



**UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA**

GRADO EN INGENIERÍA INFORMÁTICA

**TECNOLOGÍA ESPECÍFICA DE
COMPUTACIÓN**

TRABAJO FIN DE GRADO

**AsgAR: sistema generador de representaciones
gráficas para la visualización de programas y
algoritmos mediante realidad mixta**

Cristian Gómez Portes

Julio, 2018

**ASGAR: SISTEMA GENERADOR DE REPRESENTACIONES GRÁFICAS
PARA LA VISUALIZACIÓN DE PROGRAMAS Y ALGORITMOS MEDIANTE
REALIDAD MIXTA**



**UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA**

Tecnologías y Sistemas de Información

**TECNOLOGÍA ESPECÍFICA DE
COMPUTACIÓN**

TRABAJO FIN DE GRADO

**AsgAR: sistema generador de representaciones
gráficas para la visualización de programas y
algoritmos mediante realidad mixta**

Autor: Cristian Gómez Portes

Director: D. Santiago Sánchez Sobrino

Director: Dr. Carlos González Morcillo

Julio, 2018

Cristian Gómez Portes

Ciudad Real – Spain

E-mail universidad: Cristian.Gomez2@alu.uclm.es

E-mail personal: Cristiancgph@gmail.com

© 2018 Cristian Gómez Portes

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Se permite la copia, distribución y/o modificación de este documento bajo los términos de la Licencia de Documentación Libre GNU, versión 1.3 o cualquier versión posterior publicada por la *Free Software Foundation*; sin secciones invariantes. Una copia de esta licencia está incluida en el apéndice titulado «GNU Free Documentation License».

Muchos de los nombres usados por las compañías para diferenciar sus productos y servicios son reclamados como marcas registradas. Allí donde estos nombres aparezcan en este documento, y cuando el autor haya sido informado de esas marcas registradas, los nombres estarán escritos en mayúsculas o como nombres propios.

TRIBUNAL:

Presidente:

Vocal:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE

VOCAL

SECRETARIO

Fdo.:

Fdo.:

Fdo.:

Resumen

La representación gráfica de programas y algoritmos es un campo de investigación con publicaciones relevantes de más de 25 años. Desde sus inicios, se ha demostrado que su utilización supone una mejora significativa en el proceso de aprendizaje por parte de alumnos de diferentes niveles académicos. Sin embargo, este tipo de aproximaciones requiere un importante esfuerzo por parte del personal docente para su correcta integración.

Por otro lado, la ausencia de dispositivos que permiten visualizar dichas representaciones gráficas de manera natural dificulta su utilización. No obstante, en los últimos años el número de dispositivos inmersivos que permiten visualizarlas ha aumentado considerablemente debido a los recientes avances tecnológicos.

El presente Trabajo de Fin de Grado surge como parte del proyecto de investigación IA-PRO, cuyo objetivo es el desarrollo de sistemas inmersivos para el aprendizaje de la programación. En este proyecto se plantea el desarrollo de una herramienta colaborativa llamada COLLECE 2.0, que permite a los estudiantes aprender conceptos de programación mediante la resolución de problemas de forma cooperativa.

En este ámbito, AsgAR se constituye como un sistema que genera representaciones gráficas basadas en una metáfora de carreteras y señales de tráfico para visualizar programas y algoritmos. Este sistema se integrará como un plug-in de COLLECE 2.0, desplegándose además como una plataforma completa que permita visualizar cualquier tipo de programa mediante el dispositivo Microsoft HoloLens.

Abstract

The graphical representation of programs and algorithms is a field of research with relevant publications of more than 25 years. Since its inception, its use has been shown to improve significantly the learning process for students of different academic levels. However, this type of approaches requires a major effort on the part of professors to be them correctly integrated.

On the other hand, the absence of devices that allow these graphical representations to be displayed in a natural way makes their use difficult. Nevertheless, in recent years the number of immersive devices for displaying them has increased considerably due to recent technological advances.

This Final Degree Project arises as part of the IAPRO research project, whose objective is the development of immersive systems for programming learning. In this project, the development of a collaborative tool called COLLECE 2.0 is proposed, which allows students to learn programming concepts through collaborative problem solving.

In this context, AsgAR is constituted as a system that generates graphical representations based on a metaphor of roads and traffic signs to visualize programs and algorithms. This system will be integrated as a COLLECE 2.0 plug-in, and will also be deployed as a complete platform that allows the visualization of any type of program using the Microsoft HoloLens device.

Agradecimientos

Este Trabajo Fin de Grado no es sólo fruto del esfuerzo, tiempo y dedicación empleado por el autor del mismo, sino de muchas otras personas que, interesada o desinteresadamente han contribuido a lograr el resultado final plasmado en este documento. Con estas líneas quisiera agradecer el apoyo, ayuda y comprensión a todas ellas.

En primer lugar, quiero expresar mi agradecimiento a los directores de este trabajo, Santiago Sánchez Sobrino y Carlos González Morcillo. A Santiago, por su apoyo incondicional y ayuda constante, y a Carlos, por depositar su confianza en mí e introducirme en este apasionante mundo de la Realidad Aumentada. Ambos grandes profesionales a los que admiro.

A Miguel Ángel Redondo Duque, por brindarme la oportunidad de formar parte del proyecto IAPRO desarrollado en el grupo de investigación CHICO.

A David, por sus sabios consejos y ayuda desinteresada, y a Nines, por su apoyo e infinita paciencia.

A mis padres, por la confianza, ayuda y sacrificio durante estos años de carrera, los cuales me han servido para llegar a donde estoy hoy.

A Alba, por su comprensión, paciencia y apoyo desmesurado.

Cristian

*A mis padres,
mi hermana,
mi novia,
y mi tía.*

Índice general

Resumen	V
Abstract	VII
Agradecimientos	IX
Índice general	XIII
Índice de cuadros	XVII
Índice de figuras	XIX
Índice de listados	XXI
Listado de acrónimos	XXIII
1. Introducción	1
1.1. Realidad Mixta	2
1.2. Contexto	4
1.3. Entorno	4
1.4. Impacto socio-económico	5
1.5. Problemática	6
1.6. Estructura del documento	6
2. Objetivos	9
2.1. Objetivo general	9
2.2. Objetivos específicos	9
2.2.1. Entorno integrado con COLLECE 2.0	10
2.2.2. Entorno para la visualización de programas	10
2.2.3. Desarrollo de la interfaz de usuario	11
2.2.4. Arquitectura escalable	11
2.2.5. Recuperación ante errores	11

0. ÍNDICE GENERAL

3. Objectives	13
3.1. General objective	13
3.2. Specific objectives	13
3.2.1. Environment integrated with COLLECE 2.0	14
3.2.2. Environment for visualisation of programs	14
3.2.3. Development of user interface	15
3.2.4. Scalable architecture	15
3.2.5. Error recovery	15
4. Estado del arte	17
4.1. Sistemas para la visualización de programas y algoritmos	17
4.2. Entornos de desarrollo extensibles	22
4.3. Dispositivos de Realidad Aumentada y algoritmos de tracking	25
4.4. Metáforas utilizadas durante el aprendizaje de la programación	29
4.5. Tecnologías de red	30
4.6. Conclusión	31
5. Método de trabajo	33
5.1. Metodología de trabajo	33
5.2. Desarrollo iterativo e incremental	33
5.2.1. Desarrollo incremental	34
5.2.2. Desarrollo iterativo	35
5.3. Planificación	35
5.4. Medios empleados	37
5.4.1. Medios hardware	38
5.4.2. Medios software	38
6. Arquitectura	43
6.1. Servidor	45
6.1.1. Sistema Generador de Códigos QR	46
6.1.2. Sistema de Procesamiento	47
6.1.3. Sistema de Eventos	56
6.1.4. Sistema de Streaming	59
6.1.5. Sistema de Comunicación	63
6.1.6. Patrones de diseño	65
6.2. Cliente	66

6.2.1. Sistema de Lectura de Códigos QR	67
6.2.2. Sistema de Recepción de archivos JSON	69
6.2.3. Sistema de Captura	71
6.2.4. Sistema Generador de Representaciones Gráficas	74
6.2.5. Otras características	79
6.2.6. Patrones de diseño	79
7. Resultados	81
7.1. Aspecto y funcionalidad	81
7.2. Estadística del proyecto	86
7.3. Costes	87
7.4. Medidas de rendimiento	87
8. Conclusiones	91
8.1. Conclusiones	91
8.2. Líneas de trabajo futuras	92
8.3. Conclusión personal	93
9. Conclusions	95
9.1. Conclusions	95
9.2. Future lines of work	96
9.3. Personal conclusion	97
A. Características Microsoft HoloLens	101
B. Contenido del CD	103
C. Código Fuente	105
C.1. Cliente	105
C.2. Servidor	105
D. Instalación	107
D.1. Instalación del cliente	107
D.2. Instalación del servidor	108
E. Manual de usuario	109
E.1. Servidor	109
E.2. Cliente	110

0. ÍNDICE GENERAL

F. Diagramas UML	113
F.1. Servidor	113
F.2. Cliente	114
G. Autoría de imágenes	115
H. GNU Free Documentation License	117
H.0. PREAMBLE	117
H.1. APPLICABILITY AND DEFINITIONS	117
H.2. VERBATIM COPYING	118
H.3. COPYING IN QUANTITY	118
H.4. MODIFICATIONS	119
H.5. COLLECTIONS OF DOCUMENTS	120
H.6. AGGREGATION WITH INDEPENDENT WORKS	120
H.7. TRANSLATION	120
H.8. TERMINATION	120
H.9. FUTURE REVISIONS OF THIS LICENSE	121
H.10. RELICENSING	121
Referencias	123

Índice de cuadros

7.1. Estadísticas del código del proyecto entre cliente y servidor	86
7.2. Desglose de costes de AsgAR	87
A.1. Características del dispositivo de RA Microsoft HoloLens [hol18]	101

Índice de figuras

1.1.	Representación del CV	2
1.2.	Diagrama de Venn que muestra la relación entre los 3 modos de interacción	3
1.3.	Diagrama que muestra brevemente el flujo del sistema	4
4.1.	Captura del pantalla del sistema SRec en una sesión	18
4.2.	Torres de Hanoi en progreso utilizando Alice tool	19
4.3.	Resolución del problema mediante animación textual	19
4.4.	Representación de algoritmo en Jeliot 2000	20
4.5.	(a) Taza virtual sobre la marca (b) Taza trasladada (c) Renderizado utilizando la cámara virtual ubicada en la marca	21
4.6.	Versión mejorada del sistema de aprendizaje de RA	22
4.7.	Captura de pantalla del editor de NetBeans	23
4.8.	Captura de pantalla del editor de IntelliJ IDEA	24
4.9.	Captura de pantalla del editor de Eclipse	24
4.10.	Brazo mecánico usado para medir la orientación de la cabeza del usuario . .	25
4.11.	(a) Objeto tridimensional superpuesto en la pantalla de un smartphone y (b) un computador	26
4.12.	(a) Dispositivos de RA de Meta (b) Epson y (c) Atheer	27
4.13.	Microsoft HoloLens	28
5.1.	Modelo de desarrollo iterativo e incremental	34
5.2.	Tiempo empleado en cada tarea para el desarrollo del servidor	36
5.3.	Tiempo empleado en cada tarea para el desarrollo del cliente	37
6.1.	Esquema de la plataforma AsgAR	44
6.2.	Elementos que componen el Sistema de Procesamiento	48
6.3.	Métodos públicos de la clase <i>Parser</i>	50
6.4.	Diagrama de la clase <i>RootObject</i> y <i>Visualization</i>	53
6.5.	Elementos que componen el Sistema de Eventos	56
6.6.	Elementos que componen el Sistema de Streaming	60

0. ÍNDICE DE FIGURAS

6.7. Estructura de la tubería para el cambio de estados	61
6.8. Elementos que componen el Sistema de Comunicación	63
6.9. Elementos que componen el Sistema de Recepción de archivos JSON	70
6.10. Elementos que componen el Sistema de Captura	72
6.11. Máquina de estados para procesar los fotogramas	73
6.12. Elementos que componen el Sistema Generador de Representaciones Gráficas	74
6.13. Ejemplo que refleja las marcas de la representación 3D de la sentencia bucle: <i>joiner_in</i> (verde), <i>joiner_out</i> (azul), <i>joiner_fill</i> (rojo)	77
6.14. Ejemplo que refleja las marcas para el papel de la representación 3D de la sentencia bucle y expresión: <i>paper_in</i> (verde) y <i>paper_out</i> (rojo)	78
7.1. Sentencia bucle	81
7.2. Sentencia <i>if ... then ... else</i>	82
7.3. Definición de función y sentencia de retorno	82
7.4. Diseño de la vista del plug-in de Eclipse	83
7.5. Grid que representa mediante imágenes el aspecto final del sistema	85
7.6. Diagrama que muestra las contribuciones de la aplicación del cliente	86
7.7. Diagrama que muestra las contribuciones de la aplicación del servidor	86
7.8. Tiempo de procesamiento del código fuente en función del nº de líneas de código	88
7.9. Tiempo de envío y recepción de paquetes en función del tamaño del archivo	89
7.10. Tiempo de generación de representaciones gráficas en función del nº de lí- neas de código	90
7.11. Tiempo en generar y pintar un fotograma	90
D.1. Windows Device Portal	108
E.1. (a) comienzo y (b) fin del gesto <i>Bloom</i>	110
E.2. (a) comienzo y (b) fin del gesto <i>Air Tap</i>	110
F.1. Diagrama UML del plug-in de Eclipse	113
F.2. Diagrama UML de la aplicación del cliente	114

Índice de listados

6.1.	Generación código QR	47
6.2.	Procesamiento del código fuente de Java	49
6.3.	Agregación de los métodos visitados en la lista <code>methodDeclaration</code>	49
6.4.	Ejemplo de nodo y conector en formato JSON	51
6.5.	Función que obtiene la cadean JSON	52
6.6.	Método que completa las ramas de la estructura de bucle y condición	54
6.7.	Función que completa la estructura de bucle y condición	55
6.8.	Eventos de la vista <i>MainViewPart</i>	57
6.9.	Registro de eventos en <i>MainViewPartController</i>	57
6.10.	Implementación de interfaz y clases	58
6.11.	Estados de la clase <i>HandlerState</i>	61
6.12.	Función que procesa y retransmite fotogramas a los miembros conectados al servidor	62
6.13.	Envío de mensaje con la especificación del proceso a realizar	64
6.14.	Envío de mensaje con la especificación del proceso a realizar	64
6.15.	Método que procesa los fotogramas de la cámara del dispositivo de RA	67
6.16.	Método que recupera el valor de un código QR	68
6.17.	Lectura asíncrona de mensajes	70
6.18.	Estados que especifican la situación en la que se encuentra un fotograma	72
6.19.	Método recursivo para generar las representaciones gráficas	75
6.20.	Método que gestiona la creación de las representaciones gráficas	77
7.1.	Algoritmo de la burbuja	88

Listado de acrónimos

AST	Abstract Syntax Tree
API	Application Programming Interface
APIs	Application Programming Interfaces
CV	Continuidad Virtual
CLR	Common Language Runtime
ECF	Eclipse Communication Framework
ESI	Escuela Superior de Informática
FDP	Final Degree Project
HMD	Head-Mounted Display
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IDEs	Integrated Development Environments
IU	Interfaz de Usuario
IMU	Inertial Measurement Unit
IPO	Interacción Persona-Ordenador
IP	Internet Protocol
JVM	Java Virtual Machine
JDT	Java Development Tools
JSON	JavaScript Object Notation
MRTK	MixedRealityToolkit
OSGI	Open Services Gateway initiative
PDE	Plug-in Development Environment
POO	Programación Orientado a Objetos
RA	Realidad Aumentada
RM	Realidad Mixta
RV	Realidad Virtual

0. LISTADO DE ACRÓNIMOS

RPC	Remote Procedure Call
SWT	Standard Widget Toolkit
TCP	Transmission Control Protocol
TFG	Trabajo Fin de Grado
TIC	Tecnologías de la Información y la Comunicación
ToF	Time-of-Flight
TTS	Text To Speech
UCLM	Universidad de Castilla-La Mancha
UDP	User Datagram Protocol
VA	Virtualidad Aumentada
WMR	Windows Mixed Reality
WWW	World Wide Web

Capítulo 1

Introducción

LA Informática Gráfica es la rama de la Informática que se ocupa de la creación de representaciones gráficas mediante un ordenador. Con sus orígenes en los años 60, con sistemas basados en representaciones analógicas (como el Sketchpad de Sutherland [Sut63]), los sistemas han ido evolucionando en diferentes áreas de conocimiento. Una de las áreas que más impacto económico tiene es la relacionada con la Interacción Persona-Ordenador (IPO) y los métodos de Visualización de Información, que permiten utilizar las capacidades del ordenador de un modo cada vez más natural e intuitivo. Gracias al conjunto de técnicas de Visualización de Información, cada día somos capaces de resolver problemas más complejos y comprender mayores volúmenes de datos empleando ordenadores.

Uno de los campos que han surgido gracias a los avances en la Informática Gráfica ha sido la Visualización de Software, término que se refiere a la representación del comportamiento de sistemas software. Este término surgió alrededor de 1980 con el propósito de comprender y representar gráficamente conceptos informáticos abstractos de difícil comprensión [Bro88].

Del concepto de Visualización de Software, se establece una taxonomía que puede encontrarse en [PBS93]. Esta clasificación se divide en: Visualización de Programas y Visualización de algoritmos, las cuales hacen uso de técnicas para incrementar y mejorar la experiencia de aprendizaje de los estudiantes.

La visualización de programas y algoritmos puede dividirse a su vez en dos tipos de representaciones: estáticas y dinámicas. La representación de aspectos estáticos aporta información válida sobre todas las posibles ejecuciones del software, mientras que la representación de aspectos dinámicos proporciona información sobre una ejecución particular del mismo [CZ11].

Estas visualizaciones proporcionan muchas ventajas en el ámbito de la educación debido a aportes pedagógicos como la creatividad y la motivación [HDS02]. No obstante, la mayoría de las representaciones gráficas existentes no son del todo explicativas, puesto que requieren de una mayor capacidad cognitiva que los estudiantes de programación de primeros cursos todavía no poseen.

1. INTRODUCCIÓN

Por este motivo, en el presente Trabajo Fin de Grado (TFG) se diseña y desarrolla un sistema que construye visualizaciones de programas a partir de un conjunto de representaciones gráficas basadas en una metáfora de carreteras y señales de tráfico. Para ello se utiliza el dispositivo Microsoft HoloLens, el cual integra tecnología inmersiva que soporta técnicas de Realidad Mixta (RM) para ofrecer una experiencia de usuario natural y conducir a los estudiantes a un mejor rendimiento de aprendizaje [DSEB15]. De este modo, este sistema pretende resolver la problemática que los docentes manifiestan respecto a la creación de representaciones gráficas, generándolas dinámicamente, y ubicándolas y alineándolas en el espacio físico real.

1.1 Realidad Mixta

La RM se refiere a la mezcla de espacios tanto físicos como virtuales en un entorno donde los objetos reales coexisten con los virtuales. Según Paul Milgram y Fumio Kishino en [MK94], la RM no sólo toma lugar en el mundo físico o real, sino que es una mezcla de realidad real y Realidad Virtual (RV), cubriendo la mayoría del espectro de la Continuidad Virtual (CV), desde la Realidad Aumentada (RA) hasta la Virtualidad Aumentada (VA).



Figura 1.1: Representación del CV

Este concepto ha sido un tema de investigación a lo largo de los años, el cual ha encontrado su aplicación en un gran número de áreas, ya que involucra visión por computador, gráficos por computador, interfaces de usuario naturales y visualización de información, entre otros muchos campos.

El primer sistema inmersivo de RM, que proporcionaba visión, sonido y tacto, fue desarrollado en los laboratorios Armstrong de la Fuerza Aérea de los EE.UU. en la década de 1990 [Ros93]. Esta plataforma fue construida para mejorar el rendimiento humano en tareas de combate aéreo que requerían manipulaciones remotas y directas. Tras las ventajas que esta herramienta supuso, la RM comenzó a aplicarse a un gran conjunto de áreas.

