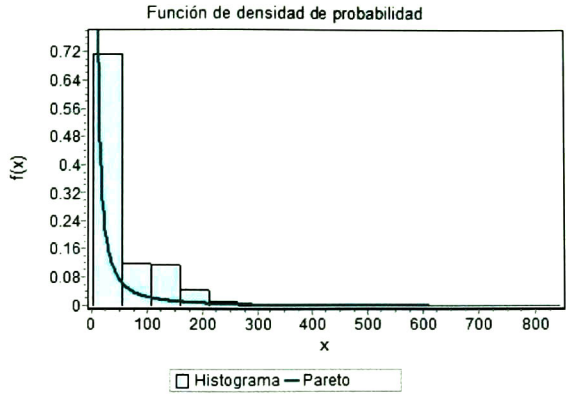


Figura 37: Distribución de Pareto del retardo en una comunicación $P2P$.

Una vez que fue validado que el retardo sigue una distribución de Pareto, se procedió a realizar la implementación del mecanismo de compresión. La compresión de los datos fue realizada de acuerdo a los perfiles de los usuarios mencionados en la **Tabla 1**. Para lo cual, se toma una marca de tiempo en el instante de envió del primer mensaje, de esta forma cuando se cumple la duración de cada perfil de usuario se cambia la probabilidad del retardo correspondiente. La compresión de los mensajes es realizada cuando se estima que el envío requerirá más tiempo del debido.

Los métodos de compresión se clasifican según los algoritmos utilizados, estos pueden ser sin pérdidas (lossless) y con pérdidas (lossy) [**90**]:

Métodos sin pérdidas: en estos métodos la información descomprimida es exactamente igual a la que había antes de comprimir. Son utilizados cuando la información no pueden tener ningún dato diferente, como en: archivos con información en binario, código ejecutable, archivos de texto, etc. Algunos ejemplos de algoritmos de compresión sin pérdidas son: Run-Length, Huffman, delta, Lempel-Ziv-Welch.

- ***Métodos con pérdidas:*** en estos la información sufre una pequeña modificación al realizar la descompresión, siendo esta la mayoría de las veces imperceptible. Este método de compresión es utilizado para comprimir imágenes, vídeo y música, ya que, aunque perdamos algún dato no perderemos gran información. Algunos ejemplos de algoritmos de compresión con pérdidas son: JPEG, MPEG, MP3.

Debido a que es necesario obtener los mensajes tal como fueron generados la compresión de las solitudes se realizó con el algoritmo de $Huffman$ [**91**], el cual se describe a continuación:

banda disponible		
Retardo	$\frac{8192\ bits}{1048576\ bits/seg} = 0.0078125$ seg $= 7.8$ mseg	$\frac{40960\ bits}{1048576\ bits/seg} = 0.0390625$ seg $= 39.06$ mseg

La compresión de los mensajes no se realiza todo el tiempo para no restar recursos (memoria, tiempo de CPU, etc.) a la visualización. Un escenario donde se desean realizar pruebas futuras, es uno donde se tengan recursos de hardware (como memoria RAM, tarjetas de video) suficientes para no restar recursos a la visualización del *AV*.

5.2 Visualización en Tiempo Real

Para la visualización en *TR* entre los *PVs* se utiliza una arquitectura *P2P*. Al igual que en la comunicación se considera suficiente el analizar el comportamiento de dos *PVs*. Para ello se asume que el resto de los *PVs* tendrán el mismo comportamiento [**24**] [**92**] [**93**].

Si un PV_i emite una solicitud al PV_j, en ambos *PVs* se debe realizar la visualización en *TR*. Una actualización originada en PV_i se realiza en t_o y en PV_j en $t_o + \Delta t$. El tiempo Δt en que se refleja el nuevo *EsA* en PV_j depende del *TTA*, que considera el tiempo definido como Tiempo Real (sección 0) y el Tiempo de los Perfiles de Usuario (**Tabla 1**), es decir:

$$t_y + \Delta t \leq TTA$$

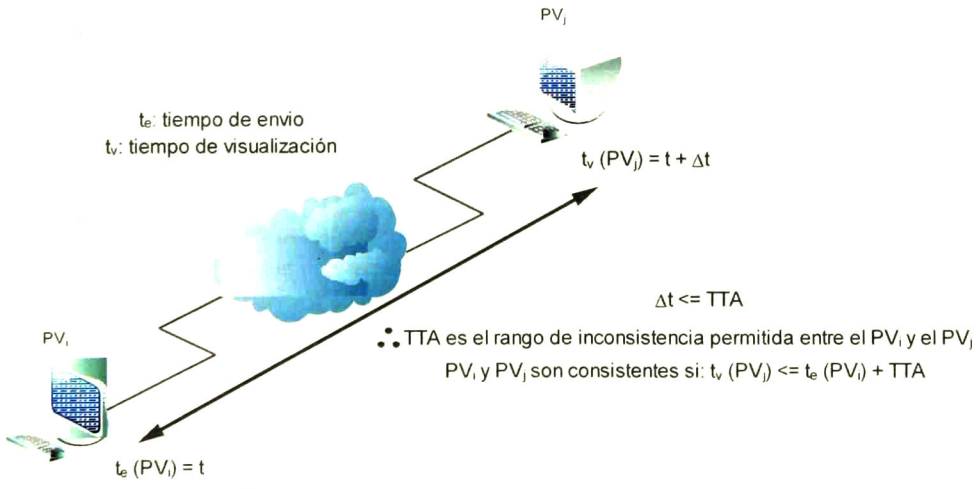


Figura 40: Consistencia entre *PVs*.

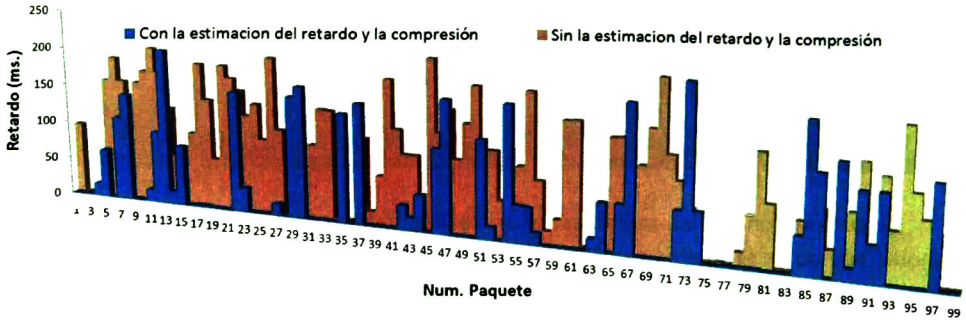


Figura 39: Comparación de retardos los mensajes.

En la **Figura 38**, se gráfica una muestra de 4000 retardos, los cuales fueron tomados con el uso (azul, al frente) y sin el uso (rojo) de la estimación del retardo y la compresión. Es posible mostrar que su utilización disminuye el retardo en la mayoría de los envíos de mensajes. En la **Figura 39**, el tamaño de la muestra es de 100, lo cual deja ver con mayor claridad que los paquetes en los cuales se utilizó la estimación y compresión, sufren un menor retardo que los paquetes en los que no se utilizó. Por lo tanto, a partir de los resultados obtenidos, se considera útil el uso de la estimación y compresión, dando la pauta para asumir que el uso de la cadena simple de Markov propuesta es útil para la administración de la información entre los PVs.

$$retardo\ promedio\ CON\ la\ estimacio\ y\ compresión \ll retardo\ promedio\ SIN\ la\ estimacio\ y\ compresión$$

Cuando se estima que los mensajes tendrán un retardo mayor al permitido (sección 3.4.3), se realiza una compresión. Los mensajes comprimidos tienen un retardo menor que el resto de los mensajes, como es descrito en la sección 3.4.3.

$$\frac{Tamaño\ del\ mensaje}{Ancho\ de\ banda\ disponible} \approx retardo$$

El retardo es aproximado debido a que no se consideran los tiempos de propagación, serialización y encolamiento. En la **Tabla 15** se muestra un ejemplo del retardo en el envío de mensajes, cuando se envía un mensaje de 1024 bytes con un ancho de banda disponible de 1 mb el retardo es de 7.8 mseg., mientras que en un mensaje de 5120 bytes con el mismo ancho de banda disponible el retardo obtenido es de 39.06 mseg.

Tabla 15: Retardo en el envió de mensajes.

Mensaje	$1024\ bytes * 8 = 8192\ bits$	$5120\ bytes * 8 = 40960\ bits$
Ancho de	$1\ mb = 1024\ kb = 1048576\ {}^{bits}/_{seg}$	$1\ mb = 1024\ kb = 1048576\ {}^{bits}/_{seg}$

El tráfico cruzado contiene las mismas características descritas en la sección anterior. Se tomó el tiempo de envió en el PV_i y el tiempo de ejecución en el PV_j (Figura 41), obteniendo el TTA de las siguientes tablas:

Tabla 16: Muestras para el Perfil Bajo (PB).

Tipo de Perfil	Tamaño de la muestra	Visualización en Tiempo Real		Transmisión Adaptable	
PB	50,000	$TTA <= t_v$	42,208	Estimación > Perfil	8,855
				Estimación <= Perfil	33,353
		$TTA > t_v$	7,792	Estimación > Perfil	2,058
				Estimación <= Perfil	5,734

Tabla 17: Porcentajes para PB.

Tipo de Perfil	Tamaño de la muestra	%			
PB	50,000	Consistente	84.416	Estimación > Perfil	20.97943518
				Estimación <= Perfil	79.02056482
		Inconsistente	15.584	Estimación > Perfil	26.41170431
				Estimación <= Perfil	73.58829569

Como se mencionó al inicio de esta sección, cada acción debe ser ejecutada en un rango de tiempo dado por TTA para que los PVs se consideren consistentes. La **Tabla 16** y la **Tabla 17** muestran que durante el Perfil Bajo los PVs son consistentes en el 84.416%. Es decir, de la muestra tomada 42,208 acciones fueron ejecutadas en un tiempo menor o igual al TTA perteneciente a dicho perfil.

Tabla 18: Muestras para el Perfil Medio (PM).

Tipo de Perfil	Tamaño de la muestra	Visualización en Tiempo Real		Transmisión Adaptable	
PM	50,000	$TTA <= t_v$	24,895	Estimación > Perfil	10,100
				Estimación <= Perfil	14,795
		$TTA > t_v$	25,105	Estimación > Perfil	10,211
				Estimación <= Perfil	14,894

Tabla 19: Porcentajes para PM.

Tipo de Perfil	Tamaño de la muestra	%			
PM	50,000	Consistente	49.79	Estimación > Perfil	40.57039566

En los *PVs* es posible tolerar un rango de inconsistencia en cuanto al tiempo de visualización, el cual depende del *TTA*. Como se muestra en la **Figura 40**, el tiempo de visualización en un segundo *PV* debe ser el tiempo de envío más un incremento de tiempo. El incremento de tiempo debe ser menor o igual al *TTA* considerado para cada perfil. Por lo tanto el rango de inconsistencia permitido entre los *PVs* está dado por *TTA*, entonces se dice que los *PVs* involucrados serán consistentes.

El PV_i (emisor) a través de la cadena de Markov propuesta realiza una estimación del retardo de los mensajes, que permite valorar si es necesaria una compresión del mensaje. De esta forma, el manejo de la información generada depende de cada *PV*, lo cual le permite adaptarse al estado de la red, ya que se considera que el estado de la red es diferente del PV_i al PV_j ($PV_i \rightarrow PV_j$) y viceversa ($PV_i \leftarrow PV_j$). Para verificar que los *PVs* son consistentes se utilizaron las siguientes características:

- Se continuo estimando el retardo que podrían sufrir el envío de los mensajes, para lo cual se utilizó la cadena de Markov de la sección anterior,
- La comunicación entre los *PVs* se realizó a través de una red de enrutadores en el simulador GNS_3 (**Figura 41**),
- Para generar saturación de la red se generó trafico cruzado,
- Se tomaron los tiempos de envío y de ejecución.

Los enrutadores se configuraron con el Protocolo de Enrutamiento de Información (Routing Information Protocol – *RIP* [**94**]), la interconexión se realizó de forma aleatoria, cuidando que se tuviera más de un camino para llegar al destino. Cabe mencionar que las pruebas objetivas no se realizaron a nivel de enrutadores, sino en los *PVs* involucrados.

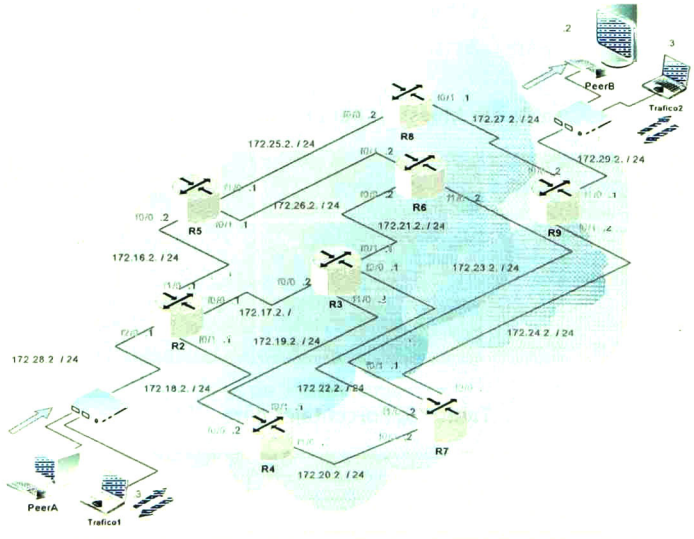


Figura 41: Estructura de la red de enrutadores.

Tabla 23: Porcentajes para todos los perfiles.

Tamaño de la muestra	%		
150,000	Consistente 69.151	Estimación > Perfil	27.11058837
		Estimación <= Perfil	72.88941163
	Inconsistente 30.849	Estimación > Perfil	36.46186761
		Estimación <= Perfil	63.53813239

La **Tabla 22** y **Tabla 23** muestran que 103,727 de las muestras tomadas tuvieron un TTA menor o igual al t_v, permitiendo una visualización en tiempo real, es decir, el 69.15% de las muestras fueron consistentes. Durante el perfil PM se obtuvo mayor inconsistencia, debido a que el rango de tiempo es reducido. Se tomó un nuevo conjunto de muestras donde en dicho perfil se comprimieran todas las solicitudes, los resultados se muestran en las siguientes tablas:

Tabla 24: Muestras para PB, solicitudes comprimidas.

Tipo de Perfil	Tamaño de la muestra	Visualización en Tiempo Real	
PM	50,000	$TTA <= t_v$	25,076
		$TTA > t_v$	24,924

Tabla 25: Porcentajes para PB con solicitudes comprimidas.

Tipo de Perfil	Tamaño de la muestra	%	
PM	50,000	Consistente	50.152
		Inconsistente	49.848

De la **Tabla 25** podemos ver que comprimiendo las solicitudes durante el perfil PM se obtiene el 50.15% de consistencia, aun así la diferencia entre la consistencia e inconsistencia es mínima (0.304%). Por lo cual, como trabajo futuro se analizarán métricas que permitan reducir el retardo, dichas métricas deben ser de complejidad baja para reducir el impacto negativo que pudiera tener en la QoE al brindar una visualización lenta. Al considerar el TTA y utilizar la cadena de Markov, se realiza un CLD de calibración vertical entre capas, debido a que desde la capa de aplicación se toman en cuenta las repercusiones que se tendrían a nivel de red, reduciendo el retardo que afecta a una visualización en tiempo real.

5.3 Conclusiones

Como se mencionó anteriormente para obtener una ejecución en Tiempo Real (en nuestro caso: Visualización en Tiempo Real) se debe sacrificar la consistencia total. Con los resultados obtenidos es posible afirmar que el grado de inconsistencia puede variar entre el 15% y el 49%,

			Estimación <= Perfil	59.42960434
		Inconsistente 50.21	Estimación > Perfil	40.67317267
			Estimación <= Perfil	59.32682733

La consistencia e inconsistencia durante el Perfil Medio son muy similares (ver **Tabla 18** y **Tabla 19**). El 49.79% de las acciones generaron consistencia en los PVs, de las cuales el 40.57% se estimó que su retardo seria mayor al requerido, por lo cual fueron comprimidas.

Tabla 20: Muestras para el Perfil Alto (PA).

Tipo de Perfil	Tamaño de la muestra	Visualización en Tiempo Real		Transmisión Adaptable	
PA	50,000	$TTA <= t_v$	36,624	Estimación > Perfil	9,166
				Estimación <= Perfil	27,458
		$TTA > t_v$	13,376	Estimación > Perfil	4,603
				Estimación <= Perfil	8,773

Tabla 21: Porcentajes para PA.

Tipo de Perfil	Tamaño de la muestra	%			
PA	50,000	Consistente	73.248	Estimación > Perfil	25.0273045
				Estimación <= Perfil	74.9726955
		Inconsistente	26.752	Estimación > Perfil	34.41238038
				Estimación <= Perfil	65.58761962

En la **Tabla 20** podemos ver los resultados del Perfil Alto, lo cual muestra que 36,624 acciones se realizaron en un tiempo menor o igual al TTA, de las cuales 9,166 se comprimieron porque se estimó que su retardo impediría una visualización en tiempo real. En la **Tabla 21** se indica que el 73.248% de las acciones generaron consistencia.

Tabla 22: Muestras considerando todos los perfiles.

Tamaño de la muestra	Visualización en Tiempo Real		Transmisión adaptable	
150,000	$TTA <= t_v$	103,727	Estimación > Perfil	28,121
			Estimación <= Perfil	75,606
	$TTA > t_v$	46,273	Estimación > Perfil	16,872
			Estimación <= Perfil	29,401

Capítulo 6 Auto-Organización de nodos

Objetivo:

- Describir los diferentes algoritmos de Auto-Organización utilizados para organizar los nodos donde se ejecutan los Puertos Visuales de un Ambiente Virtual. Los algoritmos generan una arquitectura de red adaptativa a fin de reducir el retardo y administrar la consistencia en el Ambiente.

obteniendo el mayor grado de inconsistencia durante el Perfil Medio. El Perfil Medio tiene una duración de nueve minutos, por lo cual se considera no tiene un impacto negativo considerable en la QoE brindada. La realización de pruebas subjetivas podría indicar de forma confiable si el rango de inconsistencia en el Perfil Medio afecta negativamente la QoE.

Para disminuir el rango de inconsistencia es necesario el análisis y utilización de métricas que permitan reducir tanto el retardo en la red como en la visualización. Para reducir el retardo en la red podrían realizarse: a) la compresión de todos los mensajes, b) en el algoritmo de compresión, utilizar una tabla predefinida con los pesos de los símbolos necesarios, y c) un buffer con la compresión de las solicitudes que se detecten son más utilizadas. Estos mecanismos podrían ser seleccionados dependiendo de la carga de trabajo del PV y del retardo detectado. El retardo en la visualización podría disminuirse al utilizar las ventajas de los $GPUs$ (Graphics Processing Unit [**95**]) de las tarjetas gráficas, que permitan realizar un rendereo de manera más eficiente.

Los GPU son procesadores dedicados al procesamiento gráfico u operaciones de números de punto o coma flotante. Con su uso, se puede aligerar la carga de trabajo del procesador central en aplicaciones 3D. De esta forma, el procesador central es utilizado para cálculos independientes a cuestiones 3D.