Uno de los primeros trabajos se realizó en 1993, en relación a la reparación y mantenimiento de maquinaria. Un ejemplo de esto se muestra en [FMS93], donde se desarrolló un sistema para guiar al usuario en operaciones básicas de mantenimiento de impresoras.

Otro ejemplo, en este caso relacionado con el área de la fabricación de coches, fue propuesto en 1998 [RSKM98], el cual se ayudaba mediante RM para ensamblar en la posición correcta las puertas de los coches a fabricar.

A comienzos de 2001, este concepto comenzó a aplicarse al desarrollo de interfaces de usuario naturales. Un ejemplo de esto se presenta en [BKP01], donde se desarrolló un sistema para visualizar modelos tridimensionales sobre las páginas de un libro físico.

En cuanto al área de la medicina, se han propuesto varios trabajos para visualizar, entre otra cosas, imágenes médicas que guíen las acciones de los cirujanos. En 1996, en [LGH⁺96], se implementó un sistema basado en pantallas transparentes colocadas en la cabeza (Head-Mounted Display (HMD)) para guiar al doctor a realizar incisiones con una aguja.

Estas aplicaciones destacan la infinidad de utilidades que tiene la RM sobre, prácticamente, cualquier área o industria, ya que proporciona capas de información visual a la vista que facilitan al cerebro su gestión de una forma más rápida.

Desde entonces, la aplicación de la RM ha evolucionado notablemente, dando lugar a nuevos modos de interacción (ver Figura 1.2) entre el ser humano, el computador y el entorno físico [mix18]. Esta evolución se debe a grandes avances en los campos de la visión por computador, la cual permite el reconocimiento de objetos; las tecnologías de visualización, las cuales a través de pantallas permiten mostrar modelos tridimensionales; y los sistemas de entrada, los cuales mediante gestos habilitan una interacción más natural con el sistema.

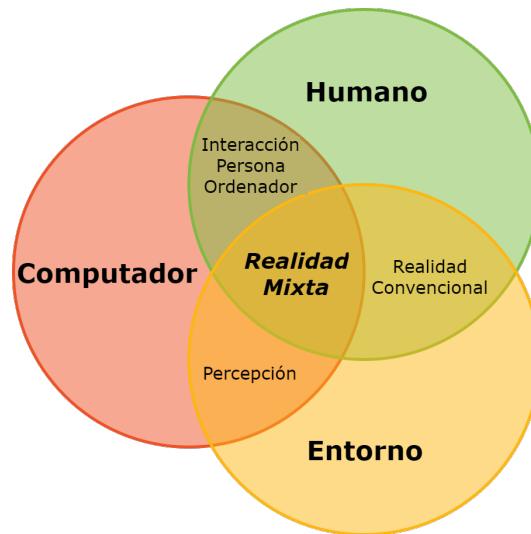


Figura 1.2: Diagrama de Venn que muestra la relación entre los 3 modos de interacción

Estos modos de interacción nacen de la definición que Microsoft proporciona de la RM, ya que con este término se refiere a las dos realidades que definen los extremos de la CV. Según Microsoft, el extremo izquierdo del espectro se refiere a la superposición de modelos tridimensionales en el mundo físico (RA), mientras que el extremo derecho se refiere a las experiencias que proporcionan contenido totalmente digital (RV). Dado que el proyecto expuesto en este trabajo se centra en el extremo izquierdo del espectro, se utilizará el término de RA a lo largo del documento para nombrar al dispositivo mencionado al comienzo de la introducción.

1.2 Contexto

El presente trabajo trata el problema de la enseñanza de la programación en niveles universitarios mediante tecnologías inmersivas y de visualización, las cuales pretenden ayudar a los estudiantes a incrementar su motivación, atención y concentración, entre otros muchos beneficios.

Este sistema nace como una parte de un proyecto de investigación, **iProg**, que propone diseñar y desarrollar nuevos niveles educativos dirigidos a estudiantes, entornos de aprendizaje activos, y sistemas software avanzados, entre otros.

Este proyecto de investigación se desarrolla en el contexto de **IAPRO**¹, llevado a cabo por los grupos de investigación CHICO y AIR de la Universidad de Castilla-La Mancha, cuyo objetivo es el desarrollo de sistemas inmersivos para el aprendizaje de la programación. En este proyecto se plantea el desarrollo de la herramienta colaborativa COLLECE 2.0², que provee de un conjunto de características que facilitan el aprendizaje de la programación mediante la participación, y colaboración remota y síncrona de usuarios ante un problema propuesto.

1.3 Entorno

El entorno propuesto se apoya en una arquitectura de red cliente-servidor (ver Figura 1.3). El servidor se desarrolla como un plug-in para Eclipse integrado en COLLECE 2.0, el cual es el encargado de realizar de forma automática la elaboración de representaciones gráficas. Por otro lado, el cliente se compone del dispositivo de RA Microsoft HoloLens, que proporciona un mecanismo de visualización de las representaciones gráficas alineadas con el espacio físico.

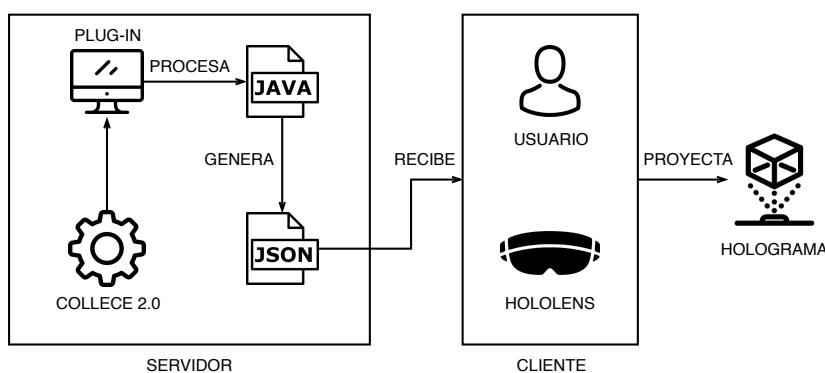


Figura 1.3: Diagrama que muestra brevemente el flujo del sistema

Para construir la visualización automáticamente a partir de las representaciones gráficas, el servidor procesa el código fuente del método que el usuario quiere visualizar y lo transforma en un formato de intercambio de datos que es enviado al cliente. Por otra parte, el cliente

¹<http://blog.uclm.es/grupochico/proyecto-iapro/>

²<http://blog.uclm.es/grupochico/proyecto-iapro/collece-2-0/>

escanea el entorno, recibe el contenido y construye la representación para ser visualizada por el usuario.

1.4 Impacto socio-económico

El avance de la tecnología está intensificando la introducción de dispositivos que mejoran la visualización de modelos tridimensionales. Esto viene ligado a los avances relacionados con tecnologías emergentes, que proporcionan experiencias y que aportan un abanico de mejoras en el ámbito de la educación, tales como la motivación, atención y concentración [CB16].

La introducción de planes curriculares relacionados con la enseñanza de la programación puede mejorar habilidades como el pensamiento crítico, la resolución de problemas y la creatividad, entre otras [LPPV17]. Con esto se espera estar preparados para un mercado laboral que cada vez demanda más profesionales en el área de las Tecnologías de la Información y la Comunicación (TIC), ya que «la Comisión Europea calcula que en el año 2020 existirán alrededor de 900.000 puestos vacantes en el ámbito de las TIC que necesitarán ser cubiertos en Europa»³.

En esta línea, Niel Kroes, vicepresidente de la Comisión Europea y responsable de la Agenda Digital para Europa, envió el 25 de julio del 2014 una carta conjunta a los Ministros de Educación de la Unión Europea⁴ animándoles a promover la enseñanza de la programación informática, ya que «la programación es parte de la solución al desempleo juvenil en Europa».

En este sentido, el presente trabajo está enmarcado en un entorno educativo de un fuerte impacto social, el cual está vinculado con la inserción de asignaturas relacionadas con la programación informática. En estos momentos, uno de los impedimentos de incluir este tipo de asignaturas se debe a la dificultad que los docentes sin formación específica en TIC encuentran a la hora de explicar conceptos de difícil comprensión. Sin embargo, este problema pretende reducirse mediante la utilización de representaciones familiares y de uso común que disminuyan el nivel de abstracción de los estudiantes. De este modo, este sistema se presenta como una solución a este problema, aportando una representación natural alineada en el mundo físico real.

En relación con lo anterior, este sistema además se expone como una herramienta para incrementar el rendimiento de aprendizaje de los estudiantes, lo que supone facilitar el aprendizaje de la programación. Por lo tanto, en términos económicos esto espera formar a nuevos profesionales para cubrir la demanda actual de programadores.

En definitiva, el proyecto plantea algunas ayudas a diferentes tipos de necesidades en el

³<https://euobserver.com/digital/137835>

⁴http://ec.europa.eu/information_society/newsroom/cf/dae/document.cfm?doc_id=6597

1. INTRODUCCIÓN

ámbito educativo. De esta forma, el sistema se configura como una solución que proporciona una experiencia de aprendizaje innovadora.

1.5 Problemática

La generación de representaciones gráficas ha supuesto un problema para el personal docente debido al esfuerzo y tiempo que se requiere para su diseño y elaboración. Además, éstas no suelen ser autoexplicativas, lo que dificulta su comprensión, y por consiguiente, la pérdida de atención de los estudiantes en la enseñanza de la programación.

Teniendo en cuenta las dificultades anteriormente señaladas, el presente TFG se ha centrado en tres áreas de actuación, basadas en los beneficios descritos en [DSEB15]:

- **Generación automática de representaciones gráficas:** resulta interesante destacar la creación de elementos gráficos para explicar conceptos de la programación que no son triviales durante los niveles iniciales del proceso de aprendizaje. Sin embargo, la extensión de su uso en las aulas no se ha llevado a cabo debido a la dedicación que requiere. Por lo tanto, en este sentido, un entorno con la capacidad de automatizar la construcción de representaciones ha sido creado con el objetivo de ayudar a los docentes a utilizar técnicas de visualización de programas y algoritmos en las aulas.
- **Conjunto de modelos gráficos:** dado que la mayoría de los modelos gráficos no son del todo naturales, se han adaptado un conjunto de representaciones gráficas ya existentes, derivadas de la metáfora de carreteras y señales de tráfico del proyecto de investigación, a un entorno tridimensional mediante un diseño modular que permite la representación de programas complejos. Este diseño trata que los estudiantes mejoren su comprensión debido a la familiarización que éstas presentan en su vida cotidiana.
- **Tecnologías inmersivas:** mantener la atención y aumentar el interés del estudiante durante el proceso de aprendizaje es otra de las dificultades a las que se enfrentan los docentes. Hoy en día, las aulas cuentan con un gran número de alumnos, lo que dificulta la estimulación del interés y la participación de todo ellos. Sin embargo, este problema puede solventarse mediante la integración de tecnologías inmersivas. Este proyecto incluye el uso de RA, la cual aumenta la receptividad del alumno, prolonga su interés, y estimula su participación, entre otras muchas ventajas, dirigidas a mejorar el rendimiento del aprendizaje.

1.6 Estructura del documento

El presente documento se ha estructurado según las indicaciones de la normativa de trabajos fin de grado de la Escuela Superior de Informática (ESI) de la Universidad de Castilla-La Mancha (UCLM) empleando los siguientes capítulos:

Capítulo 2: Objetivos

En este capítulo se desglosan y describen los objetivos y subobjetivos planteados que se han seguido para el desarrollo del presente trabajo.

Capítulo 4: Estado del arte

Este capítulo realiza una revisión en torno a los temas que trata este trabajo, recopilando las tecnologías y conceptos teóricos estudiados para el diseño e implementación del sistema.

Capítulo 5: Método de trabajo

Este capítulo explica los diferentes medios utilizados, además de la metodología de desarrollo empleada.

Capítulo 6: Arquitectura

Este capítulo describe cómo se ha diseñado e implementado el sistema, detallando los problemas que han surgido durante el desarrollo y las soluciones propuestas aplicadas a los mismos.

Capítulo 7: Resultados

Este capítulo refleja el resultado final obtenido tras lograr los objetivos propuestos al comienzo del proyecto.

Capítulo 8: Conclusiones

Este capítulo resume los resultados más importantes logrados a lo largo de las fases de diseño, implementación y pruebas, junto a posibles mejoras o evoluciones que pueden nacer a partir de este trabajo.

Capítulo 2

Objetivos

DADO el enfoque y las ideas presentadas en el capítulo de introducción (ver § 1), este apartado se centra en detallar de manera exhaustiva lo que se pretende llevar a cabo con este trabajo, sintetizando el objetivo general del proyecto y desglosando los objetivos específicos derivados.

2.1 Objetivo general

El objetivo principal del presente TFG es desarrollar un sistema software cuya finalidad sea la de **generar representaciones gráficas mediante metáforas o abstracciones basadas en una notación de señales de tráfico y carreteras para la visualización de programas y algoritmos mediante RA**. De esta forma, el cumplimiento de dicho objetivo repercutirá directamente en los resultados obtenidos en el contexto global del proyecto de investigación, a la hora de mejorar el aprendizaje de la programación.

Para ello, se propone un sistema basado en técnicas avanzadas de interacción y visualización mediante RA que ofrezca una experiencia de interacción y visualización natural.

Dicho sistema utiliza el espacio físico para ubicar y alinear las representaciones gráficas en tiempo real con el fin de mostrar la estructura estática de los programas o algoritmos. Estas representaciones son generadas automáticamente por el sistema, aprovechando su integración en COLLECE 2.0 para hacer uso de las ventajas y facilidades que proporciona.

2.2 Objetivos específicos

A partir del objetivo principal descrito en la sección anterior se presentan a continuación los subobjetivos específicos que se pretenden abordar. Éstos pueden agruparse en torno a las siguientes líneas: creación de entorno integrado en COLLECE 2.0 para la autogeneración de representaciones gráficas, elaboración de un sistema encargado de la representación de programas o algoritmos, desarrollo de la Interfaz de Usuario (IU) y propuesta de arquitectura escalable. Además, a todo esto, se le añade la estabilidad y recuperación ante errores que presenta el sistema.

2. OBJETIVOS

2.2.1 Entorno integrado con COLLECE 2.0

COLLECE 2.0 es un plug-in para la plataforma Eclipse que proporciona un conjunto de características colaborativas para ayudar a los estudiantes en el aprendizaje de la programación. Este trabajo es la base del entorno discutido en esta sección, planteado como un nuevo plug-in, para aprovechar las diversas funcionalidades de colaboración que ofrece. A continuación, se describen las tareas que se han llevado a cabo para la elaboración del plug-in propuesto:

- Creación de un plug-in para la herramienta Eclipse que sirva como base para la representación de componentes gráficos.
- Integración del plug-in anterior en COLLECE 2.0 para obtener acceso a la infraestructura de comunicaciones disponibles.
- Análisis de código fuente de un programa seleccionado por el usuario con el fin de extraer la información necesaria para su posterior representación.
- Generación de archivo JSON a partir del código fuente analizado en el punto anterior.
- Creación de servidor para establecer una comunicación con el dispositivo de RA.
- Generación automática de códigos QR con la información de conexión de red necesaria para establecer una comunicación con el servidor.
- Retransmisión en primera persona del contenido visualizado por el dispositivo de RA.
- Envío de archivos JSON al dispositivo de RA.
- Elaboración de sistema de eventos para facilitar la gestión de sucesos que ocurrán a lo largo de la ejecución del sistema.

2.2.2 Entorno para la visualización de programas

En este subobjetivo se enumeran las tareas que se han llevado a cabo para construir el sistema capaz de visualizar, a través de componentes gráficos, el código fuente de un programa.

- Creación de un cliente para establecer una comunicación con el servidor.
- Lectura de códigos QR mediante la cámara frontal del dispositivo de RA.
- Recepción de archivos JSON desde el servidor para visualizar las representaciones gráficas.
- Procesamiento del archivo JSON para la construcción de la visualización correspondiente a la representación estática del programa analizado.
- Desarrollo de un conjunto de mecanismos de interacción que faciliten el uso del dispositivo.

- Detección de las diferentes superficies que componen el entorno físico para ubicar los componentes gráficos que representan el programa o algoritmo a visualizar.
- Ajuste de tamaño, orientación y posición de las representaciones gráficas teniendo en cuenta las superficies sobre las que son colocadas.

2.2.3 Desarrollo de la interfaz de usuario

En este subobjetivo se enumeran las tareas que se han llevado a cabo para elaborar los componentes visuales que conforman la IU del sistema.

- Diseño y desarrollo de una vista para el plug-in integrado en COLLECE 2.0, en la cual se muestra un código QR con la información de conexión, la representación en JSON del programa o algoritmo a visualizar y la retransmisión en vivo de lo que el usuario visualiza a través del dispositivo de RA.
- Diseño y adaptación de la metáfora de carreteras y señales de tráfico que representen tanto estructuras de control como expresiones del código fuente de un programa.
- Diseño y desarrollo de componentes visuales (widgets) tales como menús, botones, barras de progreso e iconos que faciliten la interacción con el dispositivo de RA.

2.2.4 Arquitectura escalable

Este trabajo se ha desarrollado pensando en futuras ampliaciones que permitan mejorar el sistema. A continuación, se muestran las tareas que han permitido llevar a cabo este subobjetivo.

- Soporte para el análisis de diversos lenguajes de programación (en esta versión se analizan únicamente programas desarrollados en Java).
- Aislamiento de componentes para evitar que éstos tengan dependencias con otros.
- Elaboración de sistema de escenas que permita añadir nuevas funcionalidades.
- Diseño modular de representaciones gráficas que permita reemplazar la notación basada en la metáfora de carreteras y señales de tráfico por otra.
- Uso adecuado de patrones de diseño y otras prácticas de programación que proporcionen mecanismos para construir un sistema mantenable.

2.2.5 Recuperación ante errores

El sistema está compuesto por un conjunto de componentes que requieren el soporte de una completa infraestructura de red. Sin embargo, existen factores ajenos que pueden afectar a la estabilidad de sistema, por lo que conviene proporcionar mecanismos de recuperación ante errores que aseguren la construcción de un sistema robusto. A continuación, se muestran las tareas que han permitido la realización de este subobjetivo.

2. OBJETIVOS

- Recuperación del sistema de visualización en casos en los cuales la conexión no pueda establecerse.
- Recuperación del sistema de visualización en casos en los que se reciba la información de un programa incompleto.
- Recuperación del sistema de visualización en casos en los cuales el código QR escaneado no sea el correcto.
- Recuperación de la infraestructura de red cliente-servidor en caso en que se interrumpa la conexión en algunas de las partes.
- Recuperación del sistema ante condiciones de red desfavorables, como alta latencia o bajo ancho de banda.

Capítulo 3

Objectives

Given the approach and ideas presented in the introductory chapter (see § 1), this section is going to focus on detailing in an exhaustive manner what this work is intended to carry out, synthesising the general objective of the project and describing the derived specific ones.

3.1 General objective

The main objective of this Final Degree Project (FDP) is to develop a software system whose purpose is to **generate graphical representations using metaphors or abstractions based on a notation of traffic signs and roads for the visualization of programs and algorithms using AR**. In this way, the fulfilment of this objective will have a direct impact on the results obtained in the overall context of the research project, when it comes to improve programming learning.

To this end, a system based on advanced interaction and visualization techniques is proposed through AR, providing an interaction and visualization experience.

This system uses the physical space to locate and align the graphical representations in real time in order to show the static structure of programs. These representations are automatically generated by the system, taking advantage of its integration of COLLECE 2.0 to make use of the benefits and facilities that it provides.

3.2 Specific objectives

Based on the main objective described in the previous section, the specific sub-objectives to be addressed are presented below. They can be grouped around the following lines: creation of an environment integrated in COLLECE 2.0 for the self-generation of graphic representations, development of a system to handle the representation of programs and algorithms, development of the User Interface (UI) and proposal of a scalable architecture. In addition, the system's stability and error recovery is added to all this.

3. OBJECTIVES

3.2.1 Environment integrated with COLLECE 2.0

COLLECE 2.0 is a plug-in for the Eclipse platform that provides a set of collaborative features to help students learn programming. This work is the basis of the environment discussed in this section, proposed as a new plug-in, to take advantage of the various collaboration features it offers. The following lines describe the tasks that have been conducted for the development of the proposed plug-in.