La elección del coordinador en cada AI se lleva a cabo mediante un algoritmo distribuido que se ejecuta en cada nodo de la AI de la siguiente forma (**Figura 42**):

1. Enviar un mensaje inicial a los nodos del AI.
2. Calcular la distancia promedio hacia el resto de los nodos del AI, por medio del mensaje inicial recibido de los nodos restantes del AI. Este procedimiento es descrito en la sección 3.4.2.
3. Tomando en cuenta el orden lexicográfico se toma la dirección mayor de la tabla de nodos del AI, el cual será el coordinador temporal.
4. Si la dirección local no coincide con la del coordinador temporal se envía al coordinador temporal un mensaje que contenga la distancia promedio hacia el resto de los nodos.
5. El coordinador temporal selecciona al nodo con la menor distancia promedio y lo establece como *coordinador* del AI.
6. El coordinador temporal envía un mensaje al resto del grupo indicando que nodo será el *coordinador* del AI.
7. El grupo entero está enterado del nuevo *coordinador* y cada nodo le envía su distancia promedio.
8. El nodo *coordinador* agrega a todos los que le envíen su distancia promedio.

Nota: En los algoritmos siguientes, los números en color rojo en las llaves indican los pasos descritos para su secuencia de ejecución.

ElegirCoordinador: Una vez detectadas las AI se procede a la elección del coordinador, el cual es el encargado de administrar la consistencia en la AI.

```
ElegirCoordinador()
    Local: integer distancia_promedio_menor, mensaje mensaje_inicial, IP nodosAI[],
           mensaje mensaje_inicial_recibidos[], mensaje mensajes_recibidos[]
 1:  Enviar(AI[], mensaje_inicial)
 2:  mensaje_inicial_recibidos[] ← Recibir()
 3:  Agregar (nodosAI[], mensaje_inicial_recibidos[])
 4:  for all m ∈ nodosAI[] do
 5:      Integer i ← 0
 6:      distancia_promedio[i] ← (m.ttl/nodosAI.size())
 7:      if distancia_promedio < distancia_promedio_menor then
 8:          distancia_promedio_menor ← distancia_promedio
 9:      end if
10:      i ← i + 1
11:  end for
12:  Qsort(nodosAI[])
13:  if LocalIP not equals nodosAI[0] then
14:      Enviar(coordinador, distancia_promedio[])
15:      coordinador ← Recibir()
16:  else
17:      Qsort(distancia_promedio[])
18:      coordinador ← distancia_promedio[0]
19:      Enviar(AI[], coordinador)
```

Un AV puede estar formado por n Áreas de Interés o AIs con $n \in \aleph$. Cuando son creadas las AIs los nodos son agrupados, como es descrito en la sección 3.4.2. En cada AI existe un nodo coordinador que administra su consistencia. La auto organización de los nodos se lleva a cabo en base a dos características: a) la distancia promedio entre los nodos, b) la carga de trabajo. Se diseñaron algoritmos, que permiten modificar la arquitectura de red de tal manera que se adapte a las necesidades del AV con el objetivo de brindar una QoE aceptable. Los algoritmos son descritos en las siguientes secciones. Se asume que los nodos que ejecutan un PV pertenecientes al AV se conocen entre sí. Para el manejo de conexiones y desconexiones se diseñaron algoritmos con la finalidad de probar la auto organización propuesta.

En las siguientes secciones se describen los algoritmos necesarios para la auto organización y su diagrama de flujo correspondiente. Se describe brevemente el propósito del algoritmo y posteriormente se muestra la secuencia de pasos que es ejemplificada en pseudocódigo.

6.1 Elección del coordinador

El coordinador es el nodo con la menor distancia promedio en el AI, para lo cual cada nodo calcula su distancia promedio hacia cada uno de los nodos restantes. Cada nodo deberá informar su distancia promedio, aquel cuya distancia promedio sea menor será elegido como el coordinador.

Para el cálculo de la distancia promedio se utiliza el campo TTL del encabezado IP de los paquetes, es decir, el número de brincos (hops) necesarios para que un paquete llegue desde el origen al destino.

La reorganización de los nodos se realiza cuando se detecta que el nodo coordinador tiene una sobrecarga de trabajo, entonces se procede a crear un nodo coordinador (llamémosle secundario) con el objetivo de aligerar la carga de trabajo. Para la elección de este nuevo coordinador se elige el nodo con la segunda menor distancia promedio.

Al coordinador secundario se le asignan los nodos cuya distancia hacia él sean las menores, la cantidad de nodos asignados debe ser equitativa entre los coordinadores. Las desconexiones son consideradas cuando el nodo indica que realizará una desconexión. Cuando se genere una desconexión voluntaria los coordinadores evalúan si es posible llevar a cabo una reorganización para eliminar un grupo, así decrementar los mensajes y la carga de trabajo, obteniendo una arquitectura de red adaptativa.

6.1.1 Algoritmo de elección de coordinador

Aun cuando los coordinadores forman parte de un grupo, estos se comportan de una forma independiente, es decir, cada *coordinador* se encarga de monitorear su carga de trabajo. De esta forma se reduce el número de tareas realizadas y con ello se minimiza la dependencia hacia otros nodos (coordinadores o nodos simples). Cuando un *coordinador* detecta que está en desuso verifica si algún *coordinador* del AI o del AV puede administrar los nodos que tiene asignados, para lo cual realiza los siguientes pasos:

1. Enviar un mensaje al (los) *coordinador*(es) del AI notificando que su carga de trabajo ha disminuido.
2. El *coordinador* secundario recibe los identificadores de los demás coordinadores disponibles.
3. Si el número de coordinadores del AI es mayor a uno envía un mensaje de carga baja a todo su grupo informando acerca de los posibles coordinadores que podrían remplazarlo.

DesusoCoordinador: El coordinador elegirá otro *coordinador* que administre su grupo.

```
DesusoCoordinador()
    Local: mensaje mensaje_desuso, IP Coordinadores[], IP coordinadores_disponibles[]
1   1: Enviar(Coordinadores[], mensaje_desuso)
2   2: coordinadores_disponibles[] ←Recibir()
    3: if coordinadores_disponibles.size() > 1 then
3   4:     Enviar(AI[], coordinadores_disponibles[])
    5: End if
```

Cuando el *coordinador* secundario envía el mensaje de desuso los nodos simples realizan el siguiente procedimiento.

1. Una vez recibido el mensaje de desuso o carga baja, el nodo elige al *coordinador* más cercano a él. Se establece $k=0$, k es el número de intentos de integrarse al grupo.
2. El *nodo simple* envía un mensaje de petición de integración, e informa su distancia promedio.
3. El nodo simple espera un mensaje de confirmación de integración, si recibe la confinación actualiza la dirección del *coordinador*, si no recibe la confirmación realiza el paso 4.
4. Si k es menor que $kmax$ vuelve al paso 2. En otro caso realiza el paso 5.
5. El nodo elige al siguiente *coordinador* más cercano y realiza el paso 2.

DesusoNodoSimple: el *nodo simple* solicitara el cambio de coordinador.

```
DesusoNodoSimple()
    Local: IP coordinadores_disponibles[], IP coordinador_temporal, mensaje
           mensaje_integracion, mensaje mensaje_confirmacion, boolean
           bandera_coordinador, integer k
    1: coordinadores_disponibles[] ← recibir()
    2: distancia_promedio[] ← CalculaDistanciaPromedio(coordinadores_disponibles[])
1   3: Qsort(distancia_promedio[])
    4: coordinador_temporal ←distancia_promedio[0].IP
```

```
20:  end if
21:  if LocalIP not equals coordinador then
22:      Enviar(coordinador, distancia_promedio[])
23:  End if
24:  if LocalIP equals coordinador then
25:      mensajes_recibidos[] ← Recibir()
26:      Agregar(mensajes_recibidos[], distancia_promedio[])
27:  end if
```

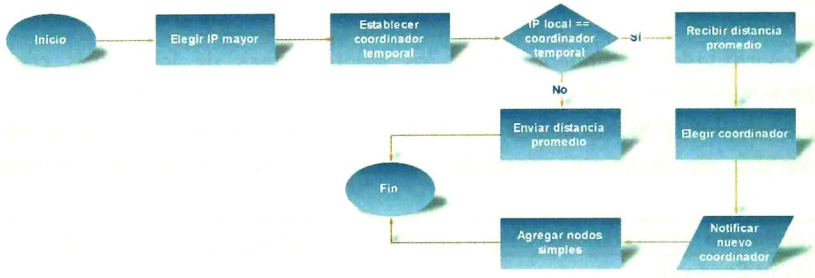


Figura 42: Elegir coordinador.

6.2 Carga de trabajo

La detección de la carga de trabajo (ver **Figura 43**) es parte del comportamiento del nodo coordinador. La carga de trabajo del *coordinador* está en función del uso del CPU. En este trabajo de investigación se definieron dos tipos de límites (inferior y superior) con valores arbitrarios, con la finalidad de realizar las pruebas necesarias al algoritmo. Los tipos de límites son:

- **Desuso:** límite inferior del 30%, se considera que la carga de trabajo actual del *coordinador* puede ser administrada por otro *coordinador*.
- **Sobrecarga de trabajo:** límite superior del 80%, para permitir que el PV realice una visualización en tiempo real es necesario reducir la carga de trabajo del nodo *coordinador*.

Si $CT(PV)$ <= limite_inferior *entonces*
    PV en $DESUSO$
Si $CT(PV)$ >= limite_superior *entonces*
    PV con $SOBRECARGA$

Los límites pueden ser administrados de tal forma que la sobrecarga de trabajo no impida realizar una visualización en TR.

6.2.1 Desuso

1. Una vez que recibe el mensaje de sobrecarga compara la dirección del *coordinador* sustituto con la dirección propia si las direcciones no coinciden ir al paso 2 en otro caso ir al paso 4.
2. El nodo envía un mensaje al *coordinador* secundario informando su distancia promedio.
3. Si el nodo recibe confirmación de integración al grupo actualiza su coordinador y envía un mensaje de cambio de *coordinador* al coordinador anterior. El nodo ya no realiza ninguna acción.
4. El nodo cambia su estatus a *coordinador*.
5. De todos los nodos que pidan integrarse al grupo elige a la mitad más cercana a él para agregarlos a su grupo.
6. El *coordinador* secundario envía una confirmación a todos los nodos elegidos.

SobreCargaNodoSimple: el *nodo simple* detecta un mensaje de sobrecarga e inicia el proceso para elegir un nuevo coordinador.

```
SobreCargaNodoSimple(mensaje mensaje_sobrecarga)
    Local: IP coordinador_temporal, mensaje mensaje_confirmacion, mensaje
           mensaje_cambioCoordinador, mensaje mensaje_nuevosNodos, integer
           distancia_nuevos[], IP nodosSiemples[], IP nodo
1   1:  coordinador_temporal ← mensaje_sobrecarga.IP
    2:  if LocalIP not equals coordinador_temporal then
2   3:      Enviar(coordinador_temporal, distancia_promedio)
    4:      mensaje_confirmacion ← recibir()
    5:      If mensaje_confirmacion not equals null then
    6:          Enviar(coordinador, mensaje_cambioCoordinador)
3   7:          coordinador ← coordinador_temporal
    8:      end if
    9:  else
4   10:     status ← COORDINADOR
    11:     mensaje_nuevosNodos[] ← recibir()
    12:     Qsort(distancia_nuevosNodos)
5   13:     for i ← 1 ... (distancia_nuevosNodos.size())/2) do
    14:         nodo ← distancia_nuevosNodos[i]
5   15:         Agregar(nodosSimples[], nodo)
    16:     end for
6   17:     Enviar(nodosSimples[], mensaje_confirmacion)
    18: end if
```

```
2  5:  k ← 0
   6:  bandera_coordinador ← false
   7:  repeat
   8:      integer i
   9:      i ← 1
  10:      Enviar(coordinador_temporal, mensaje_integracion)
3 11:      mensaje_confirmacion ← Recibir()
  12:      if mensaje_confirmacion not equals null then
  13:          coordinador ← coordinador_temporal
  14:          bandera_coordinador ← true
4 15:      else
  16:          if k < kmax then
  17:              k ← k + 1
  18:          else
  19:              coordinador_temporal ← distancia_promedio [i].IP
  20:              i ← i +1
5 21:          end if
  22:      end if
  23:  Until bandera_coordinador equals true
```

6.2.2 Sobrecarga

Cuando el nodo *coordinador* tenga una sobrecarga de trabajo disminuirá su trabajo solicitando a alguno de los nodos del AI que tome el rol de coordinador secundario. La carga de trabajo será equitativa, es decir, el número de nodos a cargo de cada *coordinador* será el mismo. El *coordinador* que detecte tiene una sobrecarga realizara el siguiente procedimiento:

1. El *coordinador* elige al nodo con la segunda menor distancia promedio.
2. El *coordinador* envía un mensaje de sobrecarga a todos los nodos de su grupo informando la dirección del coordinador secundario.
3. El *coordinador* elimina de su tabla de grupo a todos los nodos que le envíen su confirmación de cambio de *coordinador*.

SobreCargaCoordinador: La sobrecarga de trabajo podría afectar la visualización en tiempo real, por lo tanto se elige un segundo *coordinador*.

```
SobreCargaCoordinador()
    Local:  IP coordinador_temporal, mensaje mensaje_sobrecarga, IP nodosSimples[]
1   1:  coordinador_temporal ← distancia_promedio[1]
2   2:  Enviar(AI[], mensaje_sobrecarga)
3   3:  nodosSimples[] ← Recibir()
    4:  Eliminar_nodo(nodo_simples[])
```

Los *nodos simples* al recibir una notificación de sobrecarga de trabajo del *coordinador* ejecutaran la siguiente secuencia de pasos:

14. Se establece k en cero.
15. Se envía un mensaje de integración al *coordinador*.
16. Esperar una respuesta por un tiempo determinado.
17. Si se recibe una respuesta se finaliza, de lo contrario se incrementa k en uno.
18. Si k es mayor que el límite de intentos de integración ($kmax$), se realiza el paso 19, de lo contrario el paso 15.
19. Se elige el *coordinador* con la siguiente distancia promedio menor, se realiza el paso 12.

Conexion: realiza las tareas necesarias para que un nodo participe en un AV.

```
Conexion()
    Local: IP coordinador_temporal, mensaje mensaje_confirmacion, mensaje
           mensaje_cambioCoordinador, mensaje mensaje_nuevosNodos, integer
           distancia_nuevos[], IP nodosSiemples[], IP nodo
 1:  repeat
 2:      band_respuesta ← false
 3:      Enviar(nodos[], mensaje_inicio)
 4:      respuesta[] ← Recibir()
 5:      if respuesta[] equals null then
 6:          status ← COORDINADOR
 7:          break
 8:      end if
 9:      respuesta_espera ← Recibir()
10:      if respuesta_espera not equals null then
11:          band_respuesta ← true
12:      end if
13:  until band_respuesta equals false
14:  Agregar(NodosConectados[], respuesta[])
15:  ClasificaNodos(NodosConectados[])
16:  coordinadores[] ← NodosConectados.getCoordinadores()
17:  if coordinadores.size() == 0 then
18:      Agregar(AI[], NodosConectados[])
19:      distancia_promedio[] ← CalculaDistanciaPromedio(NodosConectados[])
20:      ElegirCoordinador()
21:  else
22:      Qsort(Coordinadores[])
23:      i ← 0
24:      coordinador ← coordinadores[i]
25:      band_coordinador ← false
26:      repeat
27:          nodosAI[] ← Buscar(NodosConectados[], coordinador)
28:          Agregar(AI[], nodosAI[])
29:          distancia_promedio[] ← CalcualDistanciaPromedio(AI[])
30:          k ← 0
31:          band_mensaje_integracion ← false
32:          repeat
33:              Enviar(coordinador, mensake_integracion)
```

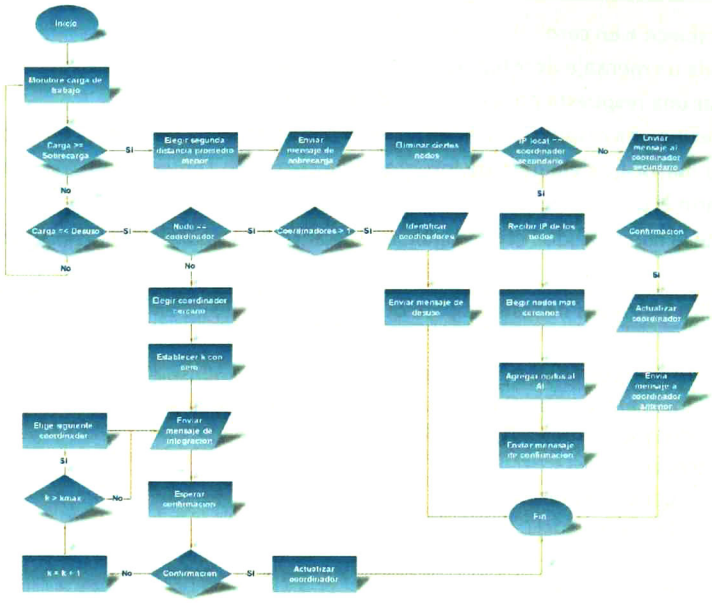


Figura 43: Carga de trabajo.

6.3 Algoritmo de conexión

El algoritmo parte de una tabla de direcciones IP pertenecientes a los nodos de un AV. Cuando un nodo desea incorporarse a un AV realiza los siguientes pasos (**Figura 44**):

1. Envía un mensaje de inicio, con el cual solicita su integración al AV.
2. Espera un tiempo determinado los mensajes de respuesta al mensaje inicial.
3. Si no existe respuesta al mensaje inicial se declara *coordinador* y finaliza el proceso.
4. Verifica si recibe un mensaje de espera por acuerdo, es decir, cuando se recibe un mensaje de este tipo indica que se está llevando a cabo una elección de coordinador. Si existe un mensaje regresa al paso 1.
5. Agrega a la tabla de AI a los nodos que contestaron el mensaje inicial.
6. Se clasifican los nodos en *coordinadores* y *nodos simples*.
7. Si no existe un coordinador se realiza el paso 8, de lo contrario el paso 11.
8. Se establece como un solo grupo a todos los nodos conectados.
9. Calcular la distancia promedio hacia el resto de los nodos a través de los mensajes recibidos.
10. Ejecuta el algoritmo para elegir un *coordinador* y finaliza.
11. Elegir al *coordinador* con la distancia promedio menor.
12. Se establece como grupo a todos los nodos que tengan como coordinador al coordinador elegido en el paso anterior.
13. Se calcula la distancia promedio a los nodos del grupo.

2. Se elige el nodo con la segunda menor distancia promedio.
3. Envía a todos los nodos a su cargo un mensaje donde notifica que nodo será el *coordinador* y finaliza.
4. Envía un mensaje de desconexión a su *coordinador*.

DesconexionVoluntaria: Cuando el nodo desee abandonar el AV realizará este procedimiento.

```
DesconexionVoluntaria()
    Local:  IP coordinador_temporal, mensaje mensaje_desconexion
1   1:  if status equals COORDINADOR then
2   2:      coordinador_temporal ← distancia_promedio[1]
3   3:      Enviar(AI[], coordinador_temporal)
    4:  else
4   5:      Enviar(coordinador, mensaje_desconexion)
    6:  end if
```

Si un nodo *coordinador* recibe un mensaje de desconexión de alguno de los nodos de su *AI*, realizará los siguientes pasos:

1. Elimina al nodo que envía el mensaje.
2. Si al eliminar el nodo el grupo queda con un solo nodo, es decir con el *coordinador*, el *coordinador* cambia su estado a *nodo simple*, realiza el algoritmo de conexión y finaliza.

DesconexionCoordinador: Es invocado cuando el coordinador recibe un mensaje de desconexión.

```
DesconexionCoordinador(mensaje mensaje_desconexion)
    Local:  ∅
1   1:  AI.erase(mensaje_desconexion.IP)
    2:  if AI.size() == 1 then
2   3:      status ← NODO_SIMPLE
    4:      Conexión()
    5:  end if
```

La desconexión de un *coordinador* provoca que un nodo simple realice la siguiente secuencia:

1. Si la dirección del nuevo *coordinador* es igual que la dirección local, realiza el paso 2, de lo contrario, el paso 4.
2. El nodo simple actualiza su estado a *coordinador*.
3. Agrega a su grupo a todos los nodos que envíen un mensaje que contenga su distancia promedio y finaliza.
4. El nodo actualiza su *coordinador*.
5. El nodo envía un mensaje de integración que contiene su distancia promedio y finaliza.

DesconexionNodoSimple: Es invocado cuando el nodo simple recibe un mensaje de nuevo *coordinador*.

```
16  34:             respuesta_integracion ← Recibir()
17  35:             if respuesta_integracion equals null then
    36:                 k ← k + 1
18  37:                 if k > kmax then
    38:                     Coordinador ← coordinadores[i]
19  39:                     i ← i + 1
    40:                     band_coordinador ← true
    41:                 else
19  42:                     band_mensaje_integracion ← true
    43:                 end if
    44:             end if
    45:         until band_mensaje equals true
    46:     until band_coordinador equals true
    47: end if
```

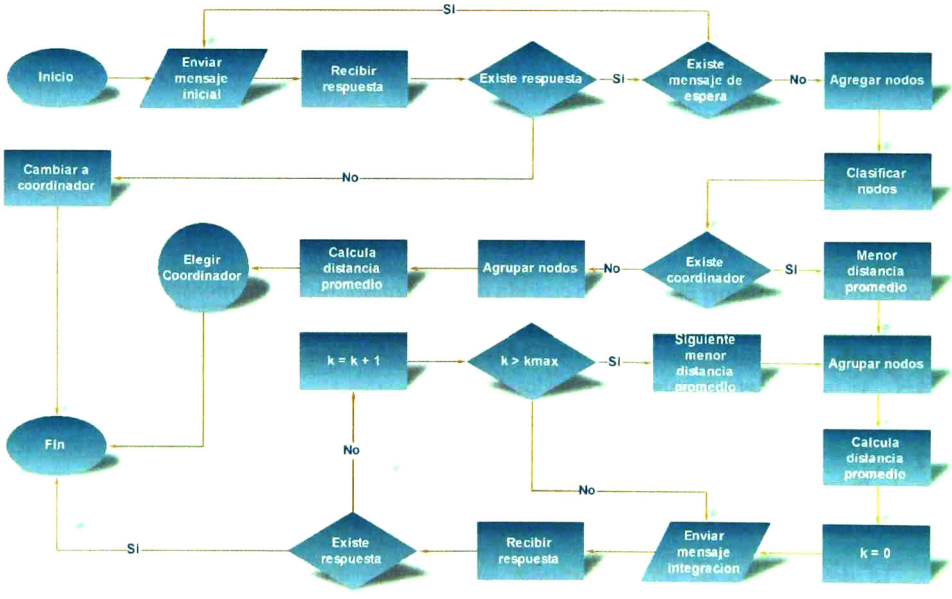


Figura 44: Integrar nuevos nodos.

6.4 Algoritmo de desconexión voluntaria

Por medio de los siguientes algoritmos se notifica cuando un nodo (*coordinador* o *nodo simple*) desea abandonar un AV (**Figura 45**). El primer algoritmo a ejecutar es DesconexionVoluntaria, el cual identifica el tipo de nodo y envía la notificación correspondiente, los pasos son:

1. Verifica si el nodo es *coordinador*, si lo es, realiza el paso 2. Si no es *coordinador* realiza el paso 4.

5. Si se trata de un nodo *coordinador* realizará los pasos del algoritmo SobreCargaCoordinador, si no es así activara a SobreCargaNodoSimple pasándole como parámetro el mensaje recibido.
6. Si el mensaje es de desconexión y el nodo es un *coordinador* realizará DesconexionCoordinador.
7. Si el tipo de mensaje es de nuevo *coordinador* y el nodo es un *nodo simple* activara el algoritmo llamado DesconexionNodoSimple, que requiere del mensaje recibido.

MonitorMensajes: verifica si existe un nuevo mensaje y lo clasifica en: desuso, sobrecarga, desconexión y nuevo coordinador.

```
MonitorMensajes()
    Local: mensaje mensaje_integracion_recibir[], mensaje mensaje_integracion
1   1:  mensaje ← recibir()
2   2:  if mensaje.type() equals DESUSO then
    3:      if status equals COORDINADOR then
    4:          DesusoCoordinador()
3   5:      else
    6:          DesusoNodoSimple()
    7:      end if
    8:  end if
    9:  if mensaje.type() equals SOBRECARGA then
4   10:     if status equals COORDINADOR then
    11:         SobreCargaCoordinador()
    12:     else
5   13:         SobreCargaNodoSimple(mensaje)
    14:     end if
    15: end if
    16: if mensaje.type() equals DESCONEXION and status equal COORDINADOR then
6   17:     DesconexionCoordinador(mensaje)
    18: end if
    19: if mensaje.type() equals NUEVO_COORDINADOR and status equal
7       NODO_SIMPLE then
    20:     DesconexionNodoSimple(mensaje)
    21: end if
```

```
DesconexionNodoSimple(mensaje mensaje_nuevo_coordinador)
    Local:  mensaje mensaje_integracion_recibir[], mensaje mensaje_integracion
1   1:  if mensaje_nuevo_coordinador.IP equals LocalIP then
2   2:      status ← COORDINADOR
3   3:      mensaje_integracion_recibir [] ← Recibir()
    4:      Agregar(AI[], mensaje_integracion_recibir [])
    5:  else
4   6:      coordinador ← mensaje_nuevo_coordinador.IP
5   7:      Enviar(coordinador, mensaje_integracion)
    8:  end if
```

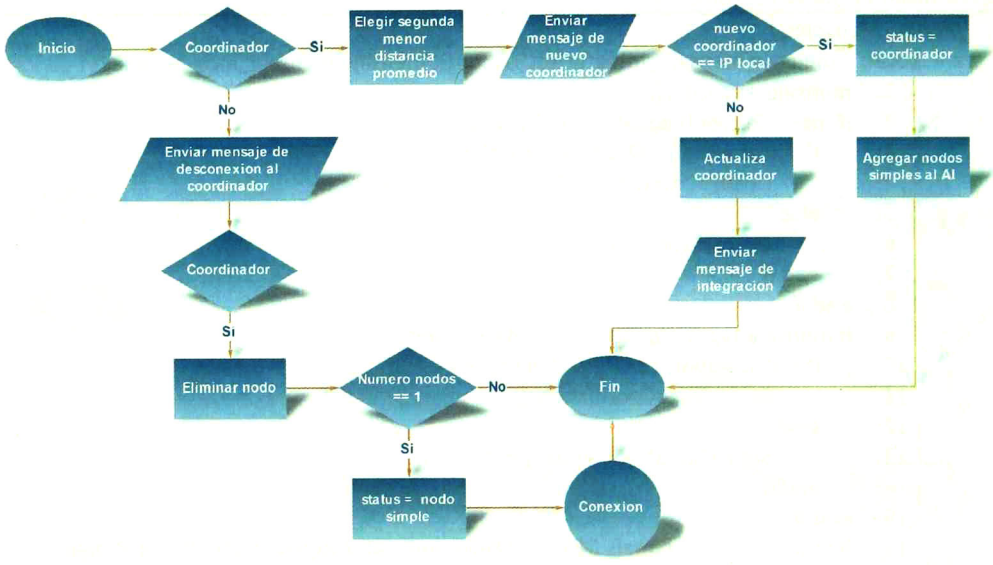


Figura 45: Desconexión de nodos.

6.5 Algoritmo de Monitoreo de mensajes

Es necesario un proceso que continuamente esté en espera de nuevos mensajes, dichos mensajes activaran algún algoritmo de los antes mencionados. La tarea principal de este proceso es la ejecución del algoritmo MonitorMensajes, que contiene las siguientes tareas (**Figura 46**):

1. Detectar si existe un nuevo mensaje.
2. Si el mensaje notifica que un nodo está en desuso identificará que tipo de nodo es.
3. Si es un nodo *coordinador* ejecutará el algoritmo DesusoCoordinador, de lo contrario ejecuta DesusoNodoSimple.
4. Si el mensaje es de sobrecarga deberá identificar el tipo de nodo local.

Capítulo 7 Formalización de Puertos Visuales utilizando Algebra de Procesos

Objetivo:

- Formalizar la arquitectura P2P propuesta para los Puertos Visuales de un Ambiente Virtual.
- Definir los procesos y su interacción para llevar a cabo una visualización en tiempo real, administrando la consistencia del ambiente.

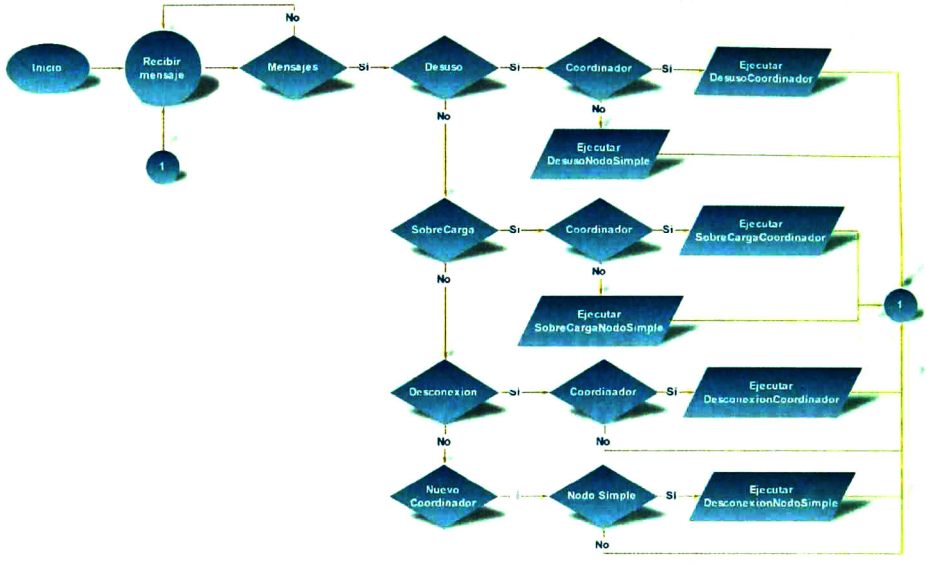


Figura 46: Monitoreo de mensajes.

6.6 Conclusiones

Los algoritmos descritos anteriormente permiten que la arquitectura de red del AV sea adaptativa en base a la carga de trabajo y la distancia promedio de los nodos que ejecutan un PV. La pérdida de mensajes y el retardo son disminuidos debido a que los nodos son agrupados de acuerdo a su cercanía, permitiendo continuar con una visualización en tiempo real y administrar la consistencia entre los PVs, características que permiten brindar un grado aceptable de QoE. La cantidad de mensajes en la red es reducida al agrupar los nodos y al disminuir el número de conexiones necesarias para mantener comunicados los PVs.

Cuando los límites están en función del porcentaje del uso del CPU es posible considerar el trabajo generado no sólo por el PV sino por los demás servicios ejecutados por el nodo. La carga de trabajo generada por el conjunto de servicios podría afectar la QoE que se brinda, y no sólo el trabajo del PV.

Al mantener los límites con porcentajes la cantidad de tareas realizadas por cada nodo dependerá de sus características tanto de software como de hardware.

La **Figura 47** muestra los procesos de nuestra arquitectura P_2P de un PV, así como sus canales de comunicación. Los procesos principales son: DA (capa de Distribución de Actualizaciones), VC (capa de Verificación de Consistencia) y PL (capa de Procesamiento Local). Dichos procesos corresponden a las capas de la arquitectura P_2P propuesta y descritas previamente en la sección 3.4. Las entradas y salidas de cada proceso habilitan su ejecución de forma paralela y/o concurrente de estos procesos así como la activación de ciertas tareas internas de cada proceso.

Un mensaje de solicitud puede ser local o externo. Una solicitud externa se considera como un mensaje de actualización (MA). Cuando una solicitud es recibida se activaran los procesos DA y PL. A continuación se describen a grandes rasgos las tareas de cada proceso:

- **Proceso DA:** Hay tres casos posibles a analizar que son: 1. Si el proceso DA recibe una solicitud local realizara el envío a los PVs pertinentes; 2. Cuando la solicitud es externa se verifica la estampilla de tiempo para evaluar si es necesario procesarla localmente; 3. Cuando el proceso DA recibe un mensaje por el canal AI este actualiza los PVs pertenecientes a los grupos de AIs. Ahora bien, si el proceso DA recibe un mensaje de consistencia MC y el nodo local es un *nodo coordinador,* este envía a su vez, el mensaje a los coordinadores de cada AI. Por el contrario, si el nodo local es un *nodo simple* entonces envía el mensaje al coordinador de su AI.

- **Proceso VC:** Este proceso recibe los mensajes de actualización del proceso DA y los envía al proceso PL. Cuando recibe un mensaje de consistencia, verifica la consistencia tanto local (en la AI al cual pertenece) como global (entre las diferentes *Áreas de Interés* - AIs). Si recibe una notificación de cambios en las posiciones de los elementos del Ambiente Virtual, verifica los grupos de nodos para reorganizarlos si es necesario. Además, este proceso genera los mensajes de consistencia que son enviados al proceso DA.

  Proceso PL: Este proceso se encarga de la administración local de las solicitudes independientemente de que sean externas o locales. La administración de las solicitudes es necesaria dado que estas generan eventos cuando la consistencia local se ve afectada. De las solicitudes recibidas se identifica el avatar y la acción que se deben administrar. Las solicitudes son agrupadas tomando en cuenta el rango de tiempo en el cual los procesos deben ser ejecutados, este rango según nuestros requerimientos debe ser en Tiempo Real.

En las siguientes secciones se describen y formalizan tanto la información (sección 7.1) como los procesos (sección 7.2) requeridos en un PV.

El álgebra de procesos se centra en la especificación y manipulación de proceso activados por un conjunto de operadores [**96**]. Un proceso constituye la unidad básica para realizar la especificación formal de un sistema. Por lo general, el álgebra de procesos contiene nombres de acciones (expresan eventos atómicos), operadores alternativos, secuenciales y paralelos para modelar sistemas basados en procesos. El álgebra de procesos también permite la especificación de la recursividad en los procesos. El álgebra de procesos ha sido utilizado para formalizar protocolos por ejemplo: el protocolo de ventana deslizante, protocolo de retransmisión limitada, etc. Nosotros utilizaremos el álgebra de procesos para formalizar la arquitectura propuesta con el objetivo de modelar las funcionalidades de la arquitectura P2P propuesta. Como ya fue descrito anteriormente, la arquitectura P_2P fue diseñada bajo el esquema de capas, lo cual permite la ejecución de diversas tareas de forma paralela y/o concurrente (ver **Figura 47**).

La especificación formal de la arquitectura se realizó con el objetivo de mostrar una mejor descripción modular de los procesos, así como un mejor entendimiento y una vista abstracta de la arquitectura P_2P. Atreves del envío y recepción de mensajes entre los procesos es posible realizar diversas tareas en forma paralela, por ejemplo, cuando el PV recibe una solicitud son activados los procesos correspondientes a las capas de Procesamiento Local (PL) y Distribución de Actualizaciones (DA).

Una vez definidos los procesos de un PV, es posible mostrar que un Ambiente Virtual está formado por un conjunto de PVs ejecutados de forma paralela, es decir, se realiza la distribución de los PVs.

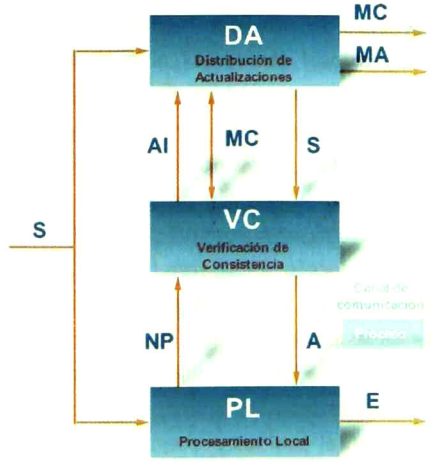


Figura 47: Procesos principales en un *Puerto Visual*.

	$NS :\rightarrow NodoSimple$
	$Coordinador, NodoSimple :\rightarrow TipoNodo$
	$Local, Externa, Actualizacion :\rightarrow TipoSolicitud$
	$Solicitud, AIs, Mensajes :\rightarrow TipoParametroDA$
	$Solicitud, Posiciones :\rightarrow TipoParametroVC$
	$Solicitud :\rightarrow TipoParametroPL$
	$Solicitud, AIs, MC, Posiciones :\rightarrow MensajesR$
	$Evento, Mensajes :\rightarrow MensajesS$

La palabra clave ***rew*** permite re-escribir términos que son interpretados de izquierda a derecha. El primer término (lado izquierdo del igual) puede contener parámetros de los cuales depende el valor producido a la derecha del igual. Por ejemplo: $eq(V, F) = F$, el primer término es $eq(V, F)$, que tiene dos parámetros V y F, los cuales son evaluadas para producir el término de la derecha F. La siguiente tabla describe las fórmulas que pertenecen a este grupo de funciones ***rew***.

Palabra clave	Función
rew	$eq\ (V,V) = V$ $eq(F,F) = V$ $eq(V,F) = F$ $eq(F,V) = F$ $eq(d,b) = V \quad if\ d == b$ $eq(d,b) = F \quad if\ d \neq b$

La palabra clave ***map*** se utiliza para declarar funciones que no son los constructores.

Palabra clave	Función	Descripción
map	$\wedge : Bool \times Bool \rightarrow Bool$ $<: Nat \times Nat \rightarrow Bool$ $\leq: Nat \times Nat \rightarrow Bool$ $>: Nat \times Nat \rightarrow Bool$ $\geq: Nat \times Nat \rightarrow Bool$	Operadores lógicos
	$PU : Perfiles \rightarrow Nat$	Tiempo de los perfiles de usuarios.
	$PerfilLocal :\rightarrow Perfiles$	Perfil actual del PV.
	$markov : Solicitud \rightarrow Bool$	Administración de los mensajes a enviar.
	$inc : Nat \times Nat \rightarrow Nat$	Realiza la suma de los parámetros recibidos.
	$tiempo : Solicitud \rightarrow Nat$	Regresa el tiempo contenido en la solicitud recibida.
	$estimaRetardo : Solicitud \times Nat \rightarrow Bool$	Estimación del retardo permitido.

7.1 Datos requeridos por los procesos

En esta sección se describen los datos que son necesarios para los procesos de un PV. Se hace una descripción breve de las palabras claves (***sort***, ***func***, ***map***, etc.) requeridas en la formalización y a continuación se formalizan el manejo de la información de los procesos utilizados de la arquitectura P_2P propuesta.

Los tipos de datos son declarados con la palabra ***sort***. Cada tipo declarado representa un conjunto no vacío de elementos de datos.

Palabra clave	Tipo de datos	Descripción
sort	$Bool$	Valores booleanos
	$Solicitud$	Solicitudes admitidas por los elementos de un PV.
	AIs	Áreas de Interés de un Ambiente Virtual.
	$Mensaje$	Mensajes de comunicación entre PVs.
	$Posiciones$	X, Y, Z de un elemento del Ambiente Virtual.
	$Perfiles$	Perfiles que puede tener un PV.
	PA	Perfil Alto.
	PM	Perfil Medio.
	PB	Perfil Bajo.
	$Coordinador$	Tipo de nodo perteneciente al Ambiente Virtual.
	MA	Mensaje de Actualización.
	MC	Mensaje de Consistencia.
	Nat	Números Naturales.
	$NodoSimple$	Tipo de nodo perteneciente al Ambiente Virtual.
	φ	Conjunto de acciones a realizar.