- Creation of a plug-in for the Eclipse tool that serves as a basis for the representation of graphic components.
- Integration of the aforementioned plug-in into COLLECE 2.0 to access the available communications infrastructure.
- Analysis of the source code of a program selected by the user in order to extract the necessary information for its later representation.
- Generation of JSON files as from the processing of source code obtained by the previous point.
- Creation of a server that communicates with the AR device.
- Automatic generation of QR codes with the necessary network connection information to establish communication with the server.
- First-person retransmission of the content displayed by the AR device.
- Sending JSON files to the AR device.
- Development of an event system to facilitate the management of events that occur throughout the execution of the system.

3.2.2 Environment for visualisation of programs

This sub-objective lists the tasks that have been undertaken to build a system capable of visualizing, through graphic components, the source code of a program.

- Creation of a client that communicates with the server.
- Reading QR codes using the front camera of the AR device.
- Receiving of JSON files from the server to visualize the graphical representations.
- Processing of the JSON file for the construction of the visualization corresponding to the static representation of the analyzed program.
- Development of a set of interaction mechanisms to facilitate the use of the device.
- Detection of the different surfaces that make up the physical environment to place the graphic components that represent the program to be displayed.
- Adjustment of size, orientation and position of the graphic representations taking into account the surfaces on which they are placed.

3.2.3 Development of user interface

This sub-objective lists the tasks that have been carried out to develop the visual components that make up the system's user interface.

- Design and development of a view for the plug-in integrated in COLLECE 2.0, in which a QR code is shown with the connection information, the JSON representation of the program to be displayed, and the live retransmission of what the user sees through the AR device.
- Design and adaptation of the metaphor of roads and traffic signals that represent both control structures and expressions of the source code of a program.
- Design and development of visual components (widgets) such as menus, buttons, progress bars and icons that facilitate the interaction with the AR device.

3.2.4 Scalable architecture

This work has been carried out with a view to future extensions to improve the system. The following tasks are those that have made it possible to carry out these sub-objectives.

- Support for the analysis of different programming languages (the programs analyzed in this version are exclusively in Java).
- Isolation of components to prevent them from having dependencies with others.
- Creation of a system of scenes that allows new functionalities to be added.
- Modular design of graphical representations to replace the notations based on the metaphor of roads and traffic signs with another one.
- Appropriate use of design patterns and other programming practices that provide mechanisms to build a sustainable system.

3.2.5 Error recovery

The system is composed of a set of components that require the support of a complete network infrastructure. However, there are external factors that can affect the stability of the system, so it is important to provide error recovery mechanisms to ensure the construction of a robust system. The following tasks are those that have enabled to achieve these sub-objectives.

- Recovery of the display system in cases when the connection cannot be established.
- Recovery of the display system when an incomplete program is received.
- Recovery of the display system in cases when the scanned QR code is not correct.
- Recovery of the client-server network infrastructure in case of connection interruption.
- Recovery of the system from unfavourable network conditions, such as high latency or low bandwidth.

Capítulo 4

Estado del arte

Este capítulo recoge una recopilación de elementos principales sobre los que se asienta este trabajo y que constituyen su base tecnológica y de conocimiento. En este sentido, el apartado comienza con un estudio acerca de la evolución, contexto y soluciones sobre los sistemas para la visualización de programas y algoritmos, seguido de entornos de desarrollo extensibles utilizados en la actualidad. Así mismo, se realiza especial hincapié en los dispositivos de realidad aumentada y algoritmos de tracking existentes hasta el momento, junto a las representaciones gráficas (metáforas/analogías) utilizadas durante el aprendizaje de la programación. Por último, se describe un conjunto de tecnologías de red que permiten la comunicación entre diferentes dispositivos, y se justifican las decisiones finalmente adoptadas para el desarrollo del presente trabajo.

4.1 Sistemas para la visualización de programas y algoritmos

Durante más de 25 años, la visualización de programas y algoritmos se ha establecido como un tema de investigación que despierta un gran interés en el aprendizaje de la programación. El principal propósito de estos trabajos ha sido reducir el nivel de abstracción que la programación requiere usando tecnologías de visualización para mejorar su comprensión [HDS02] y cumplir con el objetivo del nivel 2 de la taxonomía de Bloom [B⁺56], el cual pretende que los estudiantes comprendan hechos o ideas mediante demostraciones.

Varias investigaciones han demostrado que las capacidades cognitivas de los seres humanos están optimizadas para procesar información de una manera multimodal. Aun así, el código fuente de los programas es presentado de un modo textual, desaprovechando así la mayor parte del potencial de nuestro cerebro [Mye90].

Además, las dificultades encontradas por los profesores a la hora de crear representaciones gráficas para aportar un enfoque pedagógico y visual han hecho que este tipo de técnicas no se popularicen en las aulas [Tör09].

No obstante, este trabajo en los últimos años se ha visto intensificado conforme al auge de nuevos dispositivos que mejoran la experiencia de visualizar estas representaciones. El uso de estas tecnologías emergentes posibilita un proceso basado en interacción multimodal, el cual permite un aprendizaje más activo [CB16].

4. ESTADO DEL ARTE

A lo largo de los últimos años se han propuesto diferentes trabajos relacionados con la visualización de programas y algoritmos que aportan resultados tanto a favor como en contra de la efectividad de su uso en el contexto educativo. A continuación, se presentan un conjunto de soluciones cuyo fin es aportar una visión general a cerca de los trabajos que se han llevado a cabo en los campos de la visualización de programas y algoritmos.

Dentro del grupo de sistemas basados en la visualización de programas destaca la herramienta SRec desarrollada en [VHP17], la cual muestra la ejecución de procesos recursivos implementados en Java.

El sistema soporta un extenso número de representaciones que abarcan desde árboles recursivos hasta grafos de dependencia. Además, aparte de las visualizaciones sobre las que el sistema se sustenta para ayudar a los usuarios a la comprensión de la tarea realizada, éste permite la eliminación de recursividad redundante para ayudar a los estudiantes en la comprensión y análisis de algoritmos.

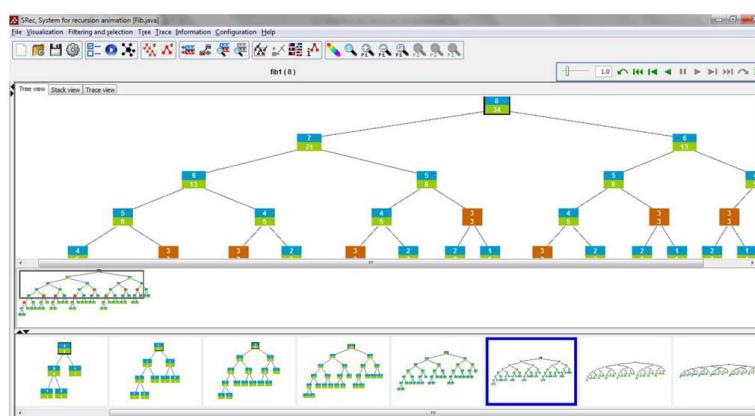


Figura 4.1: Captura del pantalla del sistema SRec en una sesión

El objetivo de la realización de este trabajo fue evaluar la herramienta SRec para conocer el efecto que este tipo de herramientas tienen en la motivación extrínseca e intrínseca de los estudiantes en lo que respecta al aprendizaje de la recursividad. En el artículo se plantea una hipótesis que pretende confirmar el aumento de la motivación en estudiantes para resolver tareas de programación cuando se usan herramientas de visualización de programas, concluyendo positivamente tras un estudio con grupos de estudiantes.

Otro enfoque para representar programas usando técnicas de visualización es la planteada en [DCP01], en el cual se hace uso animaciones tridimensionales para permitir a los estudiantes ver sus propios programas en ejecución.

En dicho trabajo se presenta Alice, un entorno gráfico interactivo tridimensional de programación que ofrece un lenguaje completo y un entorno de creación de prototipos para el comportamiento de objetos tridimensionales (p. ej., animales y vehículos) en un entorno virtual. Además, se apoya en un conjunto de objetivos pedagógicos, como lo son la introducción a objetos, métodos, sentencias condicionales, bucles y recursividad.

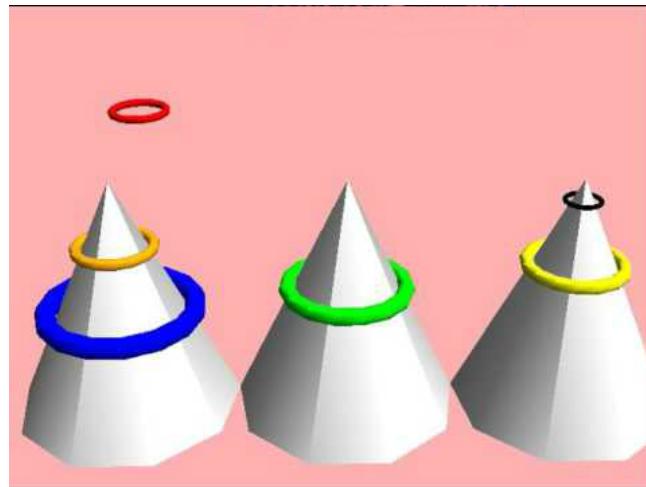


Figura 4.2: Torres de Hanoi en progreso utilizando Alice tool

El propósito de este trabajo se centró en el uso de dicha herramienta con la intención de solventar un problema recursivo que requería de la necesidad de repetir un número desconocido de acciones. A partir de esto se concluyó, que aunque los estudiantes no solventaron satisfactoriamente el problema, demostraron ciertas facilidades para tratar de resolverlo.

Dentro del grupo de visualización de algoritmos, uno de los trabajos relevantes es el expuesto en [KLH97], el cual propone la resolución de un problema típico de algoritmia conocido como el *el problema de la mochila*.

El problema consta de una mochila en la cual se puede introducir un número de barras de diferentes pesos hasta un peso máximo. El propósito del algoritmo es comprobar si existe alguna combinación de barras que pueda llenar la mochila según el peso indicado.

Este problema involucra una búsqueda exhaustiva de todas las combinaciones posibles de barras, la cual suele hacerse de una manera recursiva. Este problema fue expuesto a un conjunto de alumnos de la asignatura de Estructura de Datos, cuya resolución fue dada de la siguiente manera.

Knapsack Algorithm Animation		
Weight 1 (6)	\$\$\$\$\$\$	Weight in Bag
Weight 2 (5)	\$\$\$\$\$	11
Weight 3 (4)	\$\$\$\$	
Weight 4 (3)	\$\$\$	Weight Left
Weight 5 (0)		1
Weight in Sack	+----5----0---+	
Weight Number	 \$\$\$\$\$\$\$\$\$ 	
	1 2	
	+-----+-----+	

Figura 4.3: Resolución del problema mediante animación textual

La conclusión obtenida a partir de los datos recogidos del experimento resume que este tipo de estrategias son una forma efectiva de enseñar algoritmos, ya que la parte de estu-

4. ESTADO DEL ARTE

diantes que pudo ver dicha animación reconoció que el problema planteado era recursivo. Además, los alumnos demostraron tener una mejor comprensión del algoritmo reflejado en el código.

Otra propuesta diferente es la presentada en [LBAU03], cuya herramienta hace uso de animaciones para ayudar a estudiantes inexpertos a entender conceptos de algoritmia y programación. Aunque esta propuesta podría enmarcarse en ambos grupos, este proyecto se ha agregado a este conjunto debido una mayor claridad en la presentación de algoritmos.

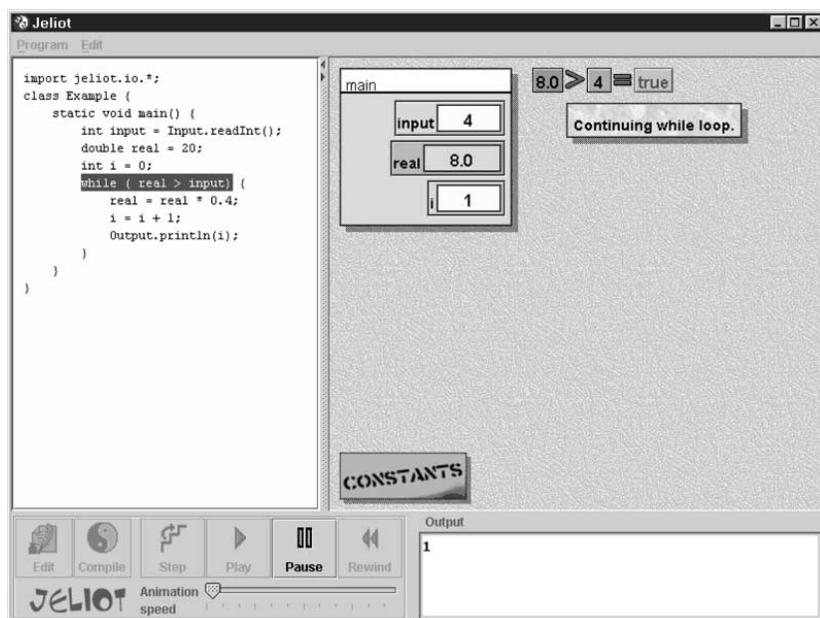


Figura 4.4: Representación de algoritmo en Jeliot 2000

Esta herramienta, llamada Jeliot 2000, se centra en el estudio de conceptos como son la asignación de variables, E/S y flujo de control, entre otros aspectos. Además, permite a los estudiantes ver el código fuente junto al conjunto de entidades que representan el algoritmo que va a ser desarrollado a lo largo de la ejecución. Como novedad, este sistema es diferente de los otros sistemas de animación existentes, ya que las animaciones producidas en Jeliot 2000 son generadas automáticamente, las cuales no requieren de la interacción del usuario.

En dicho trabajo se llevó a cabo un experimento en el cual se pretendía comprobar el impacto de la herramienta en los estudiantes. Dicho experimento fue desarrollado en dos clases paralelas de introducción a la informática, de las cuales una usaría la herramienta Jeliot 2000 para representar cierto tipo de conceptos mientras que la otra prescindiría de ella.

Los resultados obtenidos al final del curso demostraron que el grupo que utilizó la herramienta de visualización durante las clases mejoró considerablemente la comprensión de estructuras de control como *if-then-else* y *while*.

Todos los trabajos analizados anteriormente se encuentran en un marco de sistemas de interacción tradicional que ayudan en las aulas a la mejora de la enseñanza. Sin embargo, las tecnologías emergentes como la RA han dado lugar a la elaboración de nuevas soluciones gracias a las ventajas que ésta ofrece en el ámbito de la educación [PCGA15].

En [TC12] un entorno de RA fue creado para ayudar a estudiantes a aprender programación gráfica empleando la biblioteca OpenGL y utilizando códigos QR que muestran su información tridimensional en un espacio virtual. Los estudiantes programan comandos de OpenGL en códigos QR y el sistema se encarga de representar de forma visual la transformación de un objeto asociada al comando programado.

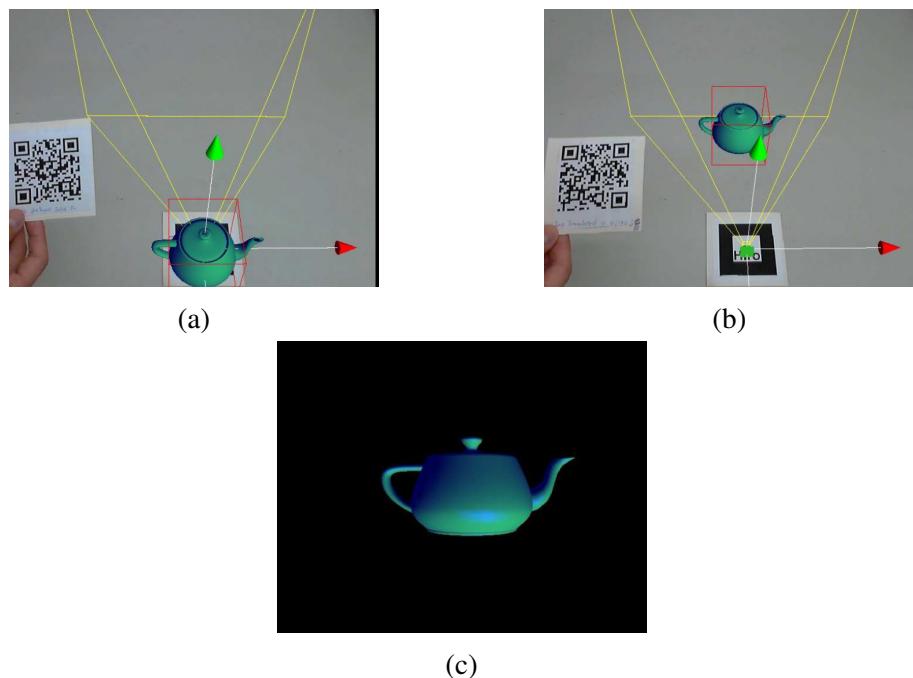


Figura 4.5: (a) Taza virtual sobre la marca (b) Taza trasladada (c) Renderizado utilizando la cámara virtual ubicada en la marca

Este trabajo propone dos ejemplos con el fin de demostrar la efectividad del sistema expuesto en el artículo. En ambos casos se muestran los resultados de aplicar una serie de operaciones tridimensionales a objetos para ilustrar la relación entre los códigos QR, imagen OpenGL renderizada y la configuración de la cámara virtual y objetos en el espacio tridimensional.

Tras la realización de los ejemplos, se demostró la capacidad y flexibilidad del sistema para manipular diferentes objetos en el espacio de trabajo virtual, además de permitir a los estudiantes visualizar y comprender el conjunto de instrucciones OpenGL.

Finalizando con el apartado, otro trabajo relacionado con la RA es el presentado en [TCC17], el cual introduce un sistema mejorado de RA que ofrece representación visual e interactividad para ayudar a los estudiantes a aprender a programar aplicaciones 3D.

4. ESTADO DEL ARTE

Para comprobar la forma en la que el sistema afecta a los alumnos durante las clases, se realizó un experimento con un grupo de 34 estudiantes que usaron una versión inicial del sistema y otra mejorada.

La versión inicial del sistema contiene dos mecanismos para ayudar a los estudiantes en el proceso de aprendizaje: una ventana de código y otra para mostrar la salida de la ejecución. Del mismo modo, la versión mejorada cuenta con los mismo mecanismos que la anterior, sin embargo, ésta añade dos más: una ventana global y unas tarjetas de comandos (códigos QR).



Figura 4.6: Versión mejorada del sistema de aprendizaje de RA

Para llevar a cabo el estudio propuesto en el trabajo se abordaron dos cuestiones. La primera cuestión se planteó para comprobar si incorporar tecnología de RA permitiría desarrollar un sistema para el aprendizaje de la programación, mientras que la segunda se propuso para verificar si el sistema discutido en el estudio beneficia al estudiante en el aprendizaje de la programación.

Las observaciones extraídas de los resultados obtenidos del experimento demostraron que el sistema mejorado permitió que los estudiantes obtengan mejores resultados de aprendizaje respecto del sistema inicial, debido principalmente a una mejora en usabilidad de la nueva versión.

4.2 Entornos de desarrollo extensibles

Hoy en día, el uso de entornos de desarrollo para la construcción de herramientas software es muy común en el campo de la informática, ya que éstos proporcionan una variedad de servicios integrados que permiten facilitar la labor del desarrollador.

Normalmente, estos entornos proporcionan un editor de código fuente, herramientas de construcción automáticas, depurador, resaltado de sintaxis de código, etc. Sin embargo, los entornos de desarrollo extensible, además de proporcionar este tipo de funcionalidades, tam-

bién permiten extender la plataforma a través de complementos (*plug-ins* o *add-ons*) para agregar funcionalidades nuevas que previamente no incorporan.

A lo largo de esta sección se presentan un conjunto de entornos extensibles que permiten extender las funcionalidades de la herramienta para ayudar al programador en las tareas de desarrollo.

Entre los entornos de desarrollo extensibles disponibles podemos nombrar **NetBeans**¹ (ver Figura 4.7), una plataforma de código abierto y libre principalmente destinada al desarrollo de aplicaciones del entorno Java (Java SE, Java EE, etc.). Este entorno cuenta con algunas características, tales como el resaltado de sintaxis, autocompletado, gestión de proyectos, herramientas de depuración y desarrollo rápido de interfaces, entre otras.