Con la palabra reservada ***func*** se declararan constructores, que definen la estructura de los tipos de datos.

Palabra clave	Constructores
func	$V, F :\rightarrow Bool$
	$sol_1, sol_2, \cdots sol_n :\rightarrow Solicitud$
	$AI_1, AI_2, \cdots AI_n :\rightarrow AIs$
	$Nat \times Nat \times Nat :\rightarrow Posiciones$
	$e_1, e_2, \cdots e_n :\rightarrow Evento$
	$PA, PM, PB :\rightarrow Perfiles$
	$MC, MA :\rightarrow Mensajes$
	$C :\rightarrow Coordinador$

		notificación de cambio de AI se actualizan las direcciones IP involucradas.
	$VerCon : Solicitud$	Es activada cuando se recibe una solicitud externa que no es una actualización, es decir, un mensaje de consistencia externo. Su función es analizar si con el mensaje recibido genera alguna inconsistencia tanto a nivel de AI ($VerCon_{AI}$) como a nivel del Ambiente Virtual ($VerCon_{\alpha}$).
	$IAvAc : Solicitud$	Identifica el avatar y la acción contenidas en la solicitud, para realizar la asignación correspondiente.
	CE	Crea el evento correspondiente al resultado de los cambios en el Ambiente Virtual.
	$asig : Nat \times Nat$	Asigna el valor del segundo parámetro al primero.
	$agregar : \varphi \times Solicitud$	Agrega al proceso (primer parámetro) la solicitud recibida (segundo parámetro).
	$cambioC : AIs$	Verifica si es necesario un cambio de coordinador en el AI.

En la siguiente sección se declararan los procesos necesarios para un PV, para su mejor entendimiento a continuación se describen los operadores utilizados. Un proceso es declarado con la palabra ***proc*** seguida del nombre del proceso y su comportamiento. En el comportamiento de un proceso es posible encontrar los siguientes operadores:

- **Condicional ($Verdadero \lhd ValorBooleano \rhd Falso$)**: Cuando el valor booleano es verdadero se ejecutara el/los proceso(s) de la izquierda, de lo contrario el/los proceso(s) del lado derecho.

- **Deadlock o inacción (δ)**: no ejecuta ninguna multi-acción, muestra un comportamiento de demora.

- **Composición Alternativa ($p + q$)**: expresión no determinista opta por ejecutar p o q.

- **Composición Secuencial ($p \cdot q$)**: la primera expresión se ejecuta (p), una vez terminado p continúa con la ejecución de q.

Sumatoria ($\sum_{d:D} p$): donde d es una variable de tipo D y p es una expresión de proceso en el que esta variable puede ocurrir. En este caso, decimos que d está ligada a p. El comportamiento que corresponde es una elección no determinista entre los procesos de $p\,(d)$ para todas las expresiones de d del tipo de datos D. Si $\{d_0, d_1, \cdots, d_n\}$ son expresiones de datos de tipo D, entonces $\sum_{d:D} p$ puede expresarse como $p\,(d_0) + p\,(d_1) + \cdots + p\,(d_n)$.

	$TS : Solicitud \rightarrow TipoSolicitud$	Recibe como parámetro una solicitud con la cual se identifica si es: Local, Externa o una Actualización, regresándolo como resultado.
	$cambioAI : Posiciones \rightarrow Bool$	Recibe como parámetro una nueva posición para validar si existe algún cambio en las AIs.
	$OriS : Solicitud \rightarrow TipoNodo$	Cuando se recibe un mensaje de consistencia externo se identifica si el origen es de un nodo simple o de un coordinador, lo cual es el resultado.
	$ejecuta : \varphi \rightarrow Bool$	Ejecuta el proceso que recibe como parámetro. Regresa verdadero si se generó algún cambio en el Ambiente Virtual.
	$TL : Solicitud \rightarrow Nat$	Regresa el tiempo del PV local.
	$GM : Posiciones \rightarrow Mensajes$	Genera un mensaje de consistencia.
	$waitDA : \rightarrow TipoParametroDA$	Regresa el tipo de datos recibido.
	$waitVC : \rightarrow TipoParametroVC$	Regresa el tipo de datos recibido.
	$waitPL : \rightarrow TipoParametroPL$	Regresa el tipo de datos recibido.

La comunicación entre dos procesos es declarada con la palabra reservada ***comm***. La comunicación entre dos procesos ocurre en este caso en forma paralela.

Palabra clave	Comunicación
comm	$AreaVision \mid LimitesAmbiente = GAI$

En el álgebra de procesos una acción representa eventos atómicos en el mundo real y es declarada por la palabra ***act***. Una acción está compuesta por un nombre y ninguno o varios parámetros.

Palabra clave	Acción	Descripción
act	GAI	Cuando se genera un cambio en alguna AI es necesario generar los grupos de AIs, función que realiza GAI.
	$r_c : MensajesR$ $s_c : TipoNodo \times MensajesS$	Las entradas y salidas son representadas por las funciones r y s respectivamente. El subíndice indica el canal de comunicación correspondiente. Los parámetros de s son: a) el destino del mensaje, b) el mensaje a enviar.
	$AcAI : AIs$	Cuando la capa de Distribución de Actualizaciones (representada por el proceso DA) recibe una

$$\textbf{proc}\ iniciarAcciones = \sum_{mi,tiempoPrimeraAccion:Nat} asig\ (mi, tiempoPrimeraAccion)$$

El proceso PU administra el retardo permitido en cada perfil. El proceso $PerfilLocal$ regresara el perfil correspondiente al PV local, tomando en cuenta el valor almacenado en una marca de inicio (mi). El valor regresado por el proceso PU es utilizado por el proceso $markov$ que estima el retardo que podría sufrir un mensaje al ser enviado.

$$\textbf{proc}\ PerfilLocal = \sum_{PB,PM,PA:Perfiles} \sum_{mi:Nat} PB \lhd mi \leq 1 \rhd \left(PM \lhd mi > 1 \wedge mi < 10 \rhd (PA \lhd mi \geq 10 \rhd \delta)\right)$$

Al activar el proceso $markov$ se estima el retardo que podría tener la solitud (sol), la estimación del retardo $\left(estimaRetardo(sol, PU\ (PerfilLocal))\right)$ regresa un verdadero si se estima que el tiempo del retardo está dentro del retardo permitido en el perfil local, en este caso se envía el mensaje de actualización (solicitud) a los nodos pertenecientes al AI, de lo contrario se comprime el mensaje para realizar su envió al AI. El perfil local es regresado por PL, y su tiempo correspondiente por $PU\ (p)$.

$$\textbf{proc}\ markov\ (sol : Solicitud) = \sum_{sol:Solicitud} \sum_{MA:Mensajes} \sum_{AI:AIs} s_{MA}\ (AI, MA) \lhd estimaRetardo(sol, PU\ (PerfilLocal)) \rhd comprime\ (sol) \cdot s_{MA}\ (AI, MA)$$

Cuando se recibe una solicitud (mensaje de actualización) de otro PV, el proceso verificar tiempo $(VT(sol))$ compara la marca de tiempo de la solicitud $\left(tiempo\ (sol)\right)$ con el tiempo local más el retardo permitido por el perfil local $\left(inc(TiempoLocal, PU\ (PerfilLocal))\right)$, si es menor o igual, la solicitud es procesada, de lo contrario es descartada. Donde [ ] significa la ausencia de parámetros.

$$\textbf{proc}\ VT\ (sol : Solicitud) = \sum_{sol:\ Solicitud} \sum_{TiempoLocal:Nat} VC\ (sol, [\ ]) \lhd tiempo\ (sol) \leq inc(TiempoLocal, PU\ (PerfilLocal)) \rhd descarta\ (sol)$$

Cuando en un Ambiente Virtual participan más de un PV, es necesario identificar el tipo de solicitud recibida (IS). Si una solicitud proviene de otro PV (solicitud externa) se analiza para ser desplegada al usuario, de lo contrario, cuando la solicitud es local es necesario además de desplegarla, enviarla al resto de los PVs del Ambiente Virtual, de esta forma se dan a conocer las

A continuación se define un proceso a modo de ejemplo. La tarea principal del proceso es la compra de un tono de verde para pintar.

$$\textbf{\textit{sort}}\ Colores$$

$$\textbf{\textit{func}}\ verdeManzana, verdeMilitar, verdeLimon, verdeSeco : \rightarrow Colores$$

$$\textbf{\textit{act}}\ pintar$$

$$\textbf{\textit{rew}}\quad eq(d,b) = V \quad if\ d == b$$

$$eq(d,b) = F \quad if\ d \neq b$$

$$\textbf{\textit{proc}}\ ElegirColor\ (color : Colores)$$
$$= \sum_{verdeManzana,verdeLimon:Colores} (comprar\ (verdeManzana)$$
$$+ compar\ (verdeLimon)) \cdot pintar \lhd eq\ (color, verde) \rhd \delta$$

El proceso $ElegirColor$ recibe como parámetro un $color$ valido para el tipo de datos $Colores$. La sumatoria indica las variables y su tipo que pueden ser utilizadas por el proceso. Si se recibe el color verde $(eq\ (color, verde))$ se puede elegir entre comprar un color verde manzana $(comprar\ (verdeManzana))$ o comprar un verde limón $(compar\ (verdeLimon))$, después de realizar la compra, se realiza el proceso de $pintar$. Cuando el color recibido no es verde se pasa a un estado de inacción (δ). Para una mayor descripción del álgebra de procesos utilizada se recomienda las siguientes referencias [96], [97], [98], [99].

7.2 Descripción y formalización de los procesos de un PV

A continuación se describen y formalizan las capas de la arquitectura P_2P propuesta. Un PV en un tiempo t está en uno de los tres perfiles del usuario (PU, ver sección 3.4.3), el tiempo correspondiente a cada perfil se formaliza de la siguiente manera:

$$\textbf{\textit{proc}}\ PU\ (p : Perfiles)$$
$$= \sum_{p,PA,PM,PB:Perfiles} 232 \lhd eq\ (p, PA)$$
$$\rhd\ (\ 176 \lhd eq\ (p, PM) \rhd (339 \lhd eq(p, PB) \rhd \emptyset))$$

Los perfiles del usuario dependen del tiempo de conexión con el Ambiente Virtual, para lo cual se guarda el tiempo de envió de la primera acción del PV. El proceso $iniciarAcciones$ almacena en la variable mi el tiempo de la primera solicitud enviada.

$$\textbf{proc}\ VAI\ (pos : Posiciones)$$
$$= \sum_{AI:AIs} \sum_{pos:Posiciones} GAI \cdot DA([\ ], AI, [\ ]) \cdot cambioC\ (AI)$$
$$\lhd\ cambioAI\ (pos) \rhd\ \delta$$

La capa de Verificación de Consistencia es representada por el proceso VC, descrito a continuación:

- Al recibir una solicitud se verificará sí es una actualización (solicitud externa) $(eq\ (TS\ (sol), Actualizacion))$, si lo es, se enviará a procesar $(s_A\ (sol))$, de lo contrario, se trata de un mensaje de consistencia, por lo cual se verifica si proviene de un nodo simple $(eq\ (\ OriS(sol), NodoSimple\ ))$. para verificar la consistencia en el AI $(VerCon_{AI}\ (sol))$, si el nodo origen es un coordinador se valida la consistencia en el Ambiente Virtual $(VerCon_{\alpha}\ (sol))$.

- Cuando se recibe una nueva posición (pos) por el canal NP $(r_{NP}\ (pos))$ se verifican las áreas de interés $(VAI\ (pos))$, para generar un mensaje de consistencia $(GM\ (pos))$ y enviarlo a la capa de Distribución de Actualizaciones $(DA\ (\ [\ ], [\ ], MC))$.

Por último, se queda en espera $(waitVC)$ de algún parámetro valido para realizar la recursión.

$$\textbf{proc}\ VC\ (sol : Solicitud, pos : Posiciones)$$
$$= \Big(\sum_{sol:Solicitud} \sum_{Actualizacion:TipoSolicitud} \sum_{NS:TipoNodo} r_s\ (sol)$$
$$\cdot \Big(PL\ (sol, [\ ]) \lhd eq\ (TS\ (sol), Actualizacion)$$
$$\rhd \big(VerCon_{AI}\ (sol) \lhd eq\ (\ OriS(sol), NS\ ) \rhd VerCon_{\alpha}\ (sol) \big) \Big)$$
$$+ \sum_{pos:Posiciones} \sum_{MC:Mensajes} r_{NP}\ (pos) \cdot VAI\ (pos) \cdot GM\ (pos)$$
$$\cdot DA\ (\ [\ ], [\ ], MC) \Big)$$
$$\cdot \big(VC\ (sol, [\ ]) \lhd eq(waitVC, Solicitud) \rhd VC\ ([\ ], pos) \big)$$

La cantidad de acciones que contenga un proceso está en función de la ranura de tiempo dinámica (RTD). La marca de tiempo mt es un contador que se reinicia cada vez que se ejecuta un proceso φ, mientras que la ranura de tiempo rt es un valor variable que tiene como valor inicial 60 milisegundos (valor dado en base a la vista humana). A lo largo de la evolución del Ambiente

acciones de cada usuario. El proceso $IS(sol)$ identifica si la solicitud es local, si lo es, se realizara el proceso llamado $markov\ (sol)$, de lo contrario, si es externa verifica el tiempo $(VT(sol))$ de la solicitud.

$$\begin{aligned}&\textbf{proc}\ IS\ (sol : Solicitud)\\&\quad = \sum_{sol:Solicitud} \sum_{Local,Externa:TipoSolicitud} markov\ (sol)\\&\quad \lhd eq(\ TS\ (sol), Local) \rhd (VT\ (sol) \lhd eq(TS\ (sol), Externa) \rhd \delta)\end{aligned}$$

La capa de Distribución de Actualizaciones está representada por el proceso DA, el cual se describe de la siguiente forma:

- Cuando se recibe una solicitud $(r_s(sol))$ se identifica su tipo por medio del proceso $IS(sol)$.
- Cuando se genera un cambio en algún AI se recibe una notificación de tipo $AI\ (r_{AI}(AI))$, activando el proceso $AcAI(AI)$ que actualizara las diferentes AIs.
- Al recibir un mensaje de consistencia del PV local identifica que tipo de nodo es (nodo simple o coordinador), si es un nodo simple enviara el mensaje a su coordinador, de lo contrario a los coordinadores de área.
- Mediante el proceso $waitDA$ se espera la llegada de algún parámetro, realiza una llamada recursiva al proceso DA.

$$\begin{aligned}&\textbf{proc}\ DA\ (sol : Solicitud, AI :\ AIs, MC : Mensajes)\\&\quad = \Big(\sum_{sol:Solicitud} \sum_{AI:\ AIs} r_s\ (sol)\ \cdot IS\ (sol) + r_{AI}\ (AI)\ \cdot AcAI\ (AI)\\&\quad + \sum_{C,NS:TipoNodo} \sum_{AI:AIs} \sum_{MC:Mensaje} r_{MC}\ (MC)\\&\qquad \big((\ s_{MC}\ (C, MC) \lhd eq\ (nodoLocal\ , NS) \rhd s_{MC}\ (AI, MC)\big)\Big)\\&\quad \cdot \big(DA\ (sol, [\ \ ], [\ \ ]) \lhd eq(waitDA, Solicitud)\\&\quad \rhd \big(DA\ ([\ \ ], AI, [\ \ ]) \lhd eq(waitDA, AIs) \rhd DA\ ([\ \ ], [\ \ ], MC)\big)\big)\end{aligned}$$

Es necesario verificar las $AIs\ (VAI)$ cuando en el ambiente es detectado un cambio en las posiciones de los elementos $(cambioAI\ (pos))$. Esto es necesario para después generar los diferentes grupos de $AI\ (GAI)$. El proceso $cambioC\ (AI)$ analiza si el coordinador actual es el adecuado para administrar la consistencia en el AI.

ejecución de los procesos de forma paralela. Cabe mencionar que los procesos *Reset e iniciarAcciones* solo son ejecutados una vez.

$$\boldsymbol{init}\ \partial_H(DA\ (sol,[\ \ ],[\ \ ])\ \|\ PL\ (sol)\ \|\ Reset\ \|\ iniciarAcciones)$$

Cuando en la evolución de un Ambiente Virtual intervienen más de un PV, es posible definir un PV como un proceso, de la siguiente manera:

$$\boldsymbol{proc}\ PV = \sum_{sol:Solicitud} DA\ (sol,[\ \ ],[\ \ ])\ \|\ PL\ (sol)\ \|\ Reset\ \|\ iniciarAcciones$$

De esta forma la participación de diversos PV se puede generar de forma paralela, dando como resultado la distribución de los PVs de un Ambiente Virtual:

$$\boldsymbol{proc}\ \alpha = \sum_{n:\ Nat} PV_1 \| PV_2 \| \cdots \| PV_{n-1} \| PV_n$$

Por lo tanto un Ambiente Virtual estará formado por un grupo de PVs, su inicialización seria la siguiente:

$$\begin{array}{ll} \boldsymbol{var} & n : Nat \\ \boldsymbol{init} & \partial_H(PV_1 \| PV_2 \| \cdots \| PV_{n-1} \| PV_n) \\ & \quad \text{ó} \\ & \partial_H(\alpha) \end{array}$$

Cada PV inicia su participación al recibir una solicitud (sol), la cual es interpretada y procesada para su Visualización en Tiempo Real, generando cambios que son notificados al usuario de forma gráfica en Tiempo Real.

7.3 Conclusiones

El modelado de la arquitectura P_2P propuesta utilizando algebras de proceso facilita el entendimiento de la secuencia de sus tareas, así como su organización para ser realizadas de forma paralela. Mostrando más claramente el funcionamiento de la arquitectura P_2P propuesta.

Virtual el tamaño de rt puede variar por factores como la sobrecarga de trabajo del PV. En el proceso RTD se ejecutará si mt es menor que la ranura de tiempo rt más el tiempo del PV local $(mt < inc\ (tiempoLocal, rt))$, se regresa un falso indicando que se deben agrupar más acciones en el proceso φ actual, de lo contrario regresará un verdadero y se reiniciará mt para posteriormente ejecutar el proceso $Reset$.

$$\boldsymbol{proc}\ RTD = \sum_{mt,tiempoLocal,rt:Nat} \sum_{F,V:Bool} F \lhd mt < inc\ (tiempoLocal, rt) \rhd V \cdot Reset$$

$$\boldsymbol{proc}\ Reset = \sum_{mt,tiempoLocal:Nat} asig\ (mt, tiempoLocal)$$

La creación de procesos (CP) se lleva acabo dependiendo de la ranura de tiempo dinámica (RTD), que indica si se ejecuta el proceso $(ejecuta\ (\varphi))$ o se agregan la acción al proceso $(agrega\ (\varphi, sol))$. El proceso $ejecuta$ valida si existe un cambio en las posiciones de los avatares modificados regresando verdadero para enviar la(s) nueva(s) posición(es) (NP) al proceso VC $(VC\ ([\ ], NP))$.

$$\boldsymbol{proc}\ CP\ (sol : Solicitud) = \sum_{sol:Solicitud} \sum_{NP:Posiciones} (VC\ ([\ ], NP) \lhd ejecuta\ (\varphi) \rhd \delta) \lhd RTD \rhd agrega\ (\varphi, sol)$$

El proceso PL representa a la capa de Procesamiento Local, que recibe solicitudes por el canal s o actualizaciones por el canal A, de las cuales se identifica el avatar y la acción que realizará. Las acciones son agrupadas en procesos que se ejecutaran $(CP\ (sol))$, provocando tal vez un evento que será enviado por el canal e. Después de procesar las solicitudes recibidas, se queda en espera $(waitPL)$ de alguna solicitud.

$$\boldsymbol{proc}\ PL\ (sol, A : Solicitud) = \left(\sum_{sol,A\ :\ Solicitud} \sum_{e:\ Evento} (r_s(sol) + r_A\ (A)) \cdot IAvAc\ (sol) \cdot CP(sol) \cdot CE\ s_e\ (e) \right) \cdot (PL\ (sol) \lhd eq(waitPL, Solicitud) \rhd \delta)$$

La inicialización está a cargo de los procesos $DA, PL, Reset$ e $iniciarAcciones$, los cuales son activados cuando se recibe una solicitud y son ejecutados en paralelo. El operador || denota la

Capítulo 8 Conclusiones

Objetivo:

- Mencionar las aportaciones logradas en el campo de la visualización distribuida de Ambientes 3D.
- El trabajo futuro que sugerimos se debe continuar para afinar los resultados
- Listar las publicaciones realizadas en esta investigación.