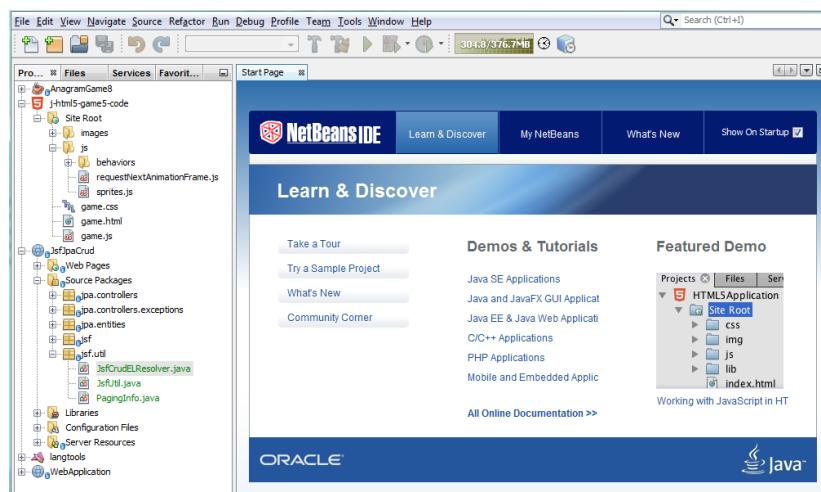


Figura 4.7: Captura de pantalla del editor de NetBeans

Esta plataforma además añade el soporte de plug-ins para extender las funcionalidades de la herramienta. NetBeans está compuesto por módulos, los cuales proporcionan funcionalidades definidas que facilitan el desarrollo de componentes. Para ello, NetBeans se apoya en un conjunto de módulos integrados para tratar de agilizar la construcción de interfaces gráficas, gestión de documentos, ventanas, etc.

Otro entorno destacado por su sistema de creación de componentes es **IntelliJ IDEA**² (ver Figura 4.8), un IDE para el desarrollo software diseñado para la construcción de programas en Java. Esta herramienta integra resaltado de sintaxis de código, autocompletado, herramientas de refactorización y soporte de histórico de cambios. Además, soporta control de versiones como Git, Mercurial, SVN, y bases de datos como SQL Server, MySQL, etc.

Esta plataforma también incorpora el ecosistema de plug-ins a través de los cuales se pueden añadir funcionalidades al entorno. IntelliJ IDEA incluye un gran número de componentes que facilitan la construcción de menús, diálogos, notificaciones, ventanas, etc. Para ello, la

¹<https://netbeans.org/>

²<https://www.jetbrains.com/idea/>

4. ESTADO DEL ARTE

plataforma hace uso de acciones del sistema para permitir añadir sus propios objetos a menús, barras y botones, entre otros; herramientas de ventana para mostrar información a través de paneles; y componentes de editor para beneficiarse del soporte que la herramienta cuenta con respecto a la edición de código, entre muchas otras funcionalidades.

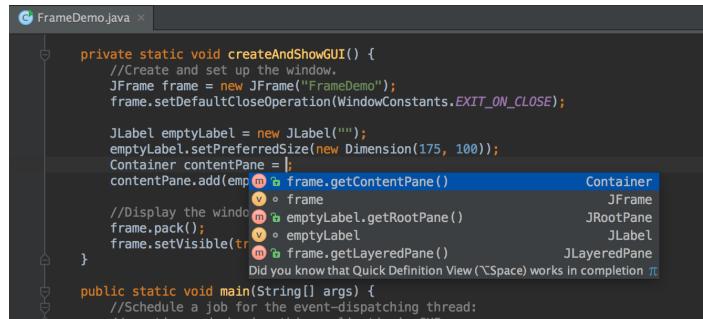


Figura 4.8: Captura de pantalla del editor de IntelliJ IDEA

Finalmente, otro de los entornos a destacar, es la plataforma de código abierto **Eclipse**³ (ver 4.9), diseñada para construir Entornos de Desarrollo Integrados (del inglés Integrated Development Environments (IDEs)) que pueden ser usados para crear aplicaciones desde sitios web hasta sistemas empotrados. Aparte, esta plataforma soporta el lenguaje de programación Java y cuenta con resaltado de sintaxis de código, autocompletado, gestión de proyectos y programas de compilación y ejecución, entre otros.

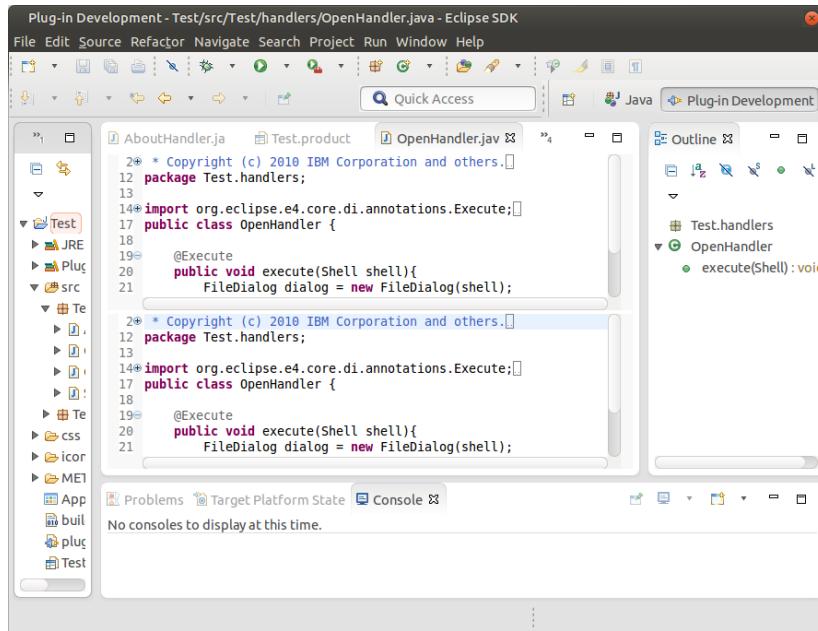


Figura 4.9: Captura de pantalla del editor de Eclipse

Esta plataforma es además conocida por el sistema de plug-ins que integra, ya que permite al desarrollador construir aplicaciones que directamente puedan beneficiarse de las características y ventajas que ésta ofrece.

³<https://www.eclipse.org/>

Típicamente, un plug-in consiste en código Java, ficheros de sólo lectura y otros recursos como imágenes, plantillas, bibliotecas, etc. Para su desarrollo, Eclipse proporciona un conjunto de componentes gráficos que permite al desarrollador mediante Standard Widget Toolkit (SWT) y JFace crear interfaces gráficas para ser representadas en la plataforma. Además, otro componente importante es Java Development Tools (JDT), el cual proporciona un conjunto de APIs para acceder y manipular código fuente de Java.

4.3 Dispositivos de Realidad Aumentada y algoritmos de tracking

Los primeros usos del término de RA surgen de un trabajo de Sutherland en 1968 [Sut68], el cual usó un dispositivo con pantalla transparente en forma de casco (o HMD) para mostrar gráficos tridimensionales. Sin embargo, no fue hasta pasada una década en la que los ingenieros de *Boeing* propusieran esta tecnología en un entorno profesional para mejorar el proceso en la fabricación de aviones [VKP10].

En 1997, Azuma publicó un estudio [Azu97] en el que define la RA como un entorno donde el usuario puede ver el mundo real compuesto de objetos virtuales sobre éste. Además, se describieron varios problemas respecto a este nuevo enfoque, además de algunos desarrollos producidos hasta ese momento, lo que consiguió que se empezara a popularizar el uso de dicho término.

Décadas atrás, la creación de entornos de RA se generaban mediante aparatos externos al HMD para obtener la posición y orientación de la cabeza para lograr una visualización alineada con el espacio (ver Figura 4.10). Sin embargo, gracias a los avances realizados en este tipo de dispositivos, se ha prescindido del uso de sistemas externos que ayuden a calcular la información espacial del dispositivo.

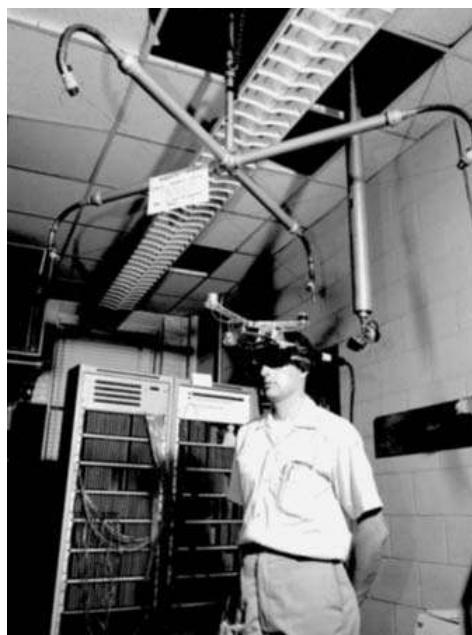


Figura 4.10: Brazo mecánico usado para medir la orientación de la cabeza del usuario

4. ESTADO DEL ARTE

Son muchos los avances que han surgido dentro del campo de la RA, no obstante, a pesar de esto, la taxonomía propuesta en [MTUK95] sigue vigente a lo largo de los años. Esta clasificación incluye dos clases: dispositivos de RA basados en monitores y dispositivos de RA con pantallas transparentes. Los primeros se caracterizan por agrupar aquellos dispositivos que tienen la habilidad de mostrar imágenes generadas por ordenador superpuestos analógica o digitalmente a la imagen en vivo o diferido. El caso de smartphones o tablets es un claro ejemplo de este tipo de dispositivos, ya que se tratan de un pequeño computador que superpone el contenido virtual sobre la imagen real capturada utilizando la cámara del dispositivo (ver Figura 4.11a).

De forma análoga, un ordenador de escritorio o portátil puede emplear una cámara externa (webcam) para superponer imágenes de objetos reales con virtuales siguiendo el mismo proceso del que hacen uso smartphones y tablets (ver 4.11b).



Figura 4.11: (a) Objeto tridimensional superpuesto en la pantalla de un smartphone y (b) un computador

Con respecto a los dispositivos de RA con pantallas transparentes, éstos tienen la habilidad de superponer tanto objetos virtuales como reales en el mundo real a través de lentes. Tales dispositivos son conocidos como HMD. Debido a los avances tecnológicos, el número de dispositivos de este tipo ha aumentado considerablemente.

Uno de ellos son las gafas de RA de **Epson**⁴ (ver Figura 4.12a). Estas gafas cuentan con un campo de visión de 22 grados, el cual dificulta la inmersión del usuario en un entorno holográfico tridimensional, ya que el campo de visión recomendado es mayor o igual a 30 grados (el campo de visión humano es de unos 180 grados). Sin embargo, una característica importante a destacar de este dispositivo es su portabilidad, ya que, aunque necesita la conexión de una unidad portátil para gestionar la batería, almacenamiento y la ubicación GPS, no requiere de conexión a un computador.

En segundo lugar, **Atheer**⁵ (ver Figura 4.12b) permite mostrar contenido tridimensional

⁴<https://epson.com/>

⁵<https://atheerair.com/>

del mismo modo que el sistema anterior, pero con un campo de visión de 50 grados, habilitando la inmersión del usuario. En cuanto a la portabilidad, este sistema también necesita conectarse a una unidad portátil, la cual procesa la interacción del usuario con los modelos tridimensionales, aparte de proporcionar conectividad tanto Wi-Fi como Bluetooth.

Otro dispositivo dentro de este grupo son las gafas de RA de **Meta**⁶ (ver Figura 4.12c), las cuales se caracterizan por permitir al usuario tocar, agarrar o manipular objetos tridimensionales. Además, cuentan con un campo de visión de 90 grados, el cual permite al usuario sumergirse en un entorno prácticamente tridimensional. Otra característica a destacar de este dispositivo es que cuenta con la capacidad de reconocer o detectar ciertas superficies del entorno, aunque con ciertas limitaciones. En cuanto a la portabilidad del dispositivo, éste necesita estar conectado a un computador, lo cual impide una mayor libertad de movimiento.

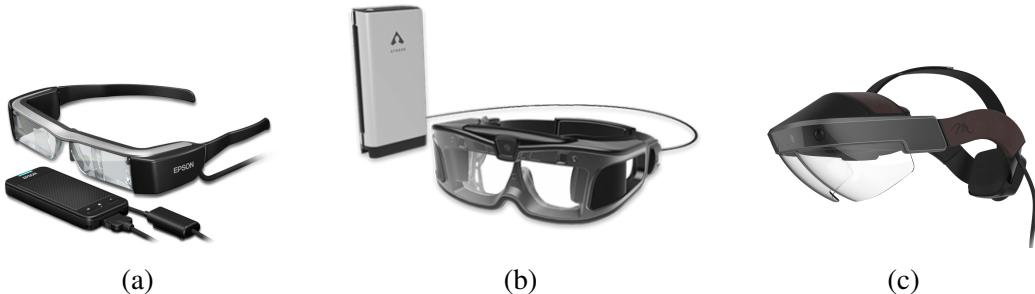


Figura 4.12: (a) Dispositivos de RA de Meta (b) Epson y (c) Athear

Por último, cabe destacar las gafas de RA de **Microsoft**⁷ (ver Figura 4.13), las cuales se basan en la tecnología de Microsoft Kinect y algoritmos de seguimiento o tracking. Este dispositivo, al igual que los anteriores, cuenta con dos lentes transparentes que permiten mostrar el contenido tridimensional. El modo en el que este dispositivo visualiza el contenido se debe a un material ligero en las lentes que muestra imágenes en color al mismo tiempo que deja pasar la luz. A diferencia de los otros dispositivos, HoloLens integra un amplio conjunto de sensores usados para alinear el mundo real con el mundo virtual.

El primer sensor a destacar dentro de este conjunto es el micrófono, el cual permite distinguir entre comandos de voz y ruido ambiental. Estos comandos son reconocidos mediante la tecnología de reconocimiento del habla basada en la nube del asistente personal Cortana.

Para obtener la orientación de la cámara, HoloLens cuenta con una unidad de medición inercial (del inglés Inertial Measurement Unit (IMU)) compuesto por un giroscopio, un magnetómetro y un acelerómetro. El giroscopio es el encargado de calcular la orientación de la cabeza, comparando la velocidad de los movimientos junto con el tiempo transcurrido de éste. Sin embargo, los datos del giroscopio pierden precisión con el tiempo debido a la acu-

⁶<http://www.metavision.com/>

⁷<https://www.microsoft.com/es-es/hololens>

4. ESTADO DEL ARTE

mulación de errores. Por ello, el dispositivo utiliza los sensores restantes para corregir este error. El acelerómetro, el cual mide la aceleración, proporciona la posición de la cabeza al oponerse ésta contra la aceleración de la gravedad. No obstante, para obtener la información con mayor precisión, se hace uso del magnetómetro, el cual mide la fuerza y dirección de un campo magnético.

Esta configuración permite obtener datos posicionales rápidos, pero sigue siendo poco preciso. Debido a esto, el dispositivo hace uso de cámaras de escala de grises, las cuales obtienen datos de posicionamiento de forma más lenta, pero más exactos.



Figura 4.13: Microsoft HoloLens

Otra característica con la que cuenta el dispositivo es la posibilidad de descubrir la profundidad del entorno. Gracias a un sensor de profundidad es capaz de determinar la distancia entre el usuario y las superficies del espacio físico empleando la tecnología de *Tiempo de Vuelo* (del inglés Time-of-Flight (ToF)) [MZC12], basada en el cálculo de tiempo que tarda un pulso de luz infrarroja emitido desde el dispositivo en volver a éste.

Aparte de lo anteriormente mencionado, este dispositivo cuenta con un campo de visión de 30 grados (campo de visión mínimo recomendado para una correcta inmersión durante una experiencia de RA) e integra una unidad de procesamiento, la cual dota al dispositivo de una mayor portabilidad y libertad de movimiento.

Todos estos dispositivos, para proyectar objetos tridimensionales superpuestos sobre el mundo real, se basan en algoritmos de tracking, los cuales tratan de obtener una estimación de la trayectoria en el espacio realizada por un objeto. Para llevar a cabo esto, los dispositivos discutidos en esta sección hacen uso de cámaras que emplean técnicas de visión por computador para obtener el posicionamiento en seis grados de libertad (tres grados de posición y otros tres de orientación) [KB99].

Para calcular estos grados se necesitan diferentes referencias tridimensionales. En [GVAC13] se proporciona una taxonomía donde se distinguen las siguientes aproximaciones: **Bottom-Up** y **Top-Down**.

El método Bottom-Up trata de obtener los seis grados de libertad de la cámara a partir de las características de la geometría del objeto junto a sus relaciones geométricas tridimensionales. Teniendo en cuenta esto, este método distingue entre tracking con marcas y tracking sin marcas. El primer método hace uso de marcas cuadradas o circulares que facilitan su

detección debido a su alto contraste, mientras que el segundo método emplea características naturales de la escena usando técnicas de visión por computador.

En cuanto al método Top-Down, éste se centra en estimar la geometría del objeto basándose en el contexto de la escena. A través de la posición de la cámara se buscan referencias en la escena que corrijan las predicciones previas para ayudar en la creación del modelo en el entorno.

4.4 Metáforas utilizadas durante el aprendizaje de la programación

En el ámbito de la literatura, las metáforas expresan por medio de conceptos o realidades diferentes ideas o términos que guardan una relación de semejanza entre sí. De igual modo, en el ámbito de la informática se suelen utilizar representaciones mediante visualizaciones o animaciones 3D, con el fin de reducir las capacidades cognitivas que requieren conceptos de difícil comprensión.

Sin embargo, la mayoría de estas representaciones siguen planteando ciertas dificultades para mostrar una relación directa con los conceptos que se pretenden enseñar. En este sentido, a lo largo de la literatura existente múltiples autores que han aportado diferentes ideas y diseños basados en metáforas para mejorar el proceso de explicación de conceptos de gran abstracción.

En [WL07], se presenta una visualización tridimensional usando una metáfora de ciudades con el fin de mejorar la explicación conceptos de Programación Orientado a Objetos (POO). Mediante esta idea, los elementos de dicho paradigma como paquetes, clases, métodos y atributos, entre otros, son relacionados directamente utilizando las representaciones propuestas.

Otro trabajo similar es el realizado en [KMT⁺17], el cual propone un entorno de juego 3D para visualizar clases y código fuente a través de una representación gráfica basada en habitaciones. El uso de esta analogía pretende reducir la carga cognitiva de los alumnos, además de mejorar la comprensión del código.

Similarmente, en [TV04] se hace uso de una metáfora de paisajes para comprender arquitecturas basadas en componentes. La representación utilizada para el estudio es la montaña, que representa las clases utilizadas durante el desarrollo de un sistema, variando la altura de las cordilleras respecto a las líneas del código o dependencias entre clases.

Finalmente, en [SGG⁺18] se propone una notación basada en una metáfora de carreteras y señales de tráfico. Este conjunto de representaciones permite identificar 5 tipos de instrucciones que soporta cualquier lenguaje de programación: condiciones, bucles, definición de funciones, sentencias de retorno y expresiones. De este modo, el usuario puede seguir el flujo de ejecución de un programa al encontrarse éste familiarizado con ellas en su vida diaria.

4.5 Tecnologías de red

Una red es un conjunto de equipos informáticos conectados entre sí mediante medios físicos o inalámbricos para el transporte de datos. La primera comunicación entre dos equipos se produjo entre la Universidad de California y la Universidad de Standford en 1969. Sin embargo, la información que podía transmitirse a través de esta comunicación era prácticamente nula.

Desde entonces, la comunicación entre dispositivos ha evolucionado considerablemente. En este ámbito se han producido numerosos avances con el fin de crear modelos de comunicación que ofrezcan soluciones y mejoras para la gestión y transmisión de la información.

Uno de los conceptos planteados para facilitar el envío y recepción de información por red fue el de **Socket**, el cual se define como una identificación única hacia o desde la cual se transmite la información en la red [Win71]. Los socket de red se caracterizan por una combinación de:

- Socket local: dirección Internet Protocol (IP) local y número de puerto.
- Socket remoto: usado en comunicaciones que utilizan el protocolo de transporte Transmission Control Protocol (TCP).
- Protocolo: dependiendo de la conexión que se quiera establecer, se usa algún protocolo de transporte como User Datagram Protocol (UDP), no orientado a conexión; y TCP, orientado a conexión.