Un PV es definido como un proceso que recibe solicitudes, activando dos de sus procesos (DA, PL) en forma paralela. Cada proceso realiza una secuencia de tareas, las cuales permiten realizar una visualización en tiempo real. Si es pertinente las solicitudes son enviadas a los PV necesarios, los envíos son administrados para reducir el retardo y no afectar la visualización en tiempo real en el resto de los PVs participantes en la evolución del Ambiente Virtual.

De esta forma un Ambiente Virtual es definido formalmente como un proceso que contiene un conjunto de PVs ejecutados de forma paralela, formando un sistema distribuido a través de la distribución de los PVs.

- **Visualización en tiempo real**: Uno de los métodos propuestos utiliza una RTD para administrar y agrupar las acciones que se pueden realizar cada 60 milisegundos, es decir en TR. Esto hace posible una mejor visualización en TR de los cambios ocurridos en los AVs. Además, las marcas de tiempo en los mensajes permiten la ejecución de acciones en el mismo orden en que fueron solicitadas, con lo cual aseguramos la consistencia en la visualización del AVs entre los PVs. *Objetivos considerados*: IV.

- **Distribución de PVs**: Actualmente, requerimientos de visualización de AVs 3D deben permiten la participación simultánea de n usuarios localizados en diferentes lugares geográficos (usuarios remotos). La mayoría de los trabajos proponen una arquitectura cliente-servidor para garantizar la consistencia de visualización, sin embargo esto tiene consecuencias de restricciones en la visualización en TR. En este trabajo, se presenta una arquitectura $P2P$ para la visualización en TR de AVs en 3D. Con esta propuesta, la saturación en la red es inherentemente disminuida, debido al régimen de comunicación $P2P$ que establecen los nodos que requieren comunicarse. Además de que la arquitectura $P2P$ proporcionar una mejor escalabilidad de los PVs. *Objetivos considerados*: IV, V, VI, VII.

- **Organización de los nodos de un PV**: Los nodos involucrados son agrupados en diferentes Áreas de Interés (AIs), minimizando así la cantidad de mensajes a enviar por la red. La arquitectura de red es re-organizada debido a la sobrecarga de trabajo y a la distancia promedio de los nodos. Por lo cual, se considera que la arquitectura de red es adaptativa, cualidad que permite brindar a los usuario una QoE aceptable, resultado de la disminución del retardo en los mensajes provocado por la cercanía de los nodos. *Objetivos considerados*: VI, IX.

- **Manejo de la información**: En la arquitectura propuesta, los mensajes a enviar son administrados conforme a la estimación y compresión de los mismos. Se calcula la probabilidad de que un mensaje llegue a tiempo a su destino, la estimación se realiza en base a una distribución de Pareto. Se considera que un mensaje llega a tiempo cuando su tiempo de entrega está dentro de los rangos definidos en los perfiles de usuarios. Al considerar el TTA, es posible modificar los parámetros utilizados para la estimación del retardo. La consideración de la repercusión del envió de los mensajes da como resultado un CLD, para disminuir el retardo de los mensajes enviados. En específico, el CLD fue llevado a cabo por calibración vertical entre capas, al considerar las repercusiones que tienen los mensajes generados que necesitan ser enviados por la red. *Objetivos considerados*: IV, V, VII.

Hoy en día es muy demandado la utilización de gráficos en 3D, debido a que permite mejorar el aspecto de los sistemas, así hacerlos más llamativos a los usuarios finales. Los Ambientes Virtuales en 3D son más atractivos a los usuarios por diversas razones, tales como: aspecto, utilidad, fácil interpretación, diversidad, entre otras. Estas y otras características hacen que los requerimientos para administrar un AV distribuido sean cada vez más complejos en su diseño.

Las investigaciones analizadas involucran tareas para el manejo de la lógica del AV, pero no consideran el cómo representar los resultados al usuario final, tarea realizada por el PV. Un PV debe realizar una visualización en tiempo real, así como permitir la interacción de diversos usuarios localizados en diferentes áreas geográficas. Debido a que un PV es la interfaz entre el AV y el usuario final, se debe tener en cuenta la Calidad de Experiencia del usuario.

En este trabajo de investigación se realizaron diversas tareas con la finalidad de identificar el rol de un PV en un AV, así como identificar los requerimientos de un PV. A continuación se enlistan las características principales y se describen de una forma general.

- **Interacción con el usuario**: En base al MVC se realizó una arquitectura de referencia con la finalidad de identificar el flujo de información en un PV. La arquitectura de referencia QoE-PV toma en cuenta la Calidad de Experiencia (QoE) del usuario y su relación con las características del PV. *Objetivos considerados*: I, II, III.

- **Elementos del AV**: Un AV está formado por un escenario, un conjunto de objetos 3D y un conjunto de avatares. En un AV la entidad avatar es considerada la más compleja para su administración. Al definir las partes que la conforman es posible admitir animaciones apegadas a la realidad. Los comportamientos de los avatares no están basados en secuencias de animación prediseñadas (animaciones estáticas) sino en micro animaciones definidas en la sección 3.2. *Objetivos considerados*: I, II, III.

  Manipulación de los elementos de un AV: Cuando se utilizan animaciones prediseñadas, se imposibilita la representación de cualquier tipo de situación, ya que sólo abarcan situaciones particulares. Por el contrario, cuando se utilizan micro-animaciones para formar macro-animaciones de manera dinámica, es posible representar una amplia gama de situaciones y comportamientos de los avatares y objetos de los AVs 3D. Las micro-animaciones permiten una mejor manipulación de cada elemento, dando como resultado un mayor realismo en las animaciones que son mostradas. En este trabajo mostramos bajo que condiciones e posible utilizar esta aproximación en TR. *Objetivos considerados*: I, II, III.

identificar los objetos que están "a la vista" es *Frustum Culling*, el análisis de este y otros algoritmos similares facilitaría la tarea de generar las AI. Un breve análisis de este tipo de algoritmos es mostrado en el Anexo C.

- Analizar el impacto del cambio de los límites de carga, así como la adaptación de sus valores dependiendo del retardo en la visualización.

- Realizar el manejo de la información utilizando la probabilidad de llegar al estado q_4 de la cadena de Markov propuesta.

- Cuando un AV está en el rango de tiempo del Perfil Medio, el porcentaje de inconsistencia es alto, debido al TTA que debe ser de 292 milisegundos. Para reducir el rango de inconsistencia durante este perfil (principalmente) es necesario analizar métricas que permitan que los mensajes no sobrepasen este límite.

- Diseñar y realizar pruebas subjetivas para la QoE, las cuales sean aplicadas a diferentes grupos de persona. Los grupos de personas podrían ser: profesionistas, adolecentes, adultos, personas con conocimiento en realidad virtual, etc.

  La consistencia se midió de una forma objetiva, la comparación de los resultados obtenidos con resultados de pruebas subjetivas mostraría el impacto del usuario en la medición de la QoE en los PVs.

  Una forma de incrementar la QoE es la utilización del sonido. La incorporación del sonido en los PV conlleva diversas tareas que permitan hacer una propagación del sonido lo más apegado a la realidad. De esta forma identificar a que PV se debe enviar el sonido, así como el grado y tipo de distorsión con el que se debe reproducir, para lo cual se deben tomar en cuenta las características del AV. Características como el material del AV, muros de piedra, montañas, agua, etc., que influyen en la distorsión del sonido.

8.3 Publicaciones generadas

Félix Ramos, Omar González, Miguel Sánchez, Alma Martínez, *"Una plataforma para el mundo real simulado"*. Ciencia y Desarrollo, CONACYT, México. Diciembre 2008.

Jaime Alberto Zaragoza Rios, Alma Verónica Martínez, Félix Ramos, Mario Siller, Véronique Gaildrat. *"Virtual World Creation and Visualization by Knowledge Based Modeling"*. GAMEON-NA, Atlanta, USA, 26 al 28 de Agosto del 2009, EUROSIS, p. 5-9, Agosto 2009.

Los *AVs* hasta ahora analizados, contemplan los tres módulos presentes en una arquitectura *MVC*. Esto quiere decir que no sólo se está evaluando la visualización de los *AVs* (tarea llevada a cabo por un *PV*), sino también el comportamiento de las entidades y objetos presentes en los mismos, impidiendo realizar una comparación.

8.1 Descripción de aportaciones

- Se realizó la descripción de un *AV*, se definieron las entidades por las que está formado y su forma de interacción. Para realizar una correcta administración del *AV* se identificó la secuencia de pasos que provoca la interacción de las entidades, es decir, como evoluciona el *AV*.

- Se diseñó una arquitectura de referencia útil para sistemas que puedan compartir el tipo de visualización requerida, y necesiten de una la evaluación del usuario final.

- Se identificaron las características necesarias para la distribución de Puertos Visuales, las cuales fueron consideradas para el diseño de una arquitectura *P2P*. Una de las características principales de la arquitectura es la visualización en tiempo real, para lo cual se definió un límite basado en cualidades de la vista humana.

  La Calidad de Experiencia en los *AV* es una reciente área de investigación, en trabajos anteriores las métricas fueron descritas para *AV* inmersos. En un *PV* no es posible implementar todas las métricas de la *QoE* de *AV* inmersos, por lo cual se realizó una clasificación de las métricas que pueden ser consideradas en los *PV*. A este último conjunto de métricas se agregó las físicas y sus características, métrica que hasta el momento no había sido contemplada en la *QoE* de los *AV*.

- Se definió un método para medir la consistencia entre Puertos Visuales de una manera objetiva, utilizando un *CLD* de calibración vertical. El *CLD* está en base a la estimación del retardo y a la compresión de los mensajes, permitiendo un manejo de la información entre los *PVs*.

- Se diseñó un conjunto de algoritmos para la auto organización de nodos que brindan un *PV* como servicio. El comportamiento de dichos algoritmos dan como resultado una arquitectura de red adaptativa.

8.2 Trabajo futuro

Las Áreas de Interés están definidas o por lo que alcanza a percibir el avatar o lo que permite ver las características del *AV*. Un método en el área 3D que

Anexo A

Como parte del trabajo, se realizó un análisis detallado de las herramientas que pudieran ser utilizadas en la capa de procesamiento local, en específico para la tarea de la representación gráfica en 3D. Dichas herramientas son los motores de 3D y los motores de juegos de vídeo.

Un motor 3D se encarga sólo de la manipulación de recursos en 3D, tales como avatares y escenarios, los cuales son diseñados previamente en un editor de 3D (usando Maya, 3DStudioMax, etc.) y pueden ser manipulados por un lenguaje de programación.

Un motor de juegos permite la administración de recursos 3D, por lo general los utilizados en vídeo juegos ya establecidos como Quake. Estos juegos incluyen librerías para el manejo de sonido, hardware especializado, inteligencia artificial, etc. Pero la desventaja que presentan es que al ser diseñados para vídeo juegos, se limitan las capacidades de manipulación de las entidades presentes en los AVs.

Las principales características que se buscaron en los motores 3D y los motores de video juegos fueron:

- ***Animaciones por esqueleto:*** si estos permiten la manipulación en tiempo de ejecución de cada empalme (articulación) de una entidad, así no se depende de animaciones prediseñadas en un editor 3D.

- ***Creación del ambiente:*** esta debe ser completada en tiempo de ejecución para permitir el diseño del ambiente con cualquier escenario.

Otro tipo de herramientas que permiten que la evolución del AV sea más apegada al mundo real, son las físicas, para lo cual algunos de los motores, ya sea 3D o de juegos,

Alma Martínez, Félix Ramos, Mario Siller. *"Visualización Distribuida en Tiempo Real de un Ambiente Virtual"*. Semana Nacional de Ingeniería Electrónica (Senie09), Ocotlán, Jal. Del 7 al 9 de Octubre del 2009.

Alma V. Martínez González, Héctor Orozco Aguirre, Félix Ramos Corchado, and Mario Siller. *"A Peer-to-Peer Architecture for Real-Time Distributed Visualization of 3D Collaborative Virtual Environments"*. The 13-th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2009), Singapore, del 25 al 28 de Octubre del 2009.

Alma Martínez G., Héctor Orozco A., Félix Ramos C., Mario Siller. *"Real Time Visualization for Distributed Virtual Environments"* To be appear in the Journal "Research in Computing Science", CORE CIC 2010, 11° Conference in Computing. May 26-28, 2010, México City, México.

- ***SED Script Editor:*** mediante el lenguaje LITE-C es posible asignar comportamientos a las entidades del ambiente.

- ***WED Level Editor:*** por medio de este editor es posible indicar las características de los niveles del ambiente, es decir, las posiciones iníciales de cada elemento en el ambiente.

El uso de estos tres editores facilita el diseño del ambiente, haciendo que sea una herramienta sencilla de utilizar por principiantes. Las características ofrecidas son las principales de un motor de juegos, como son: manejo de partículas, luces, colisiones, soporte para arquitecturas cliente-servidor, entre otras.

OGRE (Object-Oriented Graphics Rendering Engine)

Es un motor 3D escrito en C++, el cual puede ser utilizado sobre OpenGL y Direct3D. OGRE [**103**] es uno de los motores 3D más completos, ya que cumple con diferentes características necesarias para una aplicación 3D aceptable. El motor no está diseñado para la creación de juegos, pero es fácil integrarle diferentes librerías necesarias para la creación de juegos, como son el sonido, redes, etc.

Por último, este motor da soporte a animaciones por esqueleto, así como el manejo de las entidades en el ambiente. Maneja colisiones entre las entidades, sombras dinámicas y manejo de partículas.

Torque Game Engine (TGE)

TGE [**104**] es un motor de juegos para multi-jugadores, el cual permite realizar animaciones por esqueleto, permitiendo tener una mejor manipulación de los avatares. Este motor contiene un editor de mundos, el cual utiliza script en C-Like. Pero a diferencia de la mayoría de los motores de juegos comerciales, la licencia de bajo costo incluye el código fuente en C++, por lo que es posible hacerle las adaptaciones necesarias.

Es posible utilizar este motor tanto en OpenGL como en DirectX, es multiplataforma (Windows, Linux, MacOS). Otra de las características que posee es el manejo de sombras, físicas básicas, luces, animaciones por esqueleto, entre otras.

Virtools 4

Virtools [**105**] es un software enfocado a la creación de aplicaciones interactivas con gráficos 3D en tiempo real, sus usos van desde visualizaciones de entornos, simulaciones físicas en tiempo real y video juegos.

implementan librerías propias para un manejo básico de estas. Sin embargo, existen motores de físicas especializados, los cuales permiten una mejor manipulación de las propiedades físicas de los elementos en el ambiente.

Motores 3D y motores de juegos

Cube

CUBE [**100**] es un motor que brinda las características para el desarrollo de vídeo juegos de tipo multi-jugador de primera persona (First Person Shooter *FPS*). Este motor es capaz de manejar iluminación de vértice de grano fino, puede hacer luces dinámicas y sombras.

Los objetos que pueden ser cargados en el juego son los utilizados por el juego Quake, limitando la manipulación de los avatares. Este motor es útil para juegos *FPS*, ya que contiene todas las características necesarias para su desarrollo, pero no para el diseño de un *AV*, en el cual es necesario tener más control sobre cada parte de los avatares.

G3D

Es utilizado en juegos, demos de alta tecnología, trabajos de investigación, simuladores militares y cursos universitarios. Este motor proporciona, en un conjunto de bibliotecas de funciones, rutinas y estructuras comunes necesarios en casi todos los programas gráficos.

Una desventaja es que se deben tener conocimientos de programación en 3D. G3D [**101**] está destinado a usuarios que ya están familiarizados con C ++ y DirectX u OpenGL. La biblioteca también proporciona interfaces de bajo nivel para programadores con OpenGL. G3D permite compilar y ejecutar el código en Windows, FreeBSD, Linux y OS X sin sentencias de tipo "ifdef" o de modificaciones. G3D es compatible con todas las implementaciones de OpenGL, es decir, es independiente de plataformas y permite el acceso a las nuevas características de DirectX. Es posible utilizar los elementos del juego Quake.

Gamestudio/A7

Gamestudio/A7 [**102**] es un motor para el diseño de entornos virtuales tanto en 3D como en 2D. El diseño se realiza por medio de tres editores, lo cuales son:

MED Model Editor: permite el diseño de las entidades utilizadas en los Ambientes Virtuales.

NVIDIA PhysX SDK

NVIDIA PhysX SDK [**107**] es un conjunto de bibliotecas de funciones para el manejo de físicas y es además multiplataforma (incluyendo consolas de video juegos). Con estas funciones, se permite mediante el uso de dinámica de los cuerpos, simular objetos con un alto grado de realismo. Ya que en estas funciones, se aplican conceptos físicos tales como posición, velocidad, aceleración, movimiento, fuerzas, movimiento de rotación, manejo de energía, fricción, impulso, colisiones, etc. Al aplicar estas funciones, se crea un informe que puede ser considerado por los desarrolladores en futuras acciones.

Por último, NVIDIA PhysX SDK permite la simulación de líquidos y gases mediante un sistema de partículas. Otra de sus características es el manejo de telas, así como el manejo de campos de fuerza utilizados para controlar tanto la cantidad y la dirección de las mismas.

ODE (Open Dynamics Engine)

ODE [**108**] es una librería para el manejo de físicas, escrita en C++. Útil en la simulación de las estructuras de cuerpos articulados. Una estructura articulada se crea cuando los cuerpos rígidos de formas diversas están conectados entre sí con uniones de diferentes tipos. Ejemplos de ello son los vehículos terrestres (cuando las ruedas están conectadas al chasis), avatares (donde las piernas se conectan al cuerpo) o bien pilas o cúmulos de objetos.

ODE está diseñado para ser utilizado en servicios interactivos o de simulación en tiempo real. Es especialmente bueno para la simulación de objetos en movimiento. Es rápido, robusto y estable, así el usuario tiene una total libertad para cambiar la estructura del sistema aun cuando la simulación se esté ejecutando.

ODE tiene incorporado un sistema de detección de colisiones. Sin embargo, es posible ignorarlo y hacer la detección de una forma independiente. Para el manejo de las colisiones, se usan primitivos como: esferas, cajas, cilindros, planos y mallas triangulares. El sistema anticolisión de ODE proporciona una identificación rápida de la intersección de los objetos potenciales, a través del concepto de "espacios interceptados"

Motor de físicas elegido

El motor que se utiliza en este trabajo de investigación es Havok Physics, ya que contiene las características necesarias para la aplicación de físicas, con las cuales es posible dar más realismo al ambiente. Otra de las ventajas es su posible integración con otras librerías de Havok, para el manejo de destrucción, ropa, entre otras.