Otra tecnología muy popular debido a la importancia de transferencia de comunicación a través de la World Wide Web (WWW) es **Hypertext Transfer Protocol (HTTP)**, un protocolo de nivel de aplicación que sigue el esquema petición-respuesta entre un cliente y un servidor [FR14]. Un ejemplo de aplicaciones que usan este tipo de modelo son los navegadores web y las arañas web (programas que inspeccionan la WWW para recopilar información).

En este contexto, la tecnología **Remote Procedure Call (RPC)** fue diseñada para que una máquina ejecutase código remotamente en otra, sin tener que preocuparse por la comunicación entre ambas [BN84]. Derivado de esta conexión surge **Java RMI**, la cual permite ejecutar código remoto en el contexto de la POO.

Por otra parte, **ZeroC Ice** es un middleware orientado a objetos que proporciona un conjunto de herramientas, APIs, y soporte de bibliotecas para construir aplicaciones cliente-servidor, cuyos elementos pueden ejecutarse en entornos heterogéneos (entornos con diferentes lenguajes de programación, sistemas operativos y arquitecturas) [VF06]. En línea con esto, cabe destacar **Eclipse Communication Framework (ECF)**, otro middleware de comunicaciones que abstrae al desarrollador de tener que tratar con los aspectos de bajo nivel de la arquitectura de red, proporcionando mecanismos de alto nivel para ejecución de métodos remotos, comunicación entre usuarios y transmisión de objetos completos, entre otros.

Las tecnologías anteriores se pueden utilizar para construir arquitecturas de red muy diversas. Una de ellas es aquella basada en un modelo **Cliente-Servidor**, el cual se utiliza para establecer una comunicación entre dos o más procesos para compartir varios recursos o servicios [Pad82]. En este modelo el servidor se encarga de compartir los recursos, mientras que el cliente se ocupa de iniciar el contacto con el servidor para usar dichos recursos.

4.6 Conclusión

Dada la envergadura del proyecto, se ha necesitado utilizar diferentes tecnologías y herramientas que ayuden a resolver el problema expuesto en el presente trabajo. En primer lugar, en cuanto a entornos de desarrollo extensibles, Eclipse ha sido la opción elegida para crear un entorno que ayude en la enseñanza de la programación. Esta decisión se debe a la popularidad de la herramienta utilizada en cursos introductorios a la programación, además de la facilidad de extender las capacidades de la misma. Este hecho ha evitado que los alumnos no presenten problemas a la hora de utilizar la extensión desarrollada para este proyecto.

En segundo lugar, tras llevar a cabo una comparativa sobre las características y funcionalidades de los dispositivos de RA en la actualidad, destaca el dispositivo Microsoft HoloLens, que resalta por las ventajas que proporciona en cuanto al reconocimiento de superficies del espacio físico real, y la incorporación de una unidad de procesamiento que evita la utilización de unidades portátiles externas (en el Anexo A se pueden encontrar las características completas del dispositivo).

En último lugar, dado que el entorno necesita establecer una comunicación de red entre sistemas independientes (Eclipse y HoloLens), se desarrolló una arquitectura basada en el modelo cliente-servidor para permitir comunicar los dispositivos involucrados en la estructura distribuida.

Capítulo 5

Método de trabajo

TODO proyecto de ingeniería necesita de una metodología para poder abordar de manera óptima los objetivos que se propusieron al comienzo de un proyecto. En este sentido, el trabajo plasmado en este documento ha necesitado de un método para gestionar los diferentes hitos que se establecieron en etapas iniciales con el fin de realizar de forma satisfactoria su implementación. Debido a lo cual, este capítulo se centra en detallar la metodología que se ha seguido y cómo se ha llevado a cabo, junto a los medios hardware y software que se han utilizado para la elaboración del presente trabajo.

5.1 Metodología de trabajo

Dada la naturaleza del proyecto, se ha necesitado de una metodología que permitiera trabajar de forma paralela añadiendo nuevas características y funcionalidades junto a la corrección de errores de cada componente para que, una vez verificado que todos ellos funcionan de manera correcta, se pudieran integrar en el sistema final para su correcto funcionamiento.

El planteamiento que se ha llevado a cabo para seguir el continuo desarrollo del proyecto ha sido mediante reuniones cada 1 ó 2 semanas. El propósito de las reuniones semanales han tenido como fin establecer los objetivos a realizar en la semana posterior y comprobar el estado del trabajo actual.

Cabe destacar que debido a la arquitectura del sistema orientado a componentes y subsistemas, éstos han podido ser desarrollados de forma aislada sin requerir su total integración en el proyecto final. Una vez un componente o subsistema estaba implementado, se realizaban las pruebas pertinentes para dar por completado el hito correspondiente y pasar al siguiente.

5.2 Desarrollo iterativo e incremental

El presente proyecto ha seguido una metodología inspirada en el proceso de **desarrollo iterativo e incremental** [Coc08], la cual plantea una serie de iteraciones proyectadas en un periodo corto de tiempo las cuales pueden volver a ser examinadas en cualquier instante.

Esta metodología es principalmente usada en el mundo del desarrollo software, la cual

5. MÉTODO DE TRABAJO

permite mantener o adaptar el mismo de tal manera que los cambios que se produzcan sean fáciles de integrar en el proyecto. El proceso de desarrollo que se sigue mediante iteraciones en pequeñas porciones de tiempo permite la creación de componentes que son examinados una vez han sido realizados. Una vez se haya comprobado que la funcionalidad es la deseada, se continúa con las iteraciones hasta finalizar los hitos que se establecieron en etapas iniciales.

La Figura 5.1 muestra gráficamente el proceso anteriormente comentado, donde una iteración se define como un desarrollo cíclico compuesto por las fases de análisis de requisitos, diseño, implementación y pruebas. A continuación, se explica en detalle los puntos claves o principios que caracterizan a esta metodología ágil junto a su aplicación en el desarrollo del proyecto.

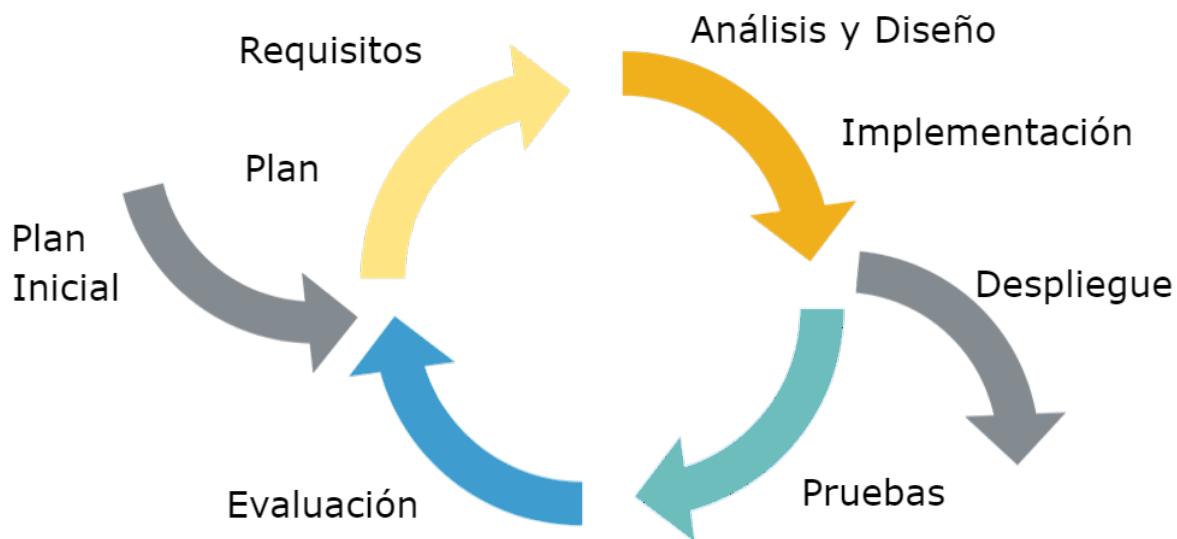


Figura 5.1: Modelo de desarrollo iterativo e incremental

5.2.1 Desarrollo incremental

El desarrollo incremental es una estrategia de organización y programación en el cual varias partes del sistema son desarrolladas en diferentes momentos. El trabajo es dividido en tareas, las cuales son programadas en el tiempo con el fin de ser integradas como un todo en el sistema final.

Esta separación de trabajo permite trabajar individualmente sobre una tarea específica, de tal modo que éstas no interfieran entre sí. En este sentido, el presente proyecto ha seguido esta estrategia a la hora de estructurar las funcionalidades del sistema. Esto ha permitido separar las responsabilidades del trabajo para trabajar de manera paralela en cada uno de los módulos o subsistemas.

5.2.2 Desarrollo iterativo

El desarrollo iterativo es una estrategia en la cual varias partes del sistema son revisadas o inspeccionadas para mejorar o corregir errores de las iteraciones anteriores. Normalmente, las partes que suelen ser examinadas por el equipo de desarrollo están relacionadas con las secciones técnicas, debido a que el proyecto es susceptible de cambios.

En el caso del sistema expuesto en este documento, esta metodología ha sido utilizada para revisar y modificar todos los componentes desarrollados. Esto ha permitido corregir errores y dotar al sistema de robustez con el fin de evitar problemas que pudieran producirse durante la ejecución.

5.3 Planificación

Esta sección pretende mostrar los esfuerzos que se han llevado a cabo para la realización de cada uno de los objetivos propuestos en el capítulo de objetivos (véase § 2). Por un lado y de manera muy resumida se detalla cada uno de los hitos realizados seguido de las fechas en la cual cada tarea se engloba. Por otro lado, se muestra de manera gráfica el esfuerzo empleado en cada uno de los hitos apreciando de tal modo la complejidad y tiempo invertido en cada uno de ellos. Cabe destacar que el presente documento ha sido escrito durante todo el desarrollo del proyecto, abarcando los meses entre octubre y junio, ambos incluidos.

A continuación, se muestran los hitos que tienen que ver con el desarrollo del entorno que actúa como servidor del sistema.

- **Estudio de la creación de plug-ins para Eclipse:** debido a los pocos conocimientos sobre la creación, desarrollo y despliegue de plug-ins para Eclipse, se realizó un estudio del 05/10/17 hasta el 20/10/17 para abordar el desarrollo de este tipo de componentes.
- **Integración de componentes externos:** se integró COLLECE 2.0 para añadir nuevas funcionalidades al entorno que den soporte a los objetivos del proyecto. Los meses empleados para la realización de esta tarea abarcaron desde 22/10/2017 hasta el 02/12/2017, ya que varias iteraciones fueron necesarias para su correcta integración.
- **Implementación de arquitectura de red:** se implementó un servidor en la plataforma de Eclipse para mantener una comunicación con el dispositivo de RA. Esta tarea fue realizada desde el 02/02/2018 hasta el 05/04/2018, debido a que un conjunto de iteraciones fueron necesarias para validar que tanto la recepción como envío de mensajes era correcta.
- **Procesamiento de código fuente:** esta tarea ha sido de vital importancia para el correcto funcionamiento del sistema, ya que esta parte tiene como objetivo generar archivos que puedan ser procesados de manera sencilla por el dispositivo de RA. Esta tarea fue extendida desde el 05/03/2018 hasta el 15/05/2018, dado que un conjunto de

5. MÉTODO DE TRABAJO

iteraciones se llevaron a cabo para verificar que el procesamiento era correcto.

- **Generación automática de códigos QR:** debido a que el entorno debe conectarse con el dispositivo de RA se añadió al sistema la generación automática de códigos QR con el fin de contener la dirección de red del equipo. Esta tarea fue desarrollada desde el 02/01/2018 hasta el 08/02/2018 en la cual sólo una iteración fue necesaria.
- **Sistema de eventos:** dado que el usuario tiene que interactuar con el entorno para poder visualizar un programa determinado, fue necesaria la implementación de un sistema de eventos para notificar a las vistas los sucesos ocurridos durante la ejecución. La elaboración de esta tarea fue extendida desde el 30/02/2018 hasta el 16/04/2018, ya que varias iteraciones fueron desarrolladas para mejorar la escalabilidad del sistema.
- **Retransmisión de vídeo:** el dispositivo de RA permite al usuario estar en un entorno donde se mezcla el mundo real con el virtual. Debido a esto, se desarrolló un componente el cual permite compartir los hologramas en tiempo real con el resto de usuarios que no disponen del mismo. Esta tarea se realizó desde el 01/04/2018 hasta el 06/06/2018 debido a conjunto de iteraciones desarrolladas con el fin de mejorar la composición de imágenes.

Una vez detalladas las tareas involucradas en el desarrollo del servidor, la siguiente imagen muestra de manera gráfica el tiempo invertido en cada una de ellas.

Tarea	Tiempo(meses)								
	Oct	Nov	Dic	Ene	Feb	Mar	Abr	May	Jun
1. Estudio de la creación de plug-ins									
2. Integración de componentes externos									
3. Implementación de arquitectura de red									
4. Procesamiento de código fuente									
5. Generación automática de QR									
6. Sistema de eventos									
7. Retransmisión de vídeo									

Figura 5.2: Tiempo empleado en cada tarea para el desarrollo del servidor

Por otro lado, los siguientes puntos resumen los hitos involucrados en el desarrollo del entorno que actúa como cliente.

- **Estudio del dispositivo de RA:** dado el poco tiempo que este dispositivo lleva en el mercado, fue necesario un estudio previo acerca de la creación, despliegue y desarrollo de aplicaciones con el fin de mostrar las visualizaciones correspondientes de manera correcta. Esta tarea se realizó en paralelo con el estudio de creación de plug-ins para Eclipse.

- **Implementación de arquitectura de red:** se implementó un cliente en el dispositivo de RA para poder recibir los archivos a partir de los que se elaboran las representaciones basadas en carreteras y señales de tráfico. Esta tarea fue realizada desde 02/01/2018 hasta el 10/03/2018, ya que un conjunto de iteraciones fueron necesarias para verificar que tanto la recepción como envío de mensajes era correcta.
- **Lectura de código QR:** dado que el servidor genera códigos QR automáticamente, fue fundamental la implementación de un lector de dichos códigos para poder establecer la comunicación entre ambas partes. Esta tarea se llevó a cabo desde 12/12/2017 hasta el 10/02/2018 debido a que se realizaron diferentes iteraciones para comprobar que la lectura de los códigos era correcta.
- **Procesamiento de archivos JSON:** para la visualización gráfica de los modelos tridimensionales (señales de tráfico y carreteras) se implementó un algoritmo el cual procesara el archivo con la información del programa a representar. Esta tarea fue extendida desde el 05/03/2018 hasta el 20/05/2018, ya que un conjunto de iteraciones fueron necesarias para verificar que el procesamiento era correcto.
- **Diseño de modelos gráficos tridimensionales:** se diseñó un conjunto de componentes tridimensionales con el objetivo de ayudar a comprender de un modo sencillo el flujo de ejecución de un programa. Dada la importancia de este hito, la tarea se alargó desde el 05/11/2017 hasta el 08/03/2018 debido a diversas iteraciones que se han llevado a cabo durante este periodo para su correcta agregación en el proyecto.

Del mismo modo que con el entorno anterior, la siguiente imagen muestra de manera gráfica el tiempo empleado en cada una de las actividades anteriormente detalladas.

Tarea	Tiempo(meses)								
	Oct	Nov	Dic	Ene	Feb	Mar	Abr	May	Jun
1. Estudio del dispositivo de AR									
2. Implementación de arquitectura de red									
3. Lectura de código QR									
4. Procesamiento de archivos JSON									
5. Diseño de modelos gráficos 3D									

Figura 5.3: Tiempo empleado en cada tarea para el desarrollo del cliente

5.4 Medios empleados

En esta sección se exponen todos los medios que se han utilizado para llevar a cabo el presente proyecto. Por un lado, se especificarán los medios hardware que se han previsto necesarios para el desarrollo del trabajo. Por otro lado, los medios software (lenguaje, entorno de desarrollo, herramientas de gestión y planificación, etc.) que se han requerido para

5. MÉTODO DE TRABAJO

la construcción del proyecto, así como las bibliotecas, frameworks, plug-ins, etc., que han tenido mayor impacto en el desarrollo proyecto.

5.4.1 Medios hardware

Esta sección tiene la intención de detallar los medios que se han requerido para la realización del presente sistema. A continuación se muestra una lista con una breve explicación de la utilidad de cada uno de los siguientes medios en el proyecto.

- **Microsoft HoloLens:** para el despliegue del sistema se ha utilizado el dispositivo de RA de Microsoft que permite sumergir al usuario en un entorno 3D virtual. El hecho de su uso se debe a la posición dominante en el mercado con respecto a otros dispositivos, ya que permite identificar un conjunto extenso de superficies en el espacio físico.
- **Computador:** para el desarrollo del sistema se ha utilizado un equipo que cumpla con los requisitos que Microsoft especifica en su página web con el fin de crear aplicaciones para el dispositivo de RA¹. Las características con las que cuenta el computador son las siguientes:
 - Windows 10 Education de 64-bit.
 - CPU con 4 núcleos.
 - 8 GB de memoria RAM.
 - Tecnología de virtualización Hyper-V².

5.4.2 Medios software

Esta sección pretende explicar de manera sintetizada los medios software que se han necesitado para la elaboración del proyecto en cuanto a desarrollo tecnológico se refiere. Por un lado se muestran los lenguajes, bibliotecas, plug-ins, etc., más relevantes que han tenido una mayor influencia en el desarrollo del proyecto, detallando de manera breve el porqué de su elección. Por otro lado, se muestran las herramientas usadas en el proyecto para la edición de código, organización y planificación, entre otras, que se han requerido para la construcción del sistema.

Sistema operativos

- **Windows 10 Fall Creators Update:** para la elaboración del trabajo se ha elegido este sistema operativo debido a la plataforma de Realidad Mixta de Windows (término en inglés Windows Mixed Reality (WMR)), la cual permite desarrollar aplicaciones para el dispositivo de RA de Microsoft. Hasta el momento, la creación de este tipo de aplicaciones sólo puede ser realizada utilizando este sistema operativo.

¹https://developer.microsoft.com/en-us/windows/mixed-reality/install_the_tools

²<https://docs.microsoft.com/es-es/virtualization/hyper-v-on-windows/>

Frameworks

- **.NET 4.6:** es un framework de Microsoft que proporciona un conjunto de servicios a las aplicaciones en ejecución el cual consta de dos componentes principales: Common Language Runtime (CLR), qué es el motor de ejecución que controla las aplicaciones en ejecución, y la biblioteca de clases de .NET framework, las cuales están dedicadas al tratamiento de datos, herencias y patrones de diseño, entre otros. Algunos de los diferentes servicios que ofrece .NET a las aplicaciones en ejecución son:
 - Administración de la memoria de forma transparente para el desarrollador gracias a los servicios que proporciona CLR.
 - Sistema de tipos comunes que implica una mayor interoperabilidad entre aquellos lenguajes compatibles con .NET framework.
 - Biblioteca de clases extensa accesible al desarrollador para controlar operaciones de bajo nivel.
- **Netty³:** es un framework de aplicación asíncrono dirigido por eventos para el desarrollo rápido de aplicaciones en Java basadas en la arquitectura cliente-servidor. Esto es utilizado para simplificar la programación de servidores que utilizan *socket* TCP y UDP. Para el desarrollo del trabajo se ha empleado la versión 4.1.16.
- **Java Abstract Syntax Tree (AST):** el Árbol de Sintaxis Abstracta de Java es un framework que es utilizado por muchas de las herramientas del IDE de Eclipse, en las cuales se incluyen la refactorización, corrección y asistencia rápida. Este framework convierte el código fuente de un archivo Java en una estructura en árbol con el objetivo de facilitar su análisis y modificación de forma programática.

Plug-ins

- **HoloLensCameraStream⁴:** es un plug-in de Unity 3D para recuperar en tiempo real los fotogramas de la cámara de las gafas de RA de Microsoft. Esto permite utilizar la cámara del dispositivo para realizar tareas de visión por computador y aprendizaje automático, entre otras. Para el desarrollo del trabajo se ha utilizado la versión 0.3.0.
- **Eclipse Plug-in Development Environment (PDE):** el Entorno de Desarrollo de Plugins de Eclipse es un entorno para crear, desarrollar, probar, desplegar y construir plugins para dicha plataforma. Éste además proporciona componentes del framework de Java Open Services Gateway initiative (OSGI), el cual lo convierte en un entorno ideal para la programación de componentes.