Virtools es un poderoso Game Engine, con un flujo de trabajo rápido, práctico y eficiente, es utilizado en muchas empresas enfocadas al desarrollo de juegos, debido a la facilidad y rapidez con que se pueden obtener excelentes resultados. El comportamiento del ambiente es dado por medio de la conexión de bloques, programados en C++. Permite la animación por esqueleto, luces, sombras, detección de colisiones, etc.

Motor 3D elegido

De los motores analizados el que permite cumplir con las características deseadas es OGRE, los demás están limitados en la forma en que se debe manipular los *AVs* a crear. OGRE no está solo diseñado para un único tipo de juego en especial, sino que permite ampliar las posibilidades de los tipos de aplicaciones para las cuales puede ser utilizado.

Con OGRE, si se desea crear un ambiente donde interactúen diversos usuarios, el cual es el caso expuesto en este documento, no se está limitado a una estructura de comunicación como en GameStudio/A7. O bien como en el caso de Cube, donde se está limitado al generar sólo comportamientos para vídeo juego de tipo *FPS*, además de que para las animaciones a usar, no se requiere que sean realistas. En el caso de G3D al utilizar los recursos de Quake, no se permite una administración total de cada empalme de los elementos en la escena. Por otra parte, Torque y Virtools no permiten la creación del ambiente en tiempo de ejecución, lo cual es una característica deseable para los *AVs* para crear en una simulación realista.

Motores de físicas

Havok Physics ™

Este es un motor de físicas utilizado en vídeo juegos, el cual recrea interacciones entre objetos y personajes del juego, detecta colisiones, maneja la gravedad, la masa y la velocidad en tiempo real llegando a recrear ambientes mucho más realistas y naturales.

Havok Physics [**106**] está optimizado para su uso con Xbox 360, PlayStation 3, PlayStation 2, PSP, Wii y PC. El SDK proporcionado es multi-tema y ofrece soporte en las siguientes áreas:

- Detección de colisiones
- Resolución de dinámica
  Control de personajes "out of the box"
  Depuración visual
  Integración total con toda la suite de productos Havok (Animation, Behavior, Cloth, Destruction, etc).

Anexo B

Físicas en un Ambiente Virtual

En este anexo se mostraran las librerías diseñadas para aplicar físicas en un *AV*. El motor de físicas elegido es Havok **[106]**. El diagrama de clase de las librerías se muestra en la **Figura 48**.

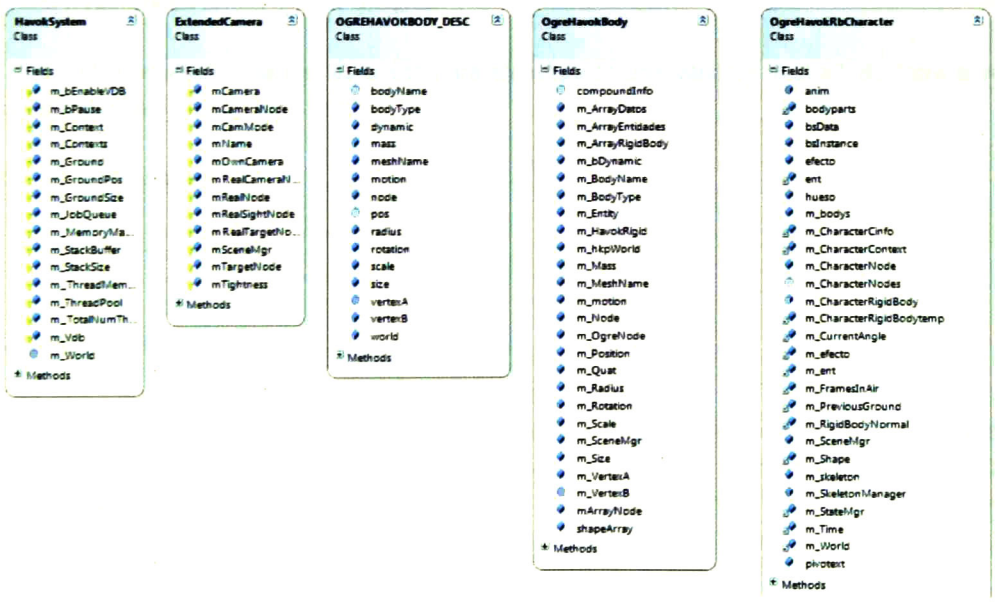


Figura 48: Librerias para el manejo de físicas.

Aunque NVIDIA PhysX SDK ofrece también el manejo de ropa, presenta una limitación al tener que utilizar un hardware de NVIDIA para tener un mejor rendimiento. En lo que respecta a ODE, este no da soporte para ropas o destrucción de objetos, para lo cual sería necesario implementar librerías externas que dieran soporte a estas características.

OgreExtendedCamera

La librería provee diferentes métodos para la manipulación de la cámara en el mundo de Ogre desde primera persona, una vista global, o realizar un auto Tracking al asignar la cámara y el objetivo.

Ejemplo 3:

```
ExtendedCamera* m_ExtCam;
Camera* mCamera;
⋮
m_ExtCam    new ExtendedCamera("CharacterExtCam", mSceneMgr,
           m_HavokRbCharacter->getCharacterNode(0),mCamera);
m_ExtCam->setCameraMode(CAMERA_MODE_WORLD);
```

Util

La librería transforma diferentes estructuras básicas de Havok a Ogre y viceversa. Las posibles conversiones son:

- Vector3 a Vector4 (Ogre → Havok).
- Quaternion a Quaternion (Ogre → Havok).
  Vector4 a Vector3 (Havok → Ogre).

El ejemplo 4 muestra los pasos para transformar una variable Vector3 de Ogre a su correspondiente en Havok (hkVector4).

Ejemplo 4:

```
Ogre::Vector3 vec (30,30,30);
hkVector4 vecH;
vecH    OgreTohkVector4(vec);
```

OgreHavokRbCharacter

Esta librería contiene los métodos necesarios para crear avatares en el mundo de Havok. El objeto que representa a un avatar es una figura con propiedades diferentes a un objeto 3D, contiene propiedades como: la inclinación máxima, actualizar sus movimientos, la agregación de diferentes estados para una mayor manipulación de los avatar (estados como: "saltar en el aire"), etc.

La actualización se realiza pasando los parámetros de Ogre a Havok, una vez obtenido el resultado del lado Havok son enviados a Ogre. Los parámetros requeridos para realizar una actualización del avatar son: el ángulo, posiciones en los ejes X y Y, si el avatar realizara alguna acción.

HavokSystem

Esta librería administra la creación y configuración de las físicas en Ogre, así como el mundo de Havok. Los métodos se diseñaron de tal forma que fuera fácil su manejo, esto a través de minimizar los argumentos requeridos. Existen dos métodos para crear un mundo:

a) Las características del mundo a crear son dadas por medio de los parámetros, como se muestra en el ejemplo 1:

Ejemplo 1:

```
HavokSystem    m_Havok;
m_Havok    new HavokSystem();
//El tamaño del Espacio de Fisicas
m_Havok->createHavokWorld(500);
//La Posicion del Terreno
m_Havok->setGroundPos(0,-2,0);
//El tamalo del Terreno
m_Havok->setGroundSize(5000,2,5000);
//Inicializa el Visual Debugger de Havok
m_Havok->InitVDB();
//Inicializa las Fisicas del Mundo.
m_Havok->createPhysicsScene();
```

b) Se crea un mundo con características predefinidas, ejemplo 2:

Ejemplo 2:

```
/*
* Se crea un mundo con las siguientes caracteristicas
  Tamaño del Terreno X,Y,Z respectivamente 100.0f,0.01f,5.0f;
  Posicion del Terreno en X Y Z respectivamente 0.0f,0.0f,0.0f;
  Con un Tamaño del mundo de 200,000
*/
HavokSystem    m_Havok;
m_Havok->setup();
```

El *Visual Debugger* de Havok es una aplicación para visualizar el mundo de Havok, permite un mayor entendimiento de la evolución del Ambiente Virtual. Este programa se encuentra adentro del SDK de Havok, ruta: *Tools → VisualDebugger→ hkVisualDebugger.exe.*

La visualización en Ogre y Havok puede ser llevada a cabo de forma paralela, en ventanas (aplicaciones) independientes, de tal forma que al cerrar una de estas no afecta el comportamiento de la otra.

Figura 49: Mundos en Ogre y Havok.

Anexo C

Algoritmos de visibilidad

La visibilidad [**109**] (Visibility Culling) describe el proceso de eliminación de objetos de la escena que no sean visibles en un frame renderizado, disminuyendo el trabajo de renderizado. Este tipo de algoritmo se utiliza para disminuir el trabajo del motor de render, pero también puede ser utilizado para detectar los objetos que pertenezcan a una AI. Se realizó un breve análisis de algunos de estos algoritmos, los cuales se muestran a continuación.

Superficies Ocultas

Los algoritmos de superficies ocultas (Backface Culling) analizan los objetos visibles y divide la escena en dos planos: frente y detrás. Solo el plano que contiene los frentes es renderizado. Un inconveniente es que requiere mayor capacidad de procesamiento, para realizar la separación, por lo cual no es comúnmente utilizado en $GPUs$.

Descarte por área

El área de visión de las cámaras está delimitada por una pirámide, la cual tiene una profundidad para determinar cuáles objetos están a su alcance. Este método es conocido como "View-Frustum Culling" Es posible utilizar otras figuras geométricas como cajas en lugar de una piramide. Debido a su simplicidad, generalidad y características de rendimiento, este tipo de algoritmos siguen siendo los más comunes en video juegos.

Ejemplo 5:

```
OgreHavokRbCharacter* m_Avatar;
⋮
m_Avatar   new OgreHavokRbCharacter(m_Havok->gethkpWorld());
m_Avatar->createCharacter(nodo, position);
⋮
//Verifica si esta vacio
if (! m_Avatar->m_bodys.isEmpty())
   //Parametros:
   //1.- Actualizara el avatar con el id 1.
   //2.- Tiempo del frame.
   //3.  Angulo de movimiento que sufrio el avatar.
   //4.- Movimiento en X.
   //5.- Movimiento en Y.
   //6.- Indica si se realiza la animacion.
   m_Avatar->updatecharacter(1, evt.timeSinceLastFrame, 0, 10, 20, false);
```

OgreHavokBody

En esta librería es donde se crean los objetos 3D en el mundo de Havok, no se necesitan actualizar como a los avatar ya que reaccionan según su interacción con el ambiente. Al crear un objeto se le dan las propiedades físicas como: mesh, posición, rotación, masa, figura geométrica, escala, etc.

La librería contiene métodos para facilitar la creación de figuras geométricas básicas, de igual manera se agrega un método para actualizar los objetos, la actualización se realiza de todos los objetos a la vez.

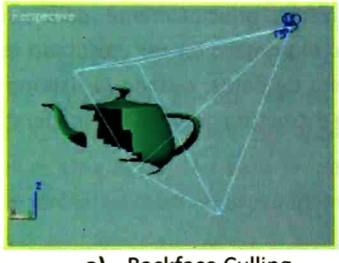
Ejemplo 6:

```
OgreHavokBody *m_bodys;
m_bodys   new OgreHavokBody();
OGREHAVOKBODY_DESC *desc   new OGREHAVOKBODY_DESC ();
//Creacion de una caja
desc->world   mHavokSystem->m_World;
desc->sceneMgr   mSceneMgr;
desc->bodyName   bodyname;
desc->meshName   meshname;
desc->bodyType   BODY_TYPE_BOX;
desc->mass   100.0f
desc.dynamic   false;
desc.radius   5;
desc->size=hkVector4(10,5,2);
desc->pos   p;
desc->dynamic   true;
desc->motion   motion;
rotation.normalize();
desc->rotation   rotation;
desc->scale   Ogre::Vector3(1,1,1);
mBodys->addBody(desc);
⋮
//Actualizacion
m_bodys->update();
```

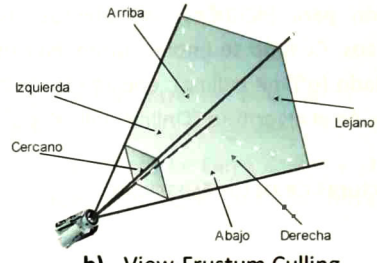
Una jerarquía de $AABB$ se organiza en una estructura de árbol, y distingue entre tres tipos de nodos diferentes:

- ***Raíz:*** es el primer nodo de la jerarquía.
- ***Hoja:*** los nodos de la hoja no tiene ningún nodo secundario, contienen la geometría real del objeto.
- ***Internos:*** internos almacenan punteros a los nodos de sus nodos hijo.

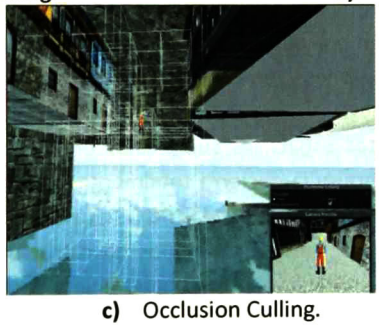
Existen diversas estrategias que se puede utilizar para construir una jerarquía [114] [115], como la heurística de superficie (surface area heuristic - SAH) [116]. Cabe señalar que (según el conjunto de datos y la heurística) $BVHs$ puede tomar algún tiempo para construir, por lo que son pre-calculados para la geometría estática. Para la geometría dinámica, una jerarquía puede ser reconstruida en cada fotograma utilizando la heurística espacial de división mediana.



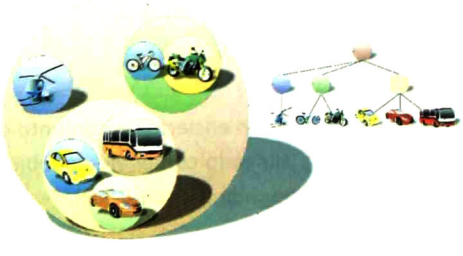
a) Backface Culling.



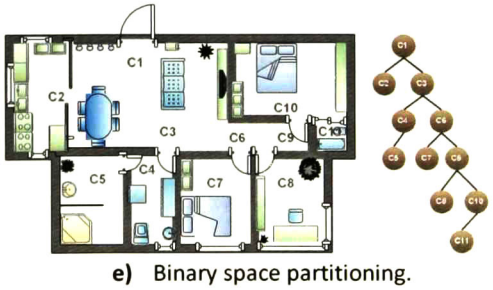
b) View-Frustum Culling.



c) Occlusion Culling.



d) Bounding volume hierarchies.



e) Binary space partitioning.

Figura 50: Algoritmos de visibilidad.

Oclusión

La Oclusión (Occlusion Culling) detecta los objetos que son completamente ocultos por otros objetos. Determinar el conjunto de primitivas ocultas es fácil en teoría, pero puede ser difícil de implementar en un contexto de renderizado en tiempo real. Algunos algoritmos de este tipo son los más complejos de implementar.

El rendimiento de este tipo de algoritmo depende del escenario, si no existen muchos obstáculos, el algoritmo podría ser una sobrecarga. Muchos de los algoritmos de oclusión se han desarrollado recientemente, algunos de ellos utilizando la funcionalidad del hardware disponible [**110**], otros se basan en soluciones de software [**111**] [**112**] [**113**].

En tiempo de ejecución y pre-calculados

Ambos tipos de algoritmos utilizan estructuras de datos donde almacena celdas y su relación para identificar los objetos visibles. Son utilizados principalmente para escenarios estáticos. Cuando se implementan este tipo de algoritmo el primero en ser ejecutado es el pre-calculado (offline culling), encargado de crear la estructura de datos, cuando el sistema está en ejecución el algoritmo "Online culling" utiliza y actualiza la estructura previamente generada.

Estructuras de Datos Espaciales

Las estructuras de datos espaciales reducen drásticamente el número de consultas realizadas, como la intersección o pruebas de visibilidad. Las estructuras de datos son útiles para algoritmos de detección de colisiones y, en general, se pueden aplicar a n-$dimensionales$.

Una estructura de datos espaciales suele estar organizado de forma jerárquica, de tal manera que un objeto en la jerarquía encierra totalmente a todos sus hijos. De esta manera, la complejidad de las consultas puede ser reducidas de O (n) a O (log n), que presenta una velocidad significativa cuando n es grande. Hay muchos tipos de estructuras de datos espaciales, como: jerarquías de volúmenes envolventes (bounding volume hierarchies - BVH), particionamiento del espacio binario (binary space partitioning - BSP), etc.

Jerarquías de volumen envolventes

En las estructuras Jerarquías de volumen envolventes un volumen encierra un conjunto de objetos, y normalmente es un volumen sencillo (como una esfera), alineado con el eje del objeto (axis-aligned bounding box - $AABB$) o una caja orientada (oriented bounding box - OBB).

Anexo D

Creación de terrenos

En ocasiones en un AV es necesario el manejo de terrenos irregulares (mapas tridimensionales). Ogre utiliza dos tipos de mapas: a) un mapa de altos (heightmap), b) un mapa de detalles. Una aplicación para la creación de este tipo de mapas es Earth Sculptor [117]. A continuación se muestra un breve manual de cómo es posible crear y/o modificar un terreno útil para Ogre.

NOTA: la versión utilizada es Earth Sculptor 1.05.

Earth Sculptor es un editor de terrenos en tiempo real con una interface fácil de usar, diseñado exclusivamente para el desarrollo de paisajes. En la Figura 51 se muestra la pantalla principal.

Figura 51: Pantalla principal de Earth Sculptor.

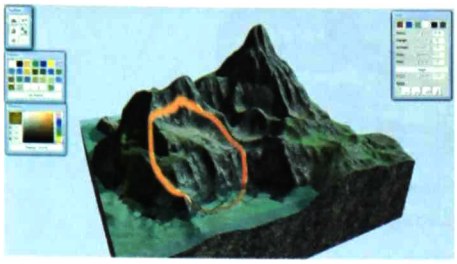
La barra de herramientas (ToolBar), contiene 6 opciones:

Estos algoritmos son utilizados para minimizar el trabajo del render o para la detección de colisiones. En nuestro caso, serían utilizados para delimitar las AI de una forma eficiente. Es necesario un estudio más amplio para seleccionar el o los más adecuados, con este breve análisis los algoritmos que serían de utilidad serian:

- ***View-Frustum Culling:*** para detectar los objetos al alcance de un avatar.

  Occlusion Culling: filtrar los objetos visibles de acuerdo a las características del ambiente.

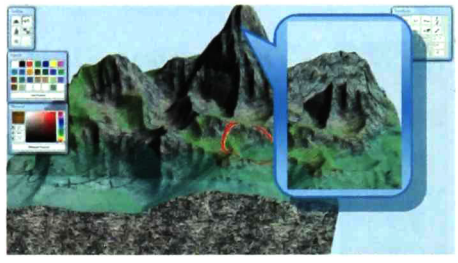
Para un mejor filtrado de objetos visibles y dar más realismo al ambiente se podrían implementar algoritmos que permitan identificar el tipo de objeto, para así saber qué acciones tomar. Por ejemplo: si un avatar se topa con un muro de piedra no podrá cruzar, sin embardo, si es una cascada o un vidrio con poco espesor debería poder cruzar.



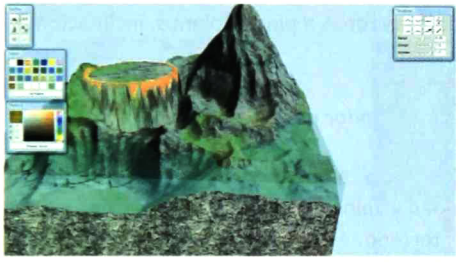
a) Terreno.

b) Color.