PDE se divide en tres componentes principales:

³<https://github.com/netty/netty>

⁴<https://github.com/VulcanTechnologies/HoloLensCameraStream>

5. MÉTODO DE TRABAJO

- IU: proporciona editores, asistentes, vistas y otras herramientas para la creación de un entorno con características completas para el desarrollo de plug-ins y paquetes OSGI.
 - Application Programming Interface (API): herramienta que permite a los desarrolladores identificar incompatibilidades binaria entre dos versiones, números de versión de plug-in incorrectos, proporcionar documentación del código, etc.
 - Construcción: facilita la automatización del proceso de creación de plug-ins gracias a los scripts producidos en tiempo de desarrollo.
- **SWT designer:** es un plug-in desarrollado por Eclipse para la creación de interfaces gráficas. Éste hace uso de un editor visual en el cual se pueden crear interfaces sin la necesidad de escribir código Java, ya que éste es generado automáticamente a partir de la descripción visual proporcionada.

Bibliotecas

- **MediaFrameQrProcessing**⁵: es una biblioteca de código abierto destinado al reconocimiento de códigos QR basada en otra biblioteca llamada ZXing.Net⁶.
- **MixedRealityToolkit-Unity**⁷: es una colección de scripts y componentes con la intención de acelerar el desarrollo de aplicaciones dirigidas a los dispositivos de RA existentes en la actualidad haciendo uso de la versión 2017.2.1.4.. MixedRealityToolkit-Unity usa código basado en MixedRealityToolkit (MRTK), el cual proporciona una biblioteca de clases extensa para el reconocimiento de superficies, interacción con objetos (hologramas) y simulación de sonidos en un entorno 3D, entre otros.
- **QRGen**⁸: sencilla API para la generación de códigos QR en Java construida sobre ZXing⁹. Se ha hecho uso de la versión 2.3.0.
- **Gson**¹⁰: biblioteca de Java usada para convertir objetos de este lenguaje en su representación JSON. Para el desarrollo del proyecto se ha empleado la versión 2.8.2.
- **Apache Commons Lang**¹¹: es una biblioteca que proporciona una serie de ayudas para el desarrollo de aplicaciones en Java, particularmente en la manipulación de cadenas, concurrencia, reflexión y operaciones matemáticas de negocios, entre otras muchas contribuciones. Se ha empleado la versión 3.5.

⁵<https://github.com/mtaulty/QRCodes>

⁶<https://www.nuget.org/packages/ZXing.Net>

⁷<https://github.com/Microsoft.MixedRealityToolkit-Unity>

⁸<https://github.com/kenglxn/QRGen>

⁹<https://github.com/zxing/zxing>

¹⁰<https://github.com/google/gson>

¹¹<https://github.com/apache/commons-lang>

Lenguajes de programación

- **Java:** lenguaje de programación creado por Sun Microsystems el cual se caracteriza por ser de propósito general, concurrente y POO. La elección de Java se debe a que es el único lenguaje utilizado para la creación de plug-ins para la plataforma de desarrollo de Eclipse. Esto se debe al compilador interno que la herramienta incorpora junto a un módulo completo de los archivos fuente de Java.
- **C#:** lenguaje de POO desarrollado y estandarizado por Microsoft como parte de su plataforma .NET. La utilización de este lenguaje se debe al soporte que tiene el entorno Unity 3D para la generación de scripts.

Herramientas de desarrollo

- **Visual Studio 2017:** para la edición de código se ha utilizado dicho editor, ya que permite aprovechar las herramientas de depuración y profiling dedicadas a C#. Además, este editor permite desplegar las soluciones tanto en el dispositivo de RA como en el emulador, el cual es proporcionado por Microsoft para simular la visualización que realiza el dispositivo.
- **Unity 3D¹²:** es un motor de juegos que permite crear aplicaciones o videojuegos para las gafas de RA de Microsoft, entre un conjunto extenso de dispositivos. Este entorno ha permitido representar los modelos 3D, de tal modo que HoloLens lo visualice conforme se ha definido en el motor de juegos. Se ha hecho uso de la versión 2017.2.0f3.
- **Blender¹³:** entorno multiplataforma el cual está dedicado a la generación de modelos, animaciones, iluminación y renderizado, entre otros. Su utilización se debe a que Unity 3D nativamente importa archivos .blend, lo que facilita la utilización de modelos creados en el motor de juegos. Cabe destacar que todos los modelos 3D integrados en el sistema han sido realizados con esta herramienta. Para el desarrollo del proyecto se ha usado al versión 2.78.
- **Eclipse Neon:** es un plataforma de desarrollo diseñada para la creación de aplicaciones, la cual ha permitido la creación de plug-ins mediante PDE. El uso de esta plataforma ha sido debido a la interiorización que los alumnos que empiezan a programar.
- **Hercules¹⁴:** herramienta desarrollada para trabajar como un terminal el cual puede manejar puertos serie y protocolos UDP/IP y TCP/IP. La utilización de esta herramienta se debe a la facilidad de simular y probar comunicaciones TCP/IP entre cliente y servidor. Se ha hecho uso de la versión 3.2.8.
- **Mercurial:** sistema de control de versiones el cual ha permitido almacenar un repositorio con el código y documentación del proyecto para mantener un historial de cambios

¹²<https://unity3d.com/es>

¹³<https://www.blender.org/>

¹⁴<https://www.hw-group.com/software/hercules-setup-utility>

5. MÉTODO DE TRABAJO

y acceso centralizado. El servicio de repositorio utilizado ha sido proporcionado por BitBucket debido a que los proyectos pueden ser creados de manera privada, además de proporcionar un límite de hasta 2 GB por repositorio de manera gratuita. Se ha hecho uso de la versión 4.2.1.

- **Trello**¹⁵: sistema web para la administración de proyectos mediante el uso de tarjetas virtuales organizadas sobre un tablero virtual. Las tarjetas han representado los hitos correspondientes a los objetivos destacados en el capítulo de objetivos (véase § 2).

Documentación

- **LATeX**: Sistema de elaboración de textos utilizando la distribución TeX Live en su versión lanzada en 2018. El editor usado para la escritura del documento ha sido TeXStudio debido al resaltado sintáctico que proporciona junto a la corrección ortográfica interactiva.
- **Gimp**¹⁶: programa tanto de edición de imágenes digitales como de dibujo para la modificación de figuras insertadas en el documento. Se ha empleado la versión 2.10.2.
- **Draw.io**¹⁷: Aplicación web usada para la realización de los diagramas insertados en el documento.

¹⁵<https://trello.com/>

¹⁶<https://www.gimp.org/>

¹⁷<https://www.draw.io/>

Capítulo 6

Arquitectura

El presente capítulo está dedicado al análisis de los diferentes componentes que conforman la arquitectura del sistema expuesto en este trabajo. Esta sección se desglosa en diferentes puntos, en los cuales se cubrirán todos los aspectos que se han tenido en cuenta para el desarrollo del proyecto. En primer lugar, se presenta una descripción general donde se detallan las características más destacables del sistema. En segundo lugar, se plantea un enfoque descriptivo de los elementos que componen el proyecto, describiendo la funcionalidad que cada uno desempeña. En último lugar, se describe en detalle la problemática encontrada y se expone la solución desarrollada, explicando la implementación realizada y comentando algunas consideraciones de cada sistema del proyecto.

Antes de comenzar a detallar en profundidad los componentes más relevantes del sistema, es importante introducir de forma breve una pequeña descripción de la arquitectura propuesta. Los siguientes puntos muestran el objetivo que desempeñan los elementos principales de la plataforma.

- **Servidor:** este elemento representa el núcleo del sistema. A través de una arquitectura cliente-servidor, éste se encarga de generar los recursos que son enviados al cliente (plug-in de Eclipse). Los recursos son archivos que se visualizan en el dispositivo de RA a través de un procesamiento y análisis previo. Además, como se ha comentado en apartados anteriores, este elemento se integra en COLLECE 2.0, del cual se hace uso para facilitar las comunicaciones y la gestión de eventos.
- **Cliente:** este elemento es el encargado de recibir los recursos generados por el servidor. A partir de éstos, el cliente genera los correspondientes modelos gráficos y construye una representación unificada que es visualizada mediante el dispositivo de RA.

Ambos elementos están compuestos por varios sistemas. Cada sistema dota tanto al servidor como al cliente de funcionalidades específicas para realizar una serie de tareas. Por lo tanto, para ofrecer una mejor comprensión de los sistemas que operan en ambos elementos, se describe, a continuación, el cometido de cada uno de ellos. Además, se presenta en la Figura 6.1 un esquema que muestra la relación de todos los sistemas de la plataforma, agrupados por cliente y servidor.

6. ARQUITECTURA

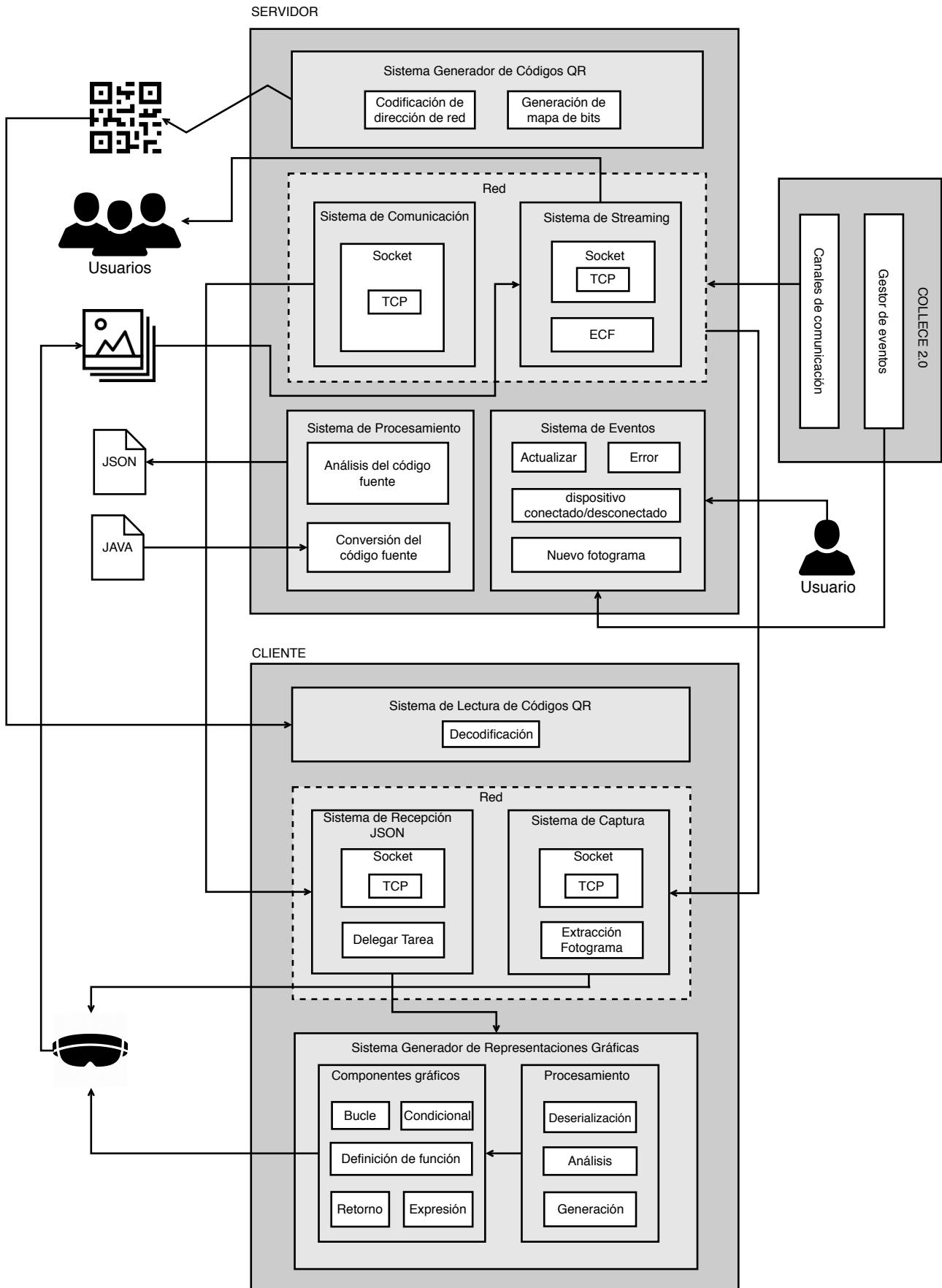


Figura 6.1: Esquema de la plataforma AsgAR

- **Sistema Generador de Códigos QR:** automatiza la creación de códigos QR con la información de red del servidor.
- **Sistema de Procesamiento:** analiza el código fuente de los métodos seleccionados por el usuario para generar archivos con un formato que facilite su posterior procesamiento.
- **Sistema de Eventos:** detecta los sucesos ocurridos en la vista del plug-in de Eclipse.
- **Sistema de Streaming:** muestra y comparte la visualización con todos los usuarios conectados al servidor que no disponen del dispositivo de RA.
- **Sistema de Comunicación:** gestiona el envío y recepción de información entre cliente y servidor.
- **Sistema de Lectura de Códigos QR:** escanea a través del dispositivo de RA los códigos QR generados por el plug-in de Eclipse.
- **Sistema de Recepción de archivos JSON:** recibe y gestiona los archivos en formato JSON generados por el *Sistema de Procesamiento*.
- **Sistema Generador de Representaciones Gráficas:** procesa los archivos enviados por el servidor para generar y construir la visualización a partir de los componentes gráficos diseñados.
- **Sistema de Captura:** recupera los fotogramas de la cámara del dispositivo de RA para enviarlos al *Sistema de Streaming*.

6.1 Servidor

Como se ha comentado anteriormente, el servidor se compone de un plug-in para Eclipse integrado en COLLECE 2.0, el cual ha sido implementado en Java utilizando PDE. Para permitir al usuario interactuar con la plataforma se ha creado una vista, la cual ha sido desarrollada mediante SWT Designer. A través de esta vista, el usuario puede conectarse con el dispositivo de RA, seleccionar un método para su posterior visualización, mostrar la transformación del código fuente de Java en el formato de intercambio de datos (JSON), y ver y retransmitir el contenido que muestran las gafas de RA.

El desarrollo de este componente se ha realizado mediante el uso de diferentes patrones de diseño. Esta técnica ha permitido facilitar la implementación de algunos componentes, además de reutilizar partes del diseño para afrontar la resolución de otros problemas. Los patrones utilizados son: «Observer», el cual notifica eventos a las vistas; «Modelos-Vista-Controlador», el cual permite separar la lógica de la parte de presentación (IU); «Factory Method», para crear instancias de objetos; «Singleton», para mantener un gestor de vistas de la interfaz; y «Visitor», para explotar el AST generado por Eclipse. Más adelante se explicará en detalle el porqué del uso de cada patrón, teniendo en cuenta la problemática a resolver y las ventajas que proporcionan al problema en cuestión.

6. ARQUITECTURA

Cada acción que el servidor debe realizar se ha traducido en un sistema que realiza ese cometido. En este sentido, éste cuenta con varios sistemas para abordar las tareas descritas al comienzo del apartado. Los sistemas que operan en el servidor son: *Sistema Generador de Códigos QR*, *Sistema de Comunicación*, *Sistema de Procesamiento*, *Sistema de Eventos* y *Sistema de Streaming*.

Tras identificar los sistemas que componen el servidor del sistema, a continuación se describe en profundidad el desarrollo de cada uno de ellos, detallando su funcionamiento y, justificando y explicando las decisiones de diseño adoptadas, así como detalles de la implementación.

6.1.1 Sistema Generador de Códigos QR

Este sistema es el encargado de establecer un puente entre servidor y cliente a través de códigos QR. Dada una dirección de red, éste genera una imagen almacenando la información en una matriz de puntos bidimensional.

El problema reside en establecer un mecanismo para comunicar el dispositivo de RA con el plug-in de Eclipse. Dado que ambos son sistemas independientes, uno debe conocer tanto la dirección como el puerto del otro proceso para establecer una conexión que permita el envío y recepción de información.

Teniendo en cuenta que ambos sistemas están en la misma red, el servidor podría realizar una búsqueda para encontrar la dirección del dispositivo requerido. Sin embargo, esta solución no es óptima, puesto que puede haber infinidad de dispositivos conectados, dificultando en este sentido la búsqueda.

Para almacenar la dirección y puerto del servidor, era necesario disponer de un elemento que permitiera, además de contener esta información, generarse de forma automática. Por ello, dada la versatilidad de los códigos QR, se decidió incluir esta tecnología para solventar el problema de la conexión. Gracias a este sistema, el establecimiento de comunicación es rápido y sencillo, puesto que el dispositivo obtiene instantáneamente la información almacenada en el código, sin que se vea comprometida la interacción entre el usuario, y el dispositivo al orientar simplemente la cabeza hacia el código QR generado.

La generación de códigos QR se ha llevado a cabo mediante la biblioteca *QRGen*, la cual está basada en *ZXing*.

La implementación desarrollada se basa en un método estático, el cual se utiliza en el controlador de la vista. Lo que se pretende es que cada vez que el plug-in se ejecute, éste recupere la dirección y puerto del servidor, pasando estos atributos como parámetros para construir un código que almacene la concatenación de ambos. La información almacenada es representada de la siguiente forma: ip:puerto1/puerto2.

Este elemento, como se puede observar, contiene dos puertos. El primer puerto es utilizado

para establecer la comunicación con el dispositivo ya compartir la información mediante mensajes JSON, mientras que el segundo es usado para recibir los fotogramas capturados por el dispositivo de RA.

En el Listado 6.1, se muestra la implementación desarrollada para la generación de códigos QR:

```

1  public static Image generateQRCode(String data, int width, int height, int color) {
2      ByteArrayOutputStream rawQRCodeStream = QRCode.from(data).withSize(width, height).
3          withColor(color, 0x00FFFFFF).stream();
4      ImageLoader loader = new ImageLoader();
5      ImageData imageData = loader.load(new ByteArrayInputStream(rawQRCodeStream.
6          toByteArray())[0];
7      Image qrCode = ResourceManager.getImage(ImageDescriptor.createFromImageData(
8          imageData));
9      return qrCode;
10 }
```

Listado 6.1: Generación código QR

Como se ha mencionado anteriormente, uno de los parámetros del método es el contenido que el código debe almacenar (parámetro data). Los demás, como sus propios nombres indican, se utilizan para indicar el ancho y altura, así como el color de las barras y puntos bidimensionales.

Esta método encapsula la información ip:puerto1/puerto2 junto al tamaño especificado en una colección de bytes. Seguidamente, esos bytes son transformados en información que describe la imagen para generar, posteriormente, el código QR asociado a esa descripción.

Finalmente, es importante destacar algunas consideraciones sobre la integración de códigos QR en el sistema, los cuales han permitido:

- Almacenar información de forma sencilla.
- Facilitar la conexión de otros dispositivos con el sistema, ya que la mayoría soporta la lectura de estos códigos.
- Agilizar el proceso de conexión facilitando la interacción entre el usuario y el dispositivo de RA.

6.1.2 Sistema de Procesamiento

Este sistema es el encargado de procesar el código fuente en Java para transformarlo en formato JSON.

La necesidad de analizar y procesar el código fuente se debe a instrucciones que dependen de un contexto, el cual debe ser recuperado. En el caso de sentencias if ... then ... else,

6. ARQUITECTURA

es necesario saber qué instrucciones están en una rama o en otra. Lo mismo ocurre con sentencias `switch ... case`, complicando la situación cuando aparece la sentencia `break` en algunas partes de la estructura.

Otro problema similar ocurre con los bucles, en los cuales es necesario saber qué conjunto de instrucciones forman la lógica del bucle y cuáles conforman la condición de éste.

Ignorar estas consideraciones supondría generar una visualización que no tenga nada que ver con el código fuente inicial que se pretende representar.

El sistema se ha desarrollado de forma transparente al usuario. Éste selecciona el método que quiere visualizar y el sistema se encarga de analizar y procesar el código fuente, dando lugar a un archivo en formato JSON con la información relevante y más simple de procesar, a la hora de generar las representaciones gráficas correspondientes en el cliente. La Figura 6.2 representa los paquetes y clases involucrados en el sistema.

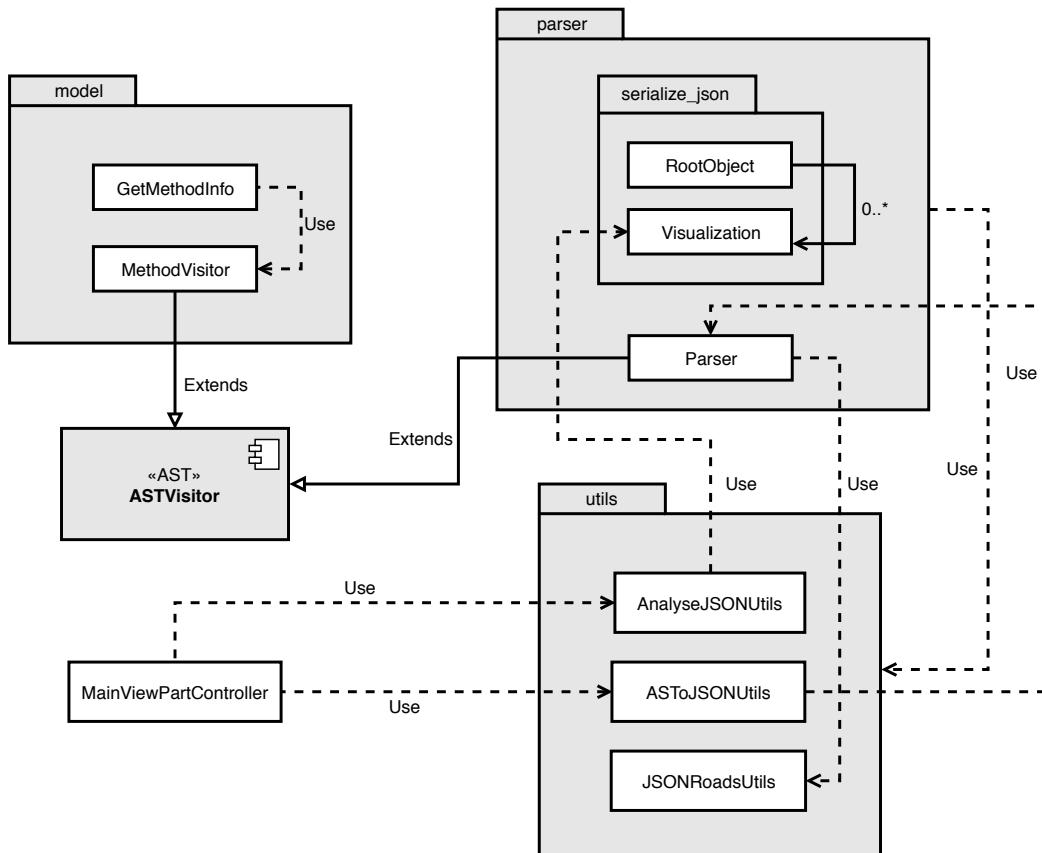


Figura 6.2: Elementos que componen el Sistema de Procesamiento

Este sistema consta de varias partes. Para detallar de un modo claro los componentes desarrollados, se ha seguido un enfoque en función del orden en el cual se utilizan los diferentes módulos. Teniendo esto cuenta, *model* sería el primer paquete del sistema a considerar.

GetMethodInfo, una de las clases de *model*, es la encargada de procesar el código fuente de los métodos que componen la clase que actualmente está editando el usuario en el editor

de Eclipse. Para ello se usa el framework de Eclipse AST, el cual permite analizar el archivo para poder obtener información sobre variables, métodos, expresiones y declaraciones, entre otros (ver Listado 6.2).

```

1 private static CompilationUnit parse(String source) {
2     ASTParser parser = ASTParser.newParser(AST.JLS8);
3     parser.setKind(ASTParser.K_COMPILATION_UNIT);
4     parser.setSource(source.toCharArray());
5     parser.setResolveBindings(true);
6     return (CompilationUnit) parser.createAST(null);
7 }
```

Listado 6.2: Procesamiento del código fuente de Java

El método recibe como parámetro el código de la clase activa, que es convertida a una cadena de caracteres para facilitar su análisis. Una vez *ASTParser* (clase proporcionada por AST) ha conseguido relacionar toda la información del código, se puede proceder a la posterior extracción de la información relevante. Para ello, se hace uso del método reflejado en el Listado 6.3, el cual almacena en una lista todos los métodos de la clase activa del editor, los cuales son incluidos en una lista desplegable de la vista del plug-in para facilitar al usuario elegir el método que desea visualizar.

```

1 private void createAST(String source) throws JavaModelException {
2     CompilationUnit parse = parse(source);
3     MethodVisitor visitor = new MethodVisitor();
4     parse.accept(visitor);
5     methodDeclaration = visitor.getMethods();
6 }
```

Listado 6.3: Agregación de los métodos visitados en la lista `methodDeclaration`

Este método hace uso de la clase *MethodVisitor* (incluida en el paquete *model*, el cual hace un análisis en el código para extraer la información de los métodos. Para poder realizar esta tarea, la clase anterior hereda de *ASTVisitor*, una clase de AST que permite explorar el código fuente de Java de forma jerárquica.

El paquete anterior hace uso de la lógica definida en el paquete *parser*, el cual se encarga única y exclusivamente del análisis del código. Para ello se hace uso de la clase *ASTVisitor*, que proporciona una serie de métodos para explorar el código fuente y definir los mecanismos de actuación al encontrar los distintos elementos del lenguaje. En la Figura 6.3 se muestran estos métodos, los cuales son sobreescritos en la clase *Parser* heredada de *ASTVisitor* para añadir la funcionalidad deseada.

6. ARQUITECTURA

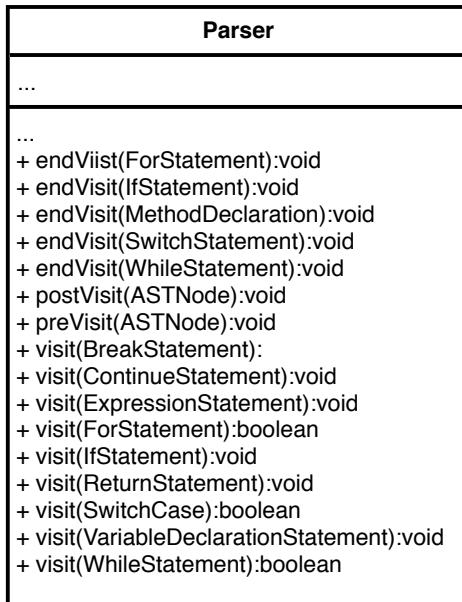


Figura 6.3: Métodos públicos de la clase *Parser*

Debido a que AST organiza el código en una estructura de árbol, la clase *Parser*, automáticamente, visita los diferentes métodos sobreescritos en función de la instrucción encontrada en el código fuente. El orden en el que se visitan estas funciones se muestra a continuación.

1. `preVisit(T node)`. Visita el nodo pasado por parámetro para desarrollar operaciones arbitrarias. Este método es invocado antes de que el método `visit` sea visitado.
2. `visit(T node)`. Visita el nodo pasado por parámetro para desarrollar operaciones arbitrarias. Si el método devuelve `true`, se visitarán, a continuación, los nodos hijos del nodo dado; sin embargo, si devuelve `false`, los nodos hijos del nodo dado no serán visitados.
3. hijos de la invocación que son procesados si `visit` devuelve `true`.
4. `endVisit(T node)`. Visita el nodo pasado por parámetro para desarrollar operaciones arbitrarias. Este método es invocado cuando todos los nodos hijos han sido visitados (o inmediatamente, si `visit` devolvió `false`).
5. `postVisit(T node)`. Visita el nodo pasado por parámetro para desarrollar operaciones arbitrarias. Este método es invocado después de que el método `endvisit` sea visitado.

Cada método de la clase *Parser* procesa el objeto recibido como argumento y actúa como función del tipo de objeto a analizar. Dependiendo del objeto recibido (definición de función, condición, bucle, retorno o expresión) la generación del archivo JSON varía de acuerdo a un esquema previamente definido (ver Listado 6.4). La conversión de cada instrucción se define en la clase *JSONRoadsUtils*, que contiene una serie de métodos que añaden al archivo JSON la correspondiente asociación directa de la instrucción en Java.

La estructura del archivo JSON se basa en varias claves y valores. En primer lugar, se especifica el tipo de elemento, que puede ser nodo o conector. Los nodos hacen referencia a las instrucciones anteriormente mencionadas, mientras que los conectores son enlazadores a estas instrucciones, es decir, conectan un nodo con otro para mantener una representación compacta y unificada. Dado que los conectores no aportan información, los siguientes puntos muestran el contenido que cada nodo puede almacenar:

- **Definición de función:** este nodo contiene el nombre de la función, una lista con sus argumentos y otra con el conjunto de instrucciones del bloque.
- **Condición:** representa la instrucción `if ... then ... else`. Este nodo contiene la expresión de la condición junto a dos listas que almacenan las instrucciones de cada rama de ejecución.
- **Bucle:** representa las instrucciones `for` y `while`. Este nodo contiene la expresión de la condición junto a una lista que almacena las instrucciones del bloque.
- **Sentencia de retorno:** este nodo contiene prácticamente la misma información que la *definición de función* (se excluye la lista de argumentos), añadiendo la expresión de retorno de la función (en el caso en que ésta devuelva un valor).
- **Expresión:** representa la ejecución de sentencias diferentes a las anteriores (p.ej., asignación de variables, operaciones, invocación a métodos, etc.).

```

1  // node
2  {
3      "type": "node"
4      "node": "FunctionIn"
5      "name": "",
6      "args": [],
7      "children": []
8  }

10 // connector
11 {
12     "type": "connector",
13     "connector": "Vertical"
14 }
```

Listado 6.4: Ejemplo de nodo y conector en formato JSON

Para crear el archivo JSON se hace uso de la clase *ASToJSONUtils* (incluida en el paquete *utils*), la cual genera una clase que simula el archivo activo en el editor, con la diferencia que ésta sólo contiene el método que se pretende procesar. Tras generar la clase, *ASToJSONUtils* utiliza la clase *Parser* para procesar la clase al completo y obtener una cadena en formato JSON. El Listado 6.5 muestra el método encargado de realizar este proceso.

6. ARQUITECTURA

```
1 public static String createJSON(String content) {
2     // extract code from active class in the editor
3     IEditorPart editorPart = PlatformUI.getWorkbench().getActiveWorkbenchWindow().
4         getActivePage().getActiveEditor();
5     String editorFilePath = PluginUtils.getPathFromEditorInput(editorPart.
6         getEditorInput()).toString();
7     String className = editorFilePath.substring(editorFilePath.lastIndexOf("/") + 1,
8         editorFilePath.length() - 5);
9
10    // create class to analyse method
11    String source = String.join("\n",
12        "public class " + className + " {",
13        content,
14        "}");
15
16    // compilate source code
17    ASTParser parser = ASTParser.newParser(AST.JLS8);
18    parser.setKind(ASTParser.K_COMPILATION_UNIT);
19    parser.setSource(source.toCharArray());
20    parser.setResolveBindings(true);
21    final CompilationUnit parse = (CompilationUnit) parser.createAST(null);
22
23    String beginningJSON = "{\"" + TYPE + "\":\""+ VISUALIZATION + "\", \""
24        + VISUALIZATION + "\": [";
25    Parser p = new Parser(beginningJSON);
26    parse.accept(p);

    // obtain JSON code
    return p.toJson();
}
```

Listado 6.5: Función que obtiene la cadean JSON

Aunque el archivo JSON se genera mediante este método, es necesario llevar a cabo un análisis del archivo, puesto que hay que incluir ciertos elementos que no se pueden añadir durante su creación. Esto ocurre con bucles o sentencias condicionales. En el caso de los bucles, las instrucciones que están dentro de este bloque se incluyen en el archivo JSON, sin embargo, los conectores que enlazan estas instrucciones para unificar la representación no son posibles de agregar. Del mismo modo ocurre con las sentencias condicionales, las cuales incluyen dos ramas de ejecución dependiendo de si se cumple la condición o no. Debido a esto, ambas ramas pueden contener diferentes número de instrucciones, dando lugar a ramas

con diferentes longitudes. Además, el problema no sólo reside en este aspecto; cuando tenemos sentencias condicionales anidadas, bucles anidados o la mezcla de ambos, los modelos se superponen unos con otros, dificultando comprender la representación generada.

Para solventar este problema se ha hecho uso de las clases *RootObject* y *Visualization* (incluidas en el paquete *Serialize_json*, el cual a su vez está en el paquete *parser*, las cuales describen los atributos y valores del archivo JSON. Esto permite convertir los objetos del archivo a una lista de listas para facilitar su procesamiento. Esta conversión se conoce comúnmente como proceso de *serialización*, el cual se ha llevado a cabo mediante la biblioteca *Gson* de Google. La Figura 6.4 muestra los atributos y métodos de cada clase junto a sus relaciones.

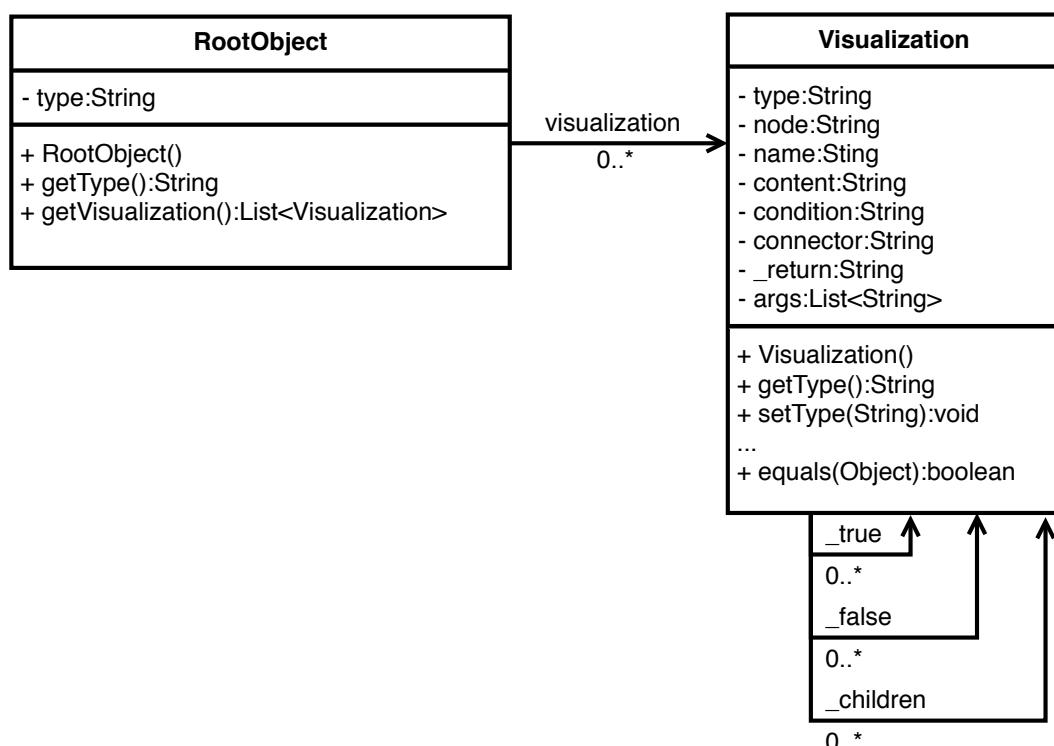


Figura 6.4: Diagrama de la clase *RootObject* y *Visualization*

La lista que genera la clase *RootObject* se analiza a través de dos métodos incluidos en la clase *AnalyseJSONUtils*, entre los cuales se reparten dos tareas: el primero se encarga expandir el modelo cuando las carreteras se superponen unas con otras, mientras que el segundo completa las zonas vacías que no se pudieron añadir durante el análisis del código fuente de Java. Para ello, estos métodos recorren de forma recursiva una lista para encontrar las partes que necesitan ser reconstruidas o completadas.

El primer método de la clase *AnalyseJSONUtils* (clase incluida en el paquete *utils*) se muestra en el Listado 6.6. Como se puede apreciar, la lógica principal se divide en dos condiciones. La primera parte se encarga de añadir conectores en la instrucción bucle, teniendo en

6. ARQUITECTURA

cuenta el número de sentencias condicionales o bucles anidados dentro del bloque. De forma similar ocurre con las condiciones. En el caso de que haya sentencias condicionales anidadas o bucles dentro de una condición, es necesario añadir conectores para expandir la instrucción e impedir la superposición de estos elementos.

La lista generada por *RootObject* se recorre mediante un bucle, sin embargo, esta parte se ha obviado en el listado debido a las dimensiones del código. Para conocer los elementos que se deben incluir en la instrucción bucle se hace uso de la variable *result*, la cual lleva la cuenta del número de sentencias condicionales. En el caso en que estas sentencias estén dentro del bucle, se añaden conectores en función del valor de la variable. Esto se consigue mediante la llamada al método *AddHorizontalConnectors*.

El proceso para completar las sentencias condicionales es similar al procedimiento anterior. Dado que el método devuelve el número de sentencias condicionales, las variables *right* y *left* almacenan el número de condiciones dentro de cada rama. De este modo, se crean tantos conectores como número de sentencias condicionales tenga cada una. Estos conectores se añaden mediante el método *AddLRConnectors*.

```
1 private static int CompleteBranchIfAndLoop(List<Visualization> children, boolean amIIInIf) {
2     ...
3     if (visualization.getChildren() != null) {
4         if (visualization.getNode().equals(JSONRoadsUtils.LOOP_IN)) {
5             result = CompleteBranchIfAndLoop(visualization.getChildren(),
6                 amIIInIf);
7             if (result > 0) {
8                 AddHorizontalConnectors(children.get(i).getChildren(),
9                     result + 1);
10                result = 0;
11            }
12        }
13        if (visualization.getType().equals(JSONRoadsUtils.NODE)) {
14            if (visualization.getNode().equals(JSONRoadsUtils.IF_IN)) {
15                result = 0;
16                int right = CompleteBranchIfAndLoop(visualization.getTrue(), true);
17                int left = CompleteBranchIfAndLoop(visualization.getFalse(), true);
18                if (right > 0) AddLRConnectors(children, i, right);
19                else if (left > 0) AddLRConnectors(children, i, left);
20                result += 1;
21            }
22        }
23    ...
24    return result;
}
```

Listado 6.6: Método que completa las ramas de la estructura de bucle y condición

En cuanto al segundo método de la clase *ASToJSONUtils* (ver Listado 6.7), éste también se divide en dos partes. La primera se encarga de añadir el lado del bucle que permite volver a ejecutar las instrucciones del bloque. La segunda parte tiene la tarea de completar las ramas de las sentencias condicionales para obtener la misma longitud. Para conocer el número de elementos a colocar en la estructura del bucle, es necesario saber cuántas instrucciones contiene el bloque. De este modo, el bucle queda unificado representando la metáfora de la rotonda. Esto se consigue mediante la llamada al método *AddDoubleConnectors*, el cual añade tantos conectores como número de elementos tenga la lista.

Para completar las ramas de las sentencias condicionales se realiza un proceso similar al anterior. Dado que el método devuelve el número de elementos dentro de un bloque, las variables *left* y *right* almacenan el número de instrucciones dentro de cada rama. De este modo, sabiendo qué rama contiene un mayor número de elementos, se puede completar aquella que cuenta con un número menor, equilibrando en este sentido ambas ramas. Este proceso se realiza mediante el método *AddVerticalConnectors*.

```

1  private static int CompleteLoopAndIf(List<Visualization> children) {
2      int result = 0;
3      for (int i = 0; i < children.size(); i++) {
4          result += 1;
5          Visualization visualization = children.get(i);
6          if (visualization.getChildren() != null) {
7              if (visualization.getNode().equals(JSONRoadsUtils.LOOP_IN)) {
8                  result = CompleteLoopAndIf(visualization.getChildren());
9                  AddDoubleConnectors(children, visualization, i, 0);
10             }
11         }
12         if (visualization.getType().equals(JSONRoadsUtils.NODE)) {
13             if (visualization.getNode().equals(JSONRoadsUtils.IF_IN)) {
14                 int right = CompleteLoopAndIf(visualization.getTrue()) - 2;
15                 int left = CompleteLoopAndIf(visualization.getFalse()) - 2;
16                 AddVerticalConnectors(children, i, right, left);
17                 result += right > left ? right : left;
18             }
19         }
20     }
21     return result;
22 }
```

Listado 6.7: Función que completa la estructura de bucle y condición

En definitiva, la integración de AST en el sistema ha aportado un conjunto de ventajas para el desarrollo del proyecto, las cuales son:

- Utilización de una estructura de datos en árbol.
- Facilidad del procesamiento del código Java.