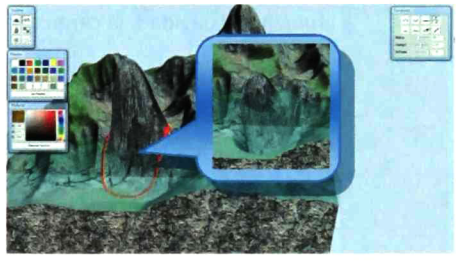
c) Raise.



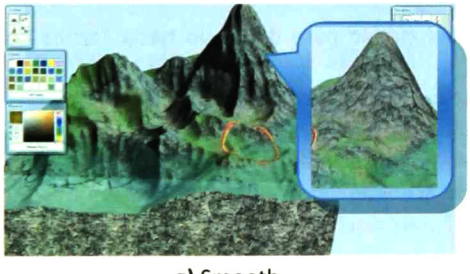
d) Lower.



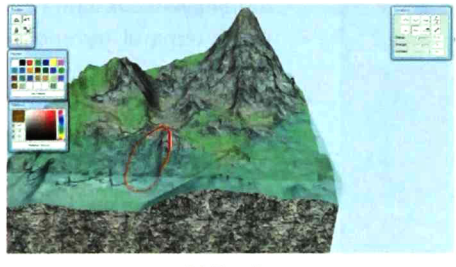
e) Level.



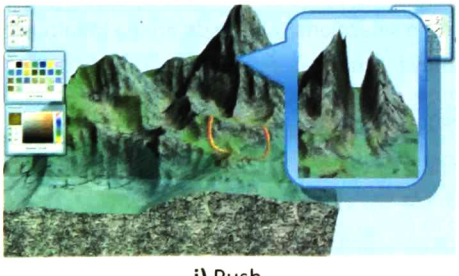
f) Grab.



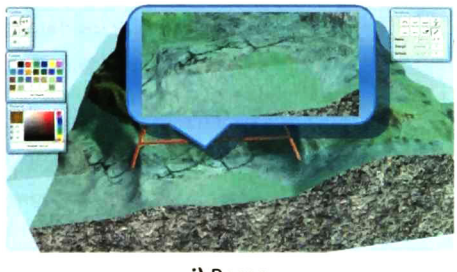
g) Smooth.



h) Erode.



i) Push.



j) Ramp

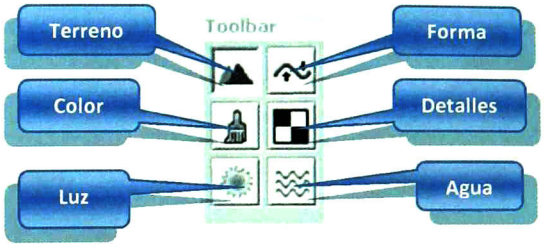


Figura 52: ToolBar

Terreno	• ***Height:*** altura que tendrá el terreno. ***Wireframe:*** permite ver la forma de la malla (mesh) del objeto. ***Grid:*** muestra las redes que conforman al mesh de la textura. ***Color, retail, Lightning:*** opciones para mostrar el mesh. • ***Fog:*** configuración de la niebla, por distancia o por color. • ***Underground:*** permite seleccionar el fondo. • ***Set skybox:*** imágenes para el cielo.
Color	Se asigna por medio de un circulo, con el cual se indica el área del terreno, la fuerza, la suavidad, la cantidad de ruido y las zonas a pintar (planos, inclinaciones, paredes, etc.).
Luz	Permite: dirigir, dirección, intensidad, y resplandor de la luz ambiental.
Terreno	• ***Raise:*** modificar la altura de la superficie del mapa por medio de un círculo para manipular el radio, la intensidad y velocidad. ***Lower:*** permite bajar la altura del terreno. ***Level:*** el terreno dentro del circulo tendrá la misma altura, ya sea elevando o disminuyendo la altura del mapa. ***Grab:*** toma el terreno dentro del círculo para deslizarlo hacia "arriba" o "abajo" • ***Smooth:*** permite dar un acabado curveado (fino) al terreno. ***Erode:*** da el terminado erosionado al área afectada por el círculo dando un resultado "rugoso" que se presenta principalmente en montañas. ***Push:*** simula un dedo para "empujar" el terreno. • ***Ramp:*** permite crear "rampas", el resultado variara dependiendo del "softness" seleccionado.
Detalle	Permite agregar detalles en la textura de su mapa por medio de un círculo. Es posible intercambiar los detalles, así como seleccionar el tipo de área a manipular (planos, semi-inclinados, Inclinados o muy inclinados).
Agua	Agregar agua de forma uniforme. Para agregar agua a un nuevo terreno se debe habilitar la opción de agua que se encuentra en la esquina superior izquierda.

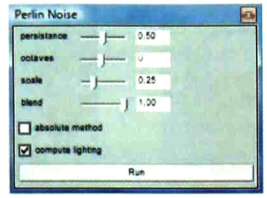
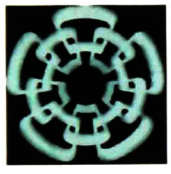


Figura 56: Perlin Noise

Para exportar el trabajo realizado de tal forma que sea útil en Ogre, se exportan las características del terreno por medio de la opción Export:

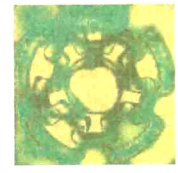
Figura 57: Menú para exportar.

Es posible exportar tanto el mapa de altos, el objeto (mesh) y/o la textura.



a) Mapa de Altos.

b) Mesh.



c) Textura.

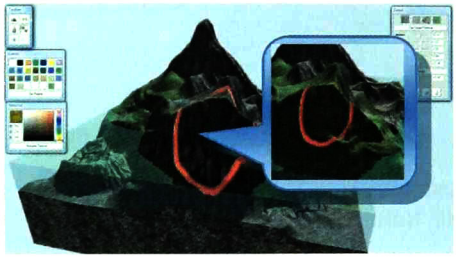
Figura 58: Objetos exportables.

Editar Avatares y objetos 3D

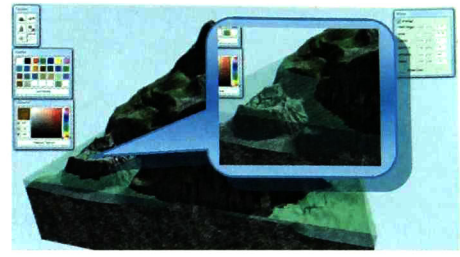
Los avatares y objetos 3D reconocibles por Ogre son almacenados en archivos .mesh, que tienen un formato propio, es decir, no son creados por algún editor 3D. En este anexo se muestran los pasos a seguir para modificar dichos archivos.

Archivo .mesh a .xml

Para poder crear un archivo .xml a partir de un objeto .mesh (Ogre) es necesario tener el siguiente programa: *OgreXMLConverter.exe.* Este programa lo proporciona Ogre en la Sección de *Download* → *Tools* en la parte de *Miscellaneous*, el paquete a descargar se llama *Command-line Tools (1.4)* la fecha de la versión es del día 25 de Junio del 2009. Los pasos para transformar el archivo .mesh a .xml son los siguientes:



k) Detalle



l) Agua

Figura 53: Edición del terreno por medio del ToolBar.

Para la creación de un nuevo terreno existen dos opciones, las cuales se describen a continuación.

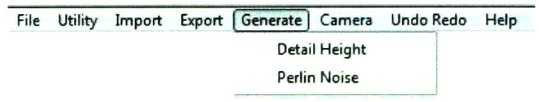
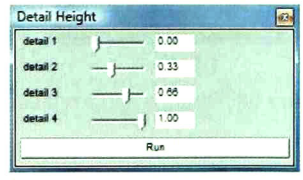


Figura 54: Nuevo terreno.

La opción *Detail Height* despliega una ventana con cuatro opciones para el manejo de la altura y texturas.



a)

b)

c)

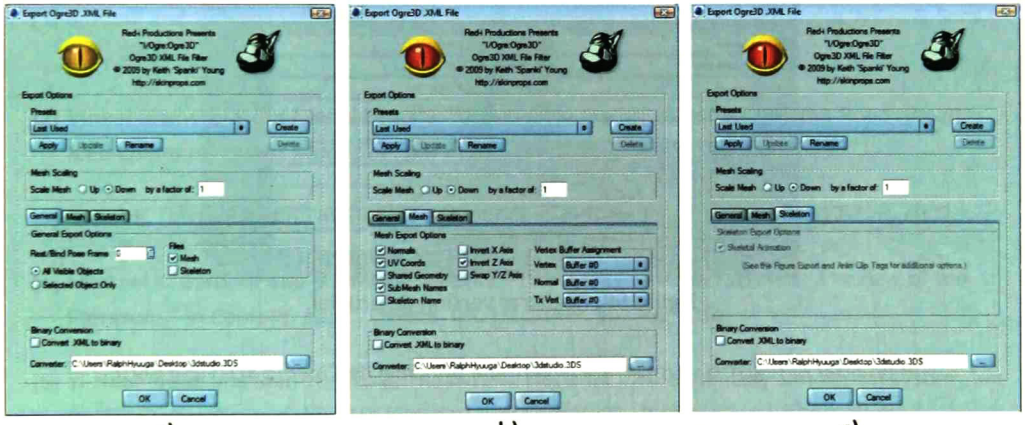
Figura 55: Detail Height

La opción *Perlin Noise* agrupa las opciones: persistencia, octavos, escala y mezcla. Utilizadas para crear un mapa plano, la escala, etc.

Los cambios realizados a *Cinema 4D* son reflejados en el menú de *Plugins*, en la opción *IOgre* (Figura 61).

Figura 61: Opciones del menú IOgre.

La opción *Ogre 3D .xml Importer* permite seleccionar el archivo con el que se desea trabajar. En este punto el archivo es editable, por lo cual una vez finalizadas las modificaciones necesarias se realiza una exportación a .xml. La opción *Ogre 3D .xml Exporter* mostrara la ventana mostrada en la Figura 62.



a) b) c)

Figura 62: Exportar a .xml.

Esta ventana contiene 3 pestañas:

- ***General***: contiene las opciones para exportar solo el mesh, el esqueleto o ambas (Figura 62 a).

- ***Mesh***: permite invertir los ejes X, Y, Z. Es recomendable no modificar esta pestaña para no ocasionar cambios al mesh (Figura 62 b).

1. Abrir una *Línea de Comandos*
2. Ir a la Carpeta de Ogrecommandline.
3. Escribir *OgreXMLConverter.exe* + espacio + el archivo que quieran transformar en XML.

NOTA: en el ejemplo el archivo *pared.mesh* está en el directorio de *OgreXMLConverter.exe*, si el archivo .mesh se encuentra en otro directorio, es necesario poner la ruta absoluta.

4. Cuando se presione *Enter*, saldrá en pantalla información acerca de la transformación del archivo .mesh. Finalmente el archivo con el formato XML se creara en el mismo nombre y directorio del archivo .mesh. En este ejemplo el nombre del archivo XML será: *pared.mesh.xml*.

Archivo .xml a .c4d

Para manipular los archivos .xml necesario descargar un plugin llamado *IOgre* utilizado por el software *Cinema 4D* y pegarlo en la carpeta llamada *Plugins*. El siguiente paso es inicializar *Cinema 4D*, debido a que es una versión de prueba debemos escribir *DEMO* en lugar del número de serie (Figura 59):

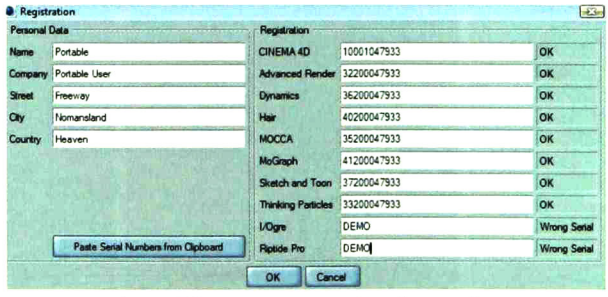


Figura 59: Registrar versión de prueba.

Al finalizar el registro se mostrara la ventana principal de *Cinema 4D* (Figura 60).

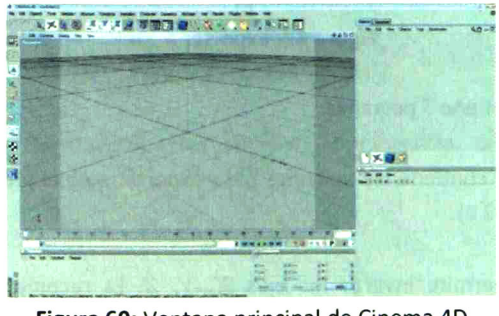


Figura 60: Ventana principal de Cinema 4D

Referencias

[1] S. Jouni and H. Harri, *Algorithms and Networking for Computer Games.*: John Wiley & Sons, Ltd., 2006.

[2] Vineet Srivastava and Mehul Motani, "Cross-Layer Design: A Survey and the Road Ahead," *IEEE Communications Magazine*, pp. 112-119, 2005.

[3] R. Robb, "Virtual Reality in Medicine: A Personal Perspective," *J. Vis.*, pp. 317-326, 2002.

[4] Michael K. Barbour and Thomas C. Reeves, "The reality of virtual schools: A review of the literature," in *Comput. Educ.*, Oxford, UK, UK, 2009, pp. 402-416.

[5] Yi-Haur Shiau and Shin-Jye Liang, "Real-Time Network Virtual Military Simulation System," in *IV '07: Proceedings of the 11th International Conference Information Visualization*, Washington, DC, USA, 2007, pp. 807-812.

[6] Tina Manoharan, Hamish Taylor, and Paul Gardiner, "Interactive Urban Development Control with Collaborative Virtual Environments," in *VSMM '01: Proceedings of the Seventh International Conference on Virtual Systems and Multimedia (VSMM'01)*, Washington, DC, USA, 2001, p. 809.

[7] Kynan Eng et al., "An Investigation of Collective Human Behavior in Large-Scale Mixed Reality Spaces," *Presence: Teleoper. Virtual Environ.*, pp. 403-418, 2006.

- ***Skeleton*:** si el archivo contiene animaciones y se desean exportar, se debe seleccionar la opción *Skeletal Animation* (Figura 62 c).

El archivo .xml obtenido puede ser exportado a .mesh y/o .skeleton con los pasos descritos en la siguiente sección. El plugin se encuentra para su descarga en el sitio: http://skinprops.com/download.php?list.13.

Archivo .xml a .mesh

Los pasos para transformar el archivo .xml a .mesh son los siguientes:

1. Abrir una *Línea de Comandos*.
2. Ir a la Carpeta de Ogrecommandline.
3. Escribir *OgreXMLConverter.exe* + espacio + El archivo que quieran transformar en XML.
4. Al dar Enter saldrá en pantalla información acerca de la transformación del archivo .xml. Finalmente el archivo .mesh se localizara en el directorio del archivo .xml. El nombre del archivo .mesh será: pared.mesh.

NOTA: Utilizaremos el archivo creado en el Ejemplo anterior "pared.mesh.xml"

Al transformar un archivo .xml a .mesh, el programa le quitara la terminación .xml, la cual debe ser agregada de manera manual.

CA, 1995, pp. 85-92.

[21] Richard C. Waters et al., "Diamond Park and Spline: Social Virtual Reality with 3D Animation, Spoken Interaction, and Runtime Extendability," in *Presence: Teleoperators and Virtual Environments*, 2000, pp. 461-481.

[22] S. Douglas, E. Tanin, A. Harwood, and S. Karunasekera, "Enabling Massively Multi-Player Online Gaming Applications on a P2P Architecture," in *Proceedings of the IEEE International Conference on Information and Automation*, Colombo, Sri Lanka, 2005, pp. 7-12.

[23] C. Bouras, D. Psaltoulis, C. Psaroudis, and T. Tsiatsos, "Multi-User Layer in the EVE Distributed Virtual Reality Platform," in *In Proceedings of the 23rd international Conference on Distributed Computing Systems*, Washington, DC, 2003.

[24] Qin X., "Delayed Consistency Model for Distributed Interactive Systems with Real-time Continuous Media," *Journal of Software*, pp. 1029 - 1039, 2002.

[25] Li Min, Zhao Xin, Yu Bin, Lu Guizhang, and Liu Jingtai, "Optimizing Virtual Environment for Micro Operation Using Genetic Algorithm," in *4* World Congress on Intelligent Control and Automation*, Shanghai, P.R. China, 2002, pp. 10-14.

[26] Tao Ruan Wan and Wen Tang, "Simulating Virtual Character's Learning Behaviour as An Evolutionary Process Using Genetic Algorithms," in *The 10-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2002*, Plzen - Bory, Czech Republic, 2002.

[27] C. Sanza, C. Panatier, H. Luga, and Y. Duthen, "Adaptive behavior for cooperation: a virtual reality application," in *Robot and Human Interaction, 2000. RO-MAN '00. 8th IEEE International Workshop on*, 2000, pp. 76-81.

[28] Alejandra Garcia-Rojas. Virtual Human Ontology. [Online]. http://vrlab.epfl.ch/~alegarcia/DSW/VHO.html

[29] Siddhartha Chaudhuri, Daniel Horn, Pat Hanrahan, and Vladlen Koltun, "Distributed Rendering of Virtual Worlds," in *Technical Report CSTR 2008-02*, Computer Science Department, Stanford University, 2008.

[30] Jonathan Strasser, Valerio Pascucci, and Kwan-Liu Ma, "Multi-Layered Image Caching for Distributed Rendering of Large Multiresolution Data," in *In Proceedings of Eurographics Symposium on Parallel Graphics and Visualization*, Lisbon, Portugal , 2006, pp. 171-177.

[31] Risto Rangel-Kuoppa, Carlos Avilés-Cruz, and David Mould, "Distributed 3D Rendering System in a Multi-agent Platform," in *ENC '03: Proceedings of the 4th Mexican International*

[8] Mario Siller, "An Agent-Based Platform to Map Quality of Service to Experience in Active and Conventional Networks," 2006.

[9] Larry L. Peterson and Bruce S. Davie, *Computer Networks: a systems approach*. San Francisco, CA: Morgan Kaufmann Publishers, 2007.

[10] Ian Sommerville, *Ingeniería del Software*. Madrid, España: Pearson Addison Wesley, 2005.

[11] Jouni Smed, Timo Kaukoranta, Harri Hakone, Oy L M Ericsson Ab, and Telecom Rd, "A Review on Networking and Multiplayer Computer Games," in *Technical Report 454*, 2002.

[12] Abdelwahab Hamam, Mohamad Eid, Abdulmotaleb El Saddik, and Nicolas D. Georganas, "A quality of experience model for haptic user interfaces," in *2008 Ambi-Sys workshop on Haptic user interfaces in ambient media systems (HAS '08)*, Brussels, 2008, pp. 1 - 6.

[13] Wanmin Wu et al., "Quality of experience in distributed interactive multimedia environments: toward a theoretical framework," in *17th ACM international conference on Multimedia (MM '09)*, New York, 2009, pp. 481 - 490.

[14] Wanmin Wu et al., ""I'm the Jedi!" - A Case Study of User Experience in 3D Tele-immersive Gaming," in *2010 IEEE International Symposium on Multimedia (ISM '10)*, Washington, DC, 2010, pp. 220 - 227.

[15] Husein M. Ahmed, Denis Gracanin, Ayman Abdel-Hamid, and Kresimir Matkovic, "An Approach to Interaction Interoperability for Distributed Virtual Environments," *In Short Paper Proceedings of the 14th Eurographics Symposium on Virtual Environments (EGVE 2008)*, p. 35—38, 2008.

[16] Jauvane C. de Oliveira, Shervin Shirmohammadi, and Nicolas D. Georganas, "A Collaborative Virtual Environment for Industrial Training," in *VR '00: Proceedings of the IEEE Virtual Reality 2000 Conference*, Washington, DC, USA, 2000, p. 288.

[17] Shervin Shirmohammadi and Nicolas D. Georganas, "Collaborating in 3D Virtual Environments: A Synchronous Architecture," in *WETICE '00: Proceedings of the 9th IEEE International Workshops on Enabling Technologies*, Washington, DC, USA, 2000, pp. 35-42.

[18] Sandeep Singhal and Zyda Zyda, *Networked Virtual Environments: Design and Implementation*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999.

[19] Pedro Morillo Tena. Departamento de Informatica, Universidad de Valencia. [Online]. http://informatica.uv.es/~pmorillo/papers/pmorillo_cisic03.pdf

[20] Thomas A. Funkhouser, "RING: A ClientServer System for Multi-User Virtual Environments," in *ACM SIGGRAPH Special Issue on 1995 Symposium on Interactive 3D Graphics*, Monterey,

conference, Lisboa, Portugal, 2006, pp. 1-9.

[43] P. Svoboda, W. Karner, and M. Rupp, "Traffic Analysis and Modeling for World of Warcraft," in *Communications, 2007. ICC '07. IEEE International Conference on*, 2007, pp. 1612-1617.

[44] Ricky A. Bangun and E. Dutkiewicz, "Modeling Multi-Player Games Traffic," in *ITCC '00: Proceedings of the The International Conference on Information Technology: Coding and Computing (ITCC'00)*, Washington, DC, USA, 2000, p. 228.

[45] Michael S. Borella, "Source Models of Network Game Traffic," in *Computer Communications*, 2000, pp. 403-410.

[46] H. Park, T. Kim, and S. Kim, "Network Traffic Analysis and Modeling for Games," in *Internet and Network Economics*, Berlin / Heidelberg., 2005.

[47] Grenville Armitage, Mark Claypool, and Philip Branch, *Networking and online games : understanding and engineering multiplayer internet games.*: John Wiley, 2006.

[48] A. Bharambe, J. Pang, and S. Seshan, "Colyseus: a distributed architecture for online multiplayer games," in *NSDI'06: Proceedings of the 3rd conference on 3rd Symposium on Networked Systems Design & Implementation*, Berkeley, CA, USA, 2007, p. 12.

[49] Kuan-Ta Chen, Polly Huang, Chun-Ying Huang, and Chin-Laung Lei, "Game Traffic Analysis: An MMORPG Perspective," in *Computer Networks*, Stevenson, Washington, USA, 2006, pp. 3002-3023.

[50] Feng Wu-chang, Chang Francis, Feng Wu-chi, and Walpole Jonathan, "Provisioning On-line Games: A Traffic Analysis of a Busy Counter-Strike Server," in *SIGCOMM Internet Measurement Workshop*, 2002.

[51] J. Färber, "Network game traffic modelling," in *In Proceedings of the 1st Workshop on Network and System Support For Games*, New York, NY., 2002.

[52] J. Kim et al., "Traffic characteristics of a massively multi-player online role playing game," in *In Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support For Games*, New York, NY., 2005.

[53] C. Diot and L. Gautier, "A distributed architecture for multiplayer interactive applications on the internet," in *IEEE Networks magazine*, 2002, pp. 6-15.

[54] Johannes Färber, "Traffic Modelling for Fast Action Network Games," in *Multimedia Tools Appl.*, Hingham, MA, USA, 2004, pp. 31-46.

Conference on Computer Science, Washington, DC, USA, 2003, p. 168.

[32] Nicholas T. Karonis et al., "High-Resolution Remote Rendering of Large Datasets in a Collaborative Environment," *Future Gener. Comput. Syst.*, pp. 909-917, 2003.

[33] Liu Zhen, Shi Jiaoying, Peng Haoyu, and Xiong Hua, "D3DPR: A Direct3D-Based Large-Scale Display Parallel Rendering System Architecture for Clusters," in *Asia-Pacific Computer Systems Architecture Conference*, Berlin / Heidelberg, 2005, pp. 540-550.

[34] Ping Yin and Jiaoying Shi, "Cluster Based Real-time Rendering System for Large Terrain Dataset," in *CAD-CG '05: Proceedings of the Ninth International Conference on Computer Aided Design and Computer Graphics*, Washington, DC, USA, 2005, pp. 365-370.

[35] Lee Won-Jong et al., "A New Bandwidth Reduction Method for Distributed Rendering Systems," in *EurAsia-ICT '02: Proceedings of the First EurAsian Conference on Information and Communication Technology*, London, UK, 2002, pp. 387-394.

[36] Rudrajit Samanta, Thomas Funkhouser, Li Kai, and Pal Singh Jaswinder, "Hybrid Sort-First and Sort-Last Parallel Rendering with a Cluster of PCs," in *HWWS '00: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, Interlaken, Switzerland, 2000, pp. 97-108.

[37] Carlos González Morcillo, Gerhard Weiss, David Vallejo Fernández, Luis Jiménez Linares, and José A. Fernández Sorribes, "3D Distributed Rendering and Optimization using Free Software," *FLOSS International Conference*, 2007.

[38] Zhu Huabing, Wang Lizhe, Yun Chan Kai, Cai Wentong, and See Simon, "A Distributed Rendering Environment for Massive Data on Computational Grids," in *P2P '03: Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, Washington, DC, USA, 2003, p. 176.

[39] M. Pullen, M. Myjak, and C. Bouwens, *Limitations of Internet Protocol Suite for Distributed Simulation the Large Multicast Environment.* United States: RFC Editor, 2000.

[40] Katherine L. Morse, Lubomir Bic, and Michael Dillencourt, "Interest Management in Large-Scale Distributed Simulation," in *Presence: Teleoper. Virtual Environ.*, Cambridge, MA, USA, 2000, pp. 52-68.

[41] Tsun-Yu Hsiao and Shyan-Ming Yuan, "Practical Middleware for Massively Multiplayer Online Games," in *IEEE Internet Computing*, Piscataway, NJ, USA, 2005, pp. 47-54.

[42] N. Degrande, D. De Vleeschauwer, R. E. Kooij, and M. R. Mandjes, "Modeling ping times in first person shooter games," in *CoNEXT '06: Proceedings of the 2006 ACM CoNEXT*

Communication, Springer-Verlag, 2001, pp. 1-13.

[69] J. Shenker Scott, "Making greed work in networks: a game-theoretic analysis of switch service disciplines," in *IEEE/ACM Trans. Netw. Piscataway*, NJ, USA, 1995, pp. 819-831.

[70] Hans-Jürgen Zepernick, Markus Fiedler Trung Q. Duong, "Cross-Layer Design for Integrated Mobile Multimedia Networks with Strict Priority Traffic," in *2010 IEEE Wireless Communications and Networking Conference, WCNC 2010, Proceedings*, Sydney, Australia, 2010, pp. 18-21.

[71] B. Melamed V. S. Frost, "Traffic modeling for telecommunications networks," in *Communications Magazine, IEEE, Vol. 32*, 1994, pp. 70-81.

[72] Wei Zhang and Jingsha He, "Modeling End-to-End Delay Using Pareto Distribution," in *ICIMP '07: Proceedings of the Second International Conference on Internet Monitoring and Protection*, Washington, DC, USA, 2007, p. 21.

[73] Red Hat. (2011, Octubre) Red Hat. [Online]. http://www.redhat.com/support/resources/howto/rhl72.html

[74] Zebra. (2011, Octubre) Zebra. [Online]. http://www.zebra.org/

[75] CISCO. (2011, Octubre) CISCO. [Online]. http://www.cisco.com/en/US/docs/internetworking/technology/handbook/bgp.html

[76] Eric Lee Helvey, *TRAFGEN: AN EFFICIENT APPROACH TO STATISTICALLY ACCURATE ARTIFICAIL NETWORK TRAFFIC GENERATION*. Ohio : Ohio University, 1998.

[77] Ricardo Cao Abad, *Introducción a la Simulación y a la Teoría de Colas*. Oleiros: NETBIBLO, S.L., 2004.

[78] K. Park and W. Willinger, "Self-similar network traffic: An overview," in *In Self-Similar Network Traffic and Performance Evaluation*, Wiley-Interscience, New York, 2000, pp. 1-38.

[79] Guy Pujolle, "Networking 2000 : broadband communications, high performance networking, and performance of communication networks : IFIP-TC6/European Commission International Conference," in *IFIP-TC6/European Union International Conference*, Paris, 2000, pp. 41-48.

[80] Julio César Ramírez Pacheco and Deni Torres Román, "Performance Analysis of Time-domain Algorithms for Self-similar Traffic," in *16th IEEE International Conference on Electronics, Communications and Computers*, Puebla, México, 2006, p. 28.

[81] IETF. (2011, Octubre) IETF. [Online]. http://tools.ietf.org/html/rfc778

[55] L. Gautier and C. Diot, "Desing and Evaluation of MiMaze a Multi-Player Game on the Internet," in *ICMCS '98: Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, Washington, DC, USA, 1998, p. 233.

[56] R. A. Bangun, E. Dutkiewicz, and G. J. Anido, "An Analysis Of Multi-Player Network Games Traffic," in *Multimedia Signal Processing*, 2001, pp. 3 - 8.

[57] Richard Johnsonbaugh, *Matemáticas discretas.*: Prentice Hall, 2005.

[58] Diego Marcos Jorquera, Virgilio Gilart Iglesias, Francisco José Mora Gimen, and Francisco Maciá Pérez, "Mecanismos de Difusion Masiva en Aplicaciones Distribuidas," in *II jornadas para el desarrollo de grandes aplicaciones de gestión de red (JDARE'05), Servicios electrónicos para la sociedad de la información. Desarrollo de grandes aplicaciones distribuidas sobre internet*, 2005, pp. 205-226.

[59] John C. Lin and Sanjoy Paul, "Reliable Multicast Transport Protocol (RMTP)," in *IEEE Journal on Selected Areas in Communications*, 1997, pp. 1414-1424.

[60] Hans Eriksson, "MBONE: the multicast backbone," in *Communications of the ACM*, 1999, pp. 54-60.

[61] S. Ballesteros, *Memoria Humana: Investigación y teoría.*: Psicothema, 1999.

[62] J. Roca, *Percepcion del movimiento, Revista de Psicología General y Aplicada.*: Univ. Barcelona, Inst. Nal. Educacio Fisica de Catalunya, España., 1995.

[63] M. A. Maiche, *Tiempo de Reacción al Inicio del Movimiento: Un Estudio sobre la Percepción de Velocidad.*: Universidad Autonoma de Balencia, España., 2002.

[64] Humanoid Animation Working Group. H-Anim. [Online]. http://www.h-anim.org/

[65] Len Bass, Paul Clements, and Rick Kazman, *Software Architecture in Practice*, Segunda ed.: Addison Wesley, 2003.

[66] Alma Verónica Martínez González, "Lenguaje para Animación de Criaturas Virtuales," Guadalajara, 2005.

[67] Anindya Basu, Bernadette Charron-Bost, and Sam Toueg, "Simulating reliable links with unreliable links in the presence of process crashes," in *Distributed Algorithms*, Berlin, 1996, pp. 105-122.

[68] Tristan Henderson, "Latency and User Behaviour on a Multiplayer Game Server," in *NGC '01: Proceedings of the Third International COST264 Workshop on Networked Group*

[98] Alexander James Cowie, "The Modelling of Temporal Properties in a Process Algebra Framework," University of South Australia, Australia, Tesis 1999.

[99] Wan Fokkink, *Introduction to Process Algebra.*: Springer-Verlag, 2007.

[100] (2001) CUBE. [Online]. http://cubeengine.com/

[101] Morgan McGuire. G3D Engine. [Online]. http://g3d-cpp.sourceforge.net/

[102] (1985) Gamestudio. [Online]. http://www.3dgamestudio.com/

[103] Steve Streeting. (2000) OGRE. [Online]. http://www.ogre3d.org/

[104] (2000) Torque. [Online]. http://www.garagegames.com/

[105] (2002) 3DVIA Virtools. [Online]. http://www.3ds.com/products/3dvia/3dvia-virtools/

[106] (1999) Havok Physics. [Online]. http://www.havok.com/index.php?page=havok-physics

[107] NVIDIA PhysX SDK. [Online]. http://developer.nvidia.com/object/physx.html

[108] Russell Smith. (2000) ODE. [Online]. http://www.ode.org/ode.html

[109] Stefan Reinalter, "Visibility in a Real-World Cross-Platform Game Engine," Institute of Computer Graphics and Algorithms, Vienna University of Technology, Vienna, Austria, Tesis de maestria 2010.

[110] Hillesland K., Salomon B., Lastra A., and Manocha D., "Fast and simple occlusion culling using hardware-based depth queries.," University of North Carolina at Chapel Hill, North Carolina , Reporte tecnico 202.

[111] Satyan Coorg and Seth Teller, "Real-time occlusion culling for models with large occluders.,' in *Proceedings of the 1997 symposium on Interactive 3D graphics*, New York, NY, USA, 1997, pp. 83-ff.

[112] Ned Greene, Michael Kass, and Gavin Miller, "Hierarchical z-buffer visibility," in *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, New York, NY, USA, 1993, pp. 231-238.

[113] Hansong Zhang, Dinesh Manocha, Tom Hudson, and Kenneth E. Hoff, "Visibility culling using hierarchical occlusion maps," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, New York, NY, USA, 1997, pp. 77-88.

[114] Stefan Popov, Iliyan Georgiev, Rossen Dimov, and Philipp Slusallek, "Object partitioning

[82] IETF. (2011, Octubre) IETF. [Online]. http://tools.ietf.org/html/rfc891

[83] IETF. (2011, Octubre) IETF. [Online]. http://tools.ietf.org/html/rfc956

[84] IETF. (2011, Octubre) IETF. [Online]. http://tools.ietf.org/html/rfc958

[85] IETF. (2011, Octubre) IETF. [Online]. http://tools.ietf.org/html/rfc1305

[86] Epoch & Unix Timestamp Conversion Tools. (2011, Octubre) Epoch & Unix Timestamp Conversion Tools. [Online]. http://www.epochconverter.com/

[87] Jay L. Devore, *PROBABILIDAD Y ESTADÍSTICA PARA INGENIERÍA Y CIENCIAS.*: CENGAGE, 2005.

[88] María José Marques Dos Santos, *Estadistica Basica Un enfoque no parametrico*. México: UNAM, 2001.

[89] Michael J. Evans, *Probabilidad y estadística*. Barcelana, España: Reverte, 2005.

[90] Forouzan Behrouz A., *Introduccion a Las Ciencias de La Computacion.*: Cengage Learning Editores, 2004.

[91] David A. Huffman, *A method for the construction of minimum-redundancy codes*. India: Springer India, in co-publication with Indian Academy of Sciences, 2006.

[92] Ryan Huebsch et al., "The Architecture of PIER: an Internet-Scale Query Processor," in *In CIDR*, 2005, pp. 28-43.

[93] Myung-sup Kim, Hun-jeong Kang, and James W. Hong, "Towards Peer-to-Peer Traffic Analysis Using Flows ," in *In Self-Managing Distributed Systems, 14th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, Heidelberg, Alemania, 2003, pp. 55-67.

[94] CISCO. (2011, Octubre) Routing Information Protocol. [Online]. http://www.cisco.com/en/US/docs/internetworking/technology/handbook/RIP.html

[95] NVidia. (2011, Octubre) Graphics Processing Unit. [Online]. http://www.nvidia.com/object/gpu.html

[96] Wan Fokkink, *Modelling Distributed Systems*. Heidelberg, Berlin: Springer-Verlag, 2007.

[97] MICHAEL ALEXANDER and WILLIAM GARDNER, *PROCESS ALGEBRA FOR PARALLEL AND DISTRIBUTED PROCESSING*, MICHAEL ALEXANDER and WILLIAM GARDNER, Eds. United States of America: Taylor & Francis Group, 2009.

[98] Alexander James Cowie, "The Modelling of Temporal Properties in a Process Algebra Framework," University of South Australia, Australia, Tesis 1999.

[99] Wan Fokkink, *Introduction to Process Algebra.*: Springer-Verlag, 2007.

[100] (2001) CUBE. [Online]. http://cubeengine.com/

[101] Morgan McGuire. G3D Engine. [Online]. http://g3d-cpp.sourceforge.net/

[102] (1985) Gamestudio. [Online]. http://www.3dgamestudio.com/

[103] Steve Streeting. (2000) OGRE. [Online]. http://www.ogre3d.org/

[104] (2000) Torque. [Online]. http://www.garagegames.com/

[105] (2002) 3DVIA Virtools. [Online]. http://www.3ds.com/products/3dvia/3dvia-virtools/

[106] (1999) Havok Physics. [Online]. http://www.havok.com/index.php?page=havok-physics

[107] NVIDIA PhysX SDK. [Online]. http://developer.nvidia.com/object/physx.html

[108] Russell Smith. (2000) ODE. [Online]. http://www.ode.org/ode.html

[109] Stefan Reinalter, "Visibility in a Real-World Cross-Platform Game Engine," Institute of Computer Graphics and Algorithms, Vienna University of Technology, Vienna, Austria, Tesis de maestria 2010.

[110] Hillesland K., Salomon B., Lastra A., and Manocha D., "Fast and simple occlusion culling using hardware-based depth queries.," University of North Carolina at Chapel Hill, North Carolina , Reporte tecnico 202.

[111] Satyan Coorg and Seth Teller, "Real-time occlusion culling for models with large occluders.,' in *Proceedings of the 1997 symposium on Interactive 3D graphics*, New York, NY, USA, 1997, pp. 83-ff.

[112] Ned Greene, Michael Kass, and Gavin Miller, "Hierarchical z-buffer visibility," in *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, New York, NY, USA, 1993, pp. 231-238.

[113] Hansong Zhang, Dinesh Manocha, Tom Hudson, and Kenneth E. Hoff, "Visibility culling using hierarchical occlusion maps," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, New York, NY, USA, 1997, pp. 77-88.

[114] Stefan Popov, Iliyan Georgiev, Rossen Dimov, and Philipp Slusallek, "Object partitioning

considered harmful: space subdivision for bvhs," in *HPG '09: Proceedings of the Conference on High Performance Graphics 2009*, New York, NY, USA, 2009, pp. 15-22.

[115] Martin Stich, Heiko Friedrich, and Andreas Dietrich, "Spatial splits in bounding volume hierarchies," in *HPG '09: Proceedings of the Conference on High Performance Graphics 2009*, New York, NY, USA, 2009, pp. 7-13.

[116] Ingo Wald, "On fast construction of sah-based bounding volume hierarchies," in *RT '07: Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing*, Washington, DC, USA, 2007, pp. 33-40.

[117] Earth Sculptor. (2011, Octubre) Earth Sculptor. [Online]. http://www.earthsculptor.com/

[118] Sandeep Singhal and Michael Zyda, *Networked virtual environments: design and implementation*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999.

CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL I.P.N.
UNIDAD GUADALAJARA

"2011, Año del Turismo en México".

El Jurado designado por la Unidad Guadalajara del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional aprobó la tesis

Distribución de Puertos Visuales para Ambientes Virtuales

del (la) C.

Alma Veronica MARTINEZ GONZALEZ

el día 15 de Diciembre de 2011.

Dr. Luis Ernesto López Mellado
Investigador CINVESTAV 3B
CINVESTAV Unidad Guadalajara

Dr. Federico Sandoval Ibarra
Investigador CINVESTAV 3B
CINVESTAV Unidad Guadalajara

Dr. Mario Angel Siller González Pico
Investigador CINVESTAV 2C
CINVESTAV Unidad Guadalajara

Dr. Marco Antonio Ramos Corchado
Profesor Investigador Nivel F
Universidad Autónoma del Estado de México

Dr. Héctor Alejandro Durán Limón
Profesor-Investigador
Universidad de Guadalajara