6. ARQUITECTURA

- Obtención de información de cada instrucción.
- Organización del código del sistema.

6.1.3 Sistema de Eventos

Este sistema es el encargado de detectar los eventos que ocurren durante la ejecución para llevar a cabo las acciones producidas por cada uno de ellos.

Las aplicaciones que cuentan con una interfaz gráfica están diseñadas para que el usuario pueda realizar todo tipo de acciones. Por lo tanto, el flujo de ejecución escapa al control del programador. En cierta manera se puede decir que el usuario es quien controla y decide el orden de ejecución de los procesos.

La programación orientada a eventos resuelve la problemática anteriormente comentada, puesto que permite a los distintos elementos del entorno reaccionar a las acciones que realiza el usuario. Inspirado en este paradigma nace el sistema discutido en esta sección, el cual se encarga de recoger los eventos y realizar las acciones pertinentes. La Figura 6.5 representa los elementos que intervienen en el funcionamiento del sistema en cuestión.

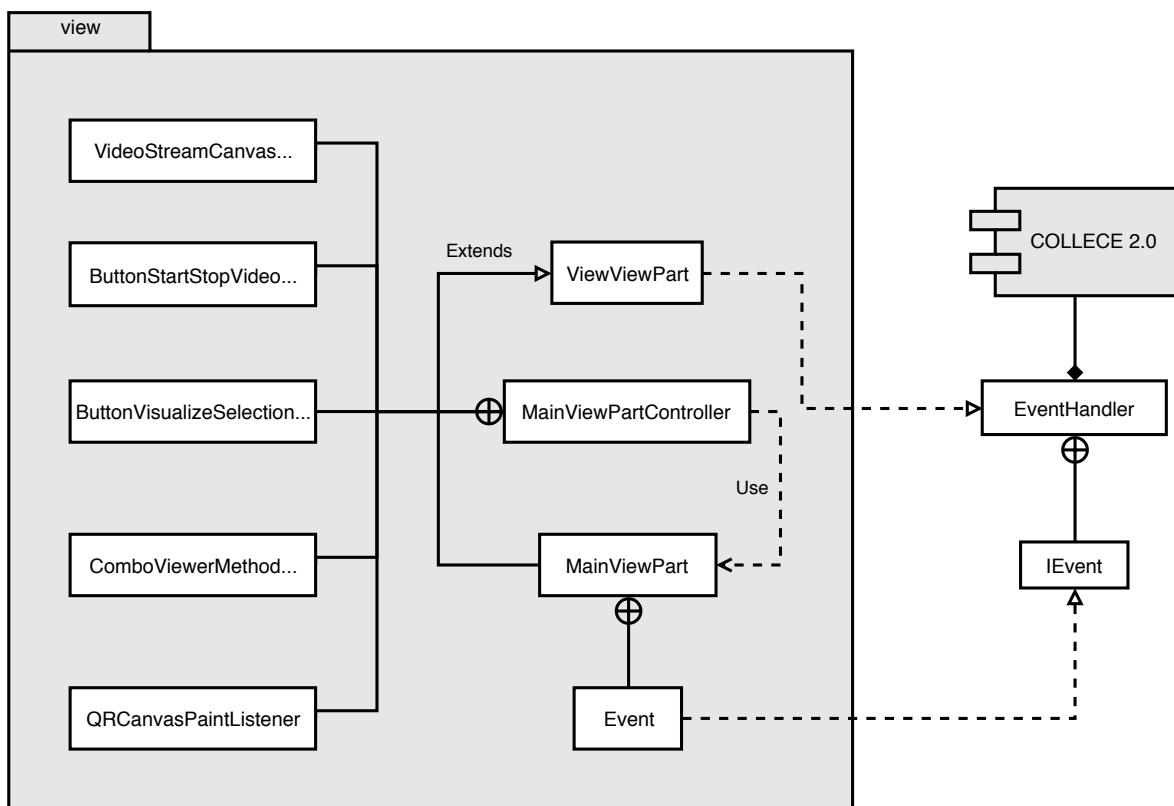


Figura 6.5: Elementos que componen el Sistema de Eventos

Como se aprecia en el diagrama anterior, este sistema hace uso del manejador de eventos de COLLECE 2.0 para ejecutar métodos disparados en el hilo principal. La clase que usa este manejador es *ViewViewPart* (clase que hace referencia a la entidad *Modelo* del patrón «MVC», el cual a su vez se basa en el patrón «Observer»), la cual ordena al controlador de la

vista que ejecute los métodos específicos cuando se dispara un evento. Éstos se recogen en la clase *Event* (esta clase hace referencia al patrón «State») de la clase *MainViewPart* (clase que hace referencia a la entidad *Vista* del patrón «MVC»), la cual implementa la interfaz *IEvent* (ver Listado 6.8).

```

1 public enum Event implements IEvent {
2     ON_UPDATE_METHODS,
3     ON_DEVICE_CONNECT,
4     ON_DEVICE_NOT_CONNECT,
5     ON_NEW_SOCKET_FRAME,
6     ON_ERROR
7 }
```

Listado 6.8: Eventos de la vista *MainViewPart*

Cada evento dispara un método producido por una acción del usuario en la vista de Eclipse. A continuación, se describe de forma breve el cometido de cada evento en el sistema en cuestión:

- **ON_UPDATE_METHODS**: actualiza los métodos en una lista desplegable cuando cambia la clase activa en el editor.
- **ON_DEVICE_CONNECT**: habilita los botones de la vista cuando se ha detectado que el dispositivo de RA se ha conectado con el servidor.
- **ON_DEVICE_NOT_CONNECT**: deshabilita los botones cuando el dispositivo se ha desconectado del servidor.
- **ON_NEW_SOCKET_FRAME**: se encarga de ejecutar los métodos que componen los fotogramas recibidos del dispositivo.
- **ON_ERROR**: se encarga de gestionar todos los errores que ocurren mientras la vista está en ejecución.

Aparte de estos eventos, el sistema cuenta con una serie de *listeners*, que son eventos asociados a *widgets*. Estos elementos aparecen a la izquierda del diagrama, los cuales representan los eventos asociados a dos canvas, una lista desplegable y un conjunto de botones.

Estos eventos se encuentran en la clase *MainViewPartController* (esta clase hace referencia a la entidad *Controlador* del patrón «MVC»), donde son registrados para disparar la acción definida. El Listado 6.9 refleja el código que registra estos eventos.

```

1 viewPart.getQRCanvas().addPaintListener(
2     new QRCanvasPaintListener());
3 viewPart.getButtonVisualizeMethod().addSelectionListener(
```

6. ARQUITECTURA

```
4     new ButtonVisualizeSelectionAdapter());
5     viewPart.getVideoStreamCanvas().addPaintListener(
6         new VideoStreamCanvasPaintListener());
7     viewPart.getButtonStartStopVideoStream().addSelectionListener(
8         new ButtonStartStopVideoStreamSelectionAdapter());
9
10    viewPart.getComboViewerMethodSelector().setContentProvider(
11        new ArrayContentProvider());
12    viewPart.getComboViewerMethodSelector().setLabelProvider(
13        new ComboViewerMethodSelectorLabelProvider());
```

Listado 6.9: Registro de eventos en *MainViewPartController*

Cada evento implementa un *listener* teniendo en cuenta la lógica que se quiere desarrollar. Para ello, se implementa una interfaz y se sobreescreiben dos clases, las cuales son: *PaintListener*, *SelectionAdapter* y *LabelProvider*. La interfaz *PaintListener* se utiliza para pintar los canvas utilizados en la vista. La lógica asociada a estos elementos se ejecuta cuando ocurre un evento de dibujado en la interfaz gráfica del usuario. La clase *SelectionAdapter* se usa para definir la lógica asociada a los botones que se encuentran en la vista cuando se pulsa sobre ellos. El método asociado a esta clase se dispara cuando un evento de selección ocurre en los botones. Finalmente, la clase *LabelProvider* permite cambiar el texto de las etiquetas asociadas a los elementos de la lista desplegable. Este evento se dispara cuando la lista desplegable es seleccionada. El Listado 6.10 muestra una parte de la implementación de la interfaz de cada *listener*.

```
1 class QRCanvasPaintListener implements PaintListener {
2     @Override
3     public void paintControl(PaintEvent e) {
4         ...
5     }
6 }
7 class ButtonVisualizeSelectionAdapter extends SelectionAdapter {
8     @Override
9     public void widgetSelected(SelectionEvent e) {
10         ...
11     }
12 }
13 class VideoStreamCanvasPaintListener implements PaintListener {
14     @Override
15     public void paintControl(PaintEvent pe) {
16         ...
17     }
18 }
19 class ButtonStartStopVideoStreamSelectionAdapter extends SelectionAdapter {
20     @Override
21     public void widgetSelected(SelectionEvent e) {
22         ...
23 }
```

```

23     }
24 }
25 class ComboViewerMethodSelectorLabelProvider extends LabelProvider {
26     @Override
27     public Image getImage(Object element) {
28         ...
29     }
30     @Override
31     public String getText(Object element) {
32         ...
33     }
34 }
```

Listado 6.10: Implementación de interfaz y clases

En definitiva, este sistema ha permitido gestionar de manera óptima las acciones llevadas a cabo por el usuario. Debido a la utilización de la programación orientada a eventos, se ha conseguido:

- Separar los datos del diseño.
- Reutilizar el código.
- Obtener un código más simple.
- Facilitar el comprensión del sistema.
- Facilitar la expansión de funcionalidades.

6.1.4 Sistema de Streaming

Este sistema es el encargado de recibir y procesar los fotogramas de la cámara del dispositivo de RA con el fin de retransmitir su contenido a todos los miembros conectados al servidor. La tecnología de red utilizada para realizar este cometido es *sockets TCP*, dado que permite asegurar que los paquetes no se perderán durante el proceso de envío y recepción, garantizando el orden de llegada de los mismos.

Debido a que la visualización que observa el usuario que lleva el dispositivo no es visible desde el exterior por otro usuario, Microsoft proporciona un portal (en el Anexo D se puede encontrar información sobre esto) que permite ver este contenido a través de una retransmisión en tiempo real. No obstante, el acceso se restringe a un usuario, dificultando la utilización de este dispositivo en las aulas.

Debido a esto se desarrolló el sistema discutido en esta sección, ya que permite mostrar el contenido que el dispositivo crea. De este modo, todos los alumnos desde su propio equipo pueden observar las representaciones generadas, facilitando en este sentido el transcurso de una clase. En la Figura 6.6 se muestran los paquetes y clases involucrados en el sistema en cuestión.

6. ARQUITECTURA

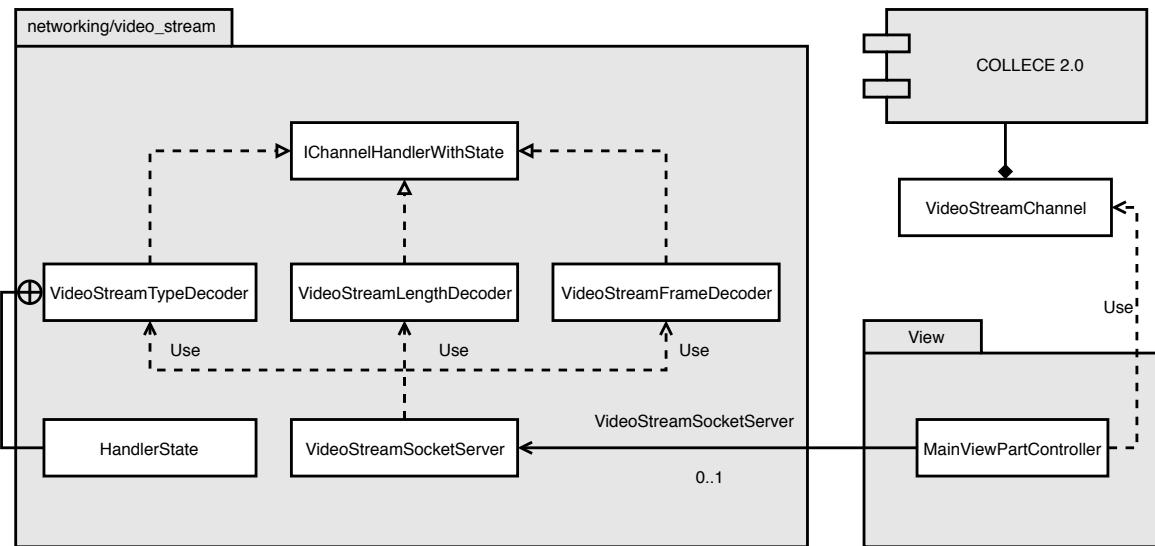


Figura 6.6: Elementos que componen el Sistema de Streaming

El planteamiento inicial para el desarrollo de este sistema se basó en la utilización de *sockets* UDP, debido a la rapidez que éstos plantean frente a TCP. Sin embargo, los problemas que presentaba esta tecnología eran más mayores que las ventajas que ofrecía.

Uno de ellos era el **orden** en el que llegan los paquetes. El protocolo de transporte UDP no garantiza que los mensajes recibidos lleguen conforme se enviaron, lo que supone un problema, puesto que los paquetes que conforman los fotogramas se deben reconstruir en el servidor en el mismo orden en el que se definieron para componer el mapa de bits que forma la imagen.

Otro problema era respecto a la **confiabilidad** que proporciona el protocolo UDP. Éste no asegura que los mensajes sean recibidos por el servidor, lo que puede suponer la pérdida de paquetes y, por tanto, la corrupción de la imagen al recomponerla en el servidor.

Finalmente, UDP es un protocolo no orientado a conexión, lo que impide mantener un estado de la misma y, por tanto, tener un **control de errores**.

Estos problemas han dificultado la utilización de *sockets* UDP, ya que, aunque éste proporciona una eficiencia considerable, TCP proporciona todas las características anteriores, haciendo de este un sistema robusto y fiable a costa de una pequeña pérdida de rendimiento.

En relación con el diseño del sistema, se ha seguido un modelo de estados por los que pasa la comunicación de red para recibir los fotogramas de la cámara del dispositivo de RA. Para ello, se ha creado la clase *HandlerState* (clase que hace referencia al patrón «State»), la cual está incluida en *VideoStreamTypeDecoder*, del paquete *video_stream*. Esta clase es la encargada de verificar si el tipo recibido es aquel definido para comenzar el proceso de retransmisión. En el Listado 6.11 se muestran los estados anteriormente mencionados:

```

1 class HandlerState {
2     static final short INITIAL = 0, TYPE RECEIVED = 1, LENGTH RECEIVED = 2,
3         FILE READING = 3, FILE RECEIVED = 4;
}

```

Listado 6.11: Estados de la clase *HandlerState*

Cada estado representa la situación en la que se encuentra el fotograma recibido, los cuales se controlan en diferentes clases por las que pasa la comunicación. A continuación, se describe el cometido de estos estados en el sistema en cuestión:

- **INITIAL**: como su nombre indica, describe el estado inicial del proceso de retransmisión.
- **TYPE RECEIVED**: representa el tipo con el cual comienza el proceso de retransmisión. De esta forma, si el tipo no concuerda con el definido para este sistema (simplemente una constante numérica de valor 25 elegida de forma aleatoria), se ignora el mensaje. Este estado es utilizado en la clase *VideoStreamTypeDecoder*
- **LENGTH RECEIVED**: representa la situación de recibir la longitud de un archivo (en este caso un fotograma). Este estado se utiliza en la clase *VideoStreamLengthDecoder*.
- **FILE READING**: describe el estado utilizado en la clase *VideoStreamFrameHandler*, que lee el archivo que está siendo recibido.
- **FILE RECEIVED**: lanza un evento para iniciar el pintado de los fotogramas en el canvas de la vista.

Este intercambio de estados se realiza a través de una tubería proporcionada por la biblioteca *netty*, donde el mensaje pasa a través de las clases anteriores, las cuales componen el canal (ver Figura 6.7). Estas clases implementan la interfaz *IChannelHandlerState*, que proporciona un método que mediante polimorfismo reinicia el estado de cada una a su estado inicial.

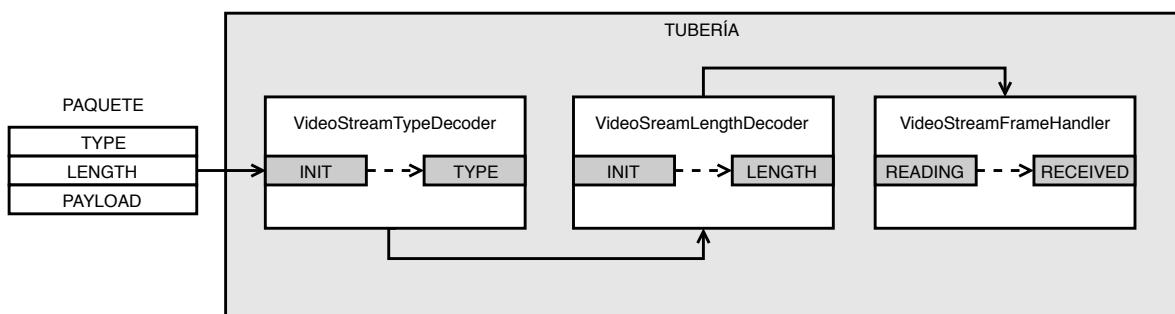


Figura 6.7: Estructura de la tubería para el cambio de estados

6. ARQUITECTURA

La asociación de las clases a la tubería se realiza en *VideStreamSocketServer*, que se encarga a la vez de obtener el puerto donde se ejecuta el proceso de retransmisión. Esta clase es instanciada en *MainViewPartController*, del paquete *view*, para realizar la recogida del puerto justo cuando la vista es cargada.

Cada clase representada en el diagrama anterior extrae una parte del paquete que se recibe, cuyo objetivo es reducir la carga de procesamiento de los fotogramas que el cliente envía. En el caso de *VideoStreamTypeDecoder*, ésta se encarga de recoger el tipo del paquete, el cual indica por la tubería a *VideoStreamLengthDecoder* que espere a recibir el longitud del archivo (tamaño del fotograma en bytes). Una vez se ha recibido el tamaño del archivo, la tubería notifica a *VideoStreamFrameHandler* que se encargue de leer y reconstruir el fotograma a partir de los bytes recibidos y almacenados en un buffer temporal.

Este proceso de ejecución a través de los estados comienza cuando el usuario pulsa sobre el botón de la interfaz *start streaming*. En ese momento, se envía un mensaje al dispositivo de RA que inicia la lógica de captura y envío de fotogramas al servidor, el cual los procesa para pintarlos en un canvas. Después de realizar el proceso intermedio de cambio de estados, un evento es lanzado el cual especifica que el fotograma está listo para ser procesado y pintado.

Además de lo anteriormente comentado, la retransmisión puede ejecutarse en varios equipos si éstos están conectados a una sesión. Una sesión es una característica de COLLECE 2.0, en la cual varios miembros pueden estar conectados para realizar tareas de forma colaborativa. Por lo tanto, utilizando la clase *VideoStreamChannel* de COLLECE 2.0, la retransmisión es multicast, es decir, se envía a todos los miembros conectados mediante los canales que el sistema soporta (ver Listado 6.12). Cabe destacar que COLLECE 2.0 utiliza ECF para realizar esta tarea.

```
1 public void processFrameFromVideoStream(byte[] bytes, boolean relayToChannels) {
2     // creation and construction the image in the canvas for retransmission
3     ...
4     Member currentLoggedMember = StateManager.INSTANCE.getClientLoggedMember();
5     if (currentLoggedMember != null && relayToChannels)
6         try {
7             VideoStreamChannel videoStreamChannel =
8                 DatashareClientManager.INSTANCE.getChannel(
9                     VideoStreamChannel.class,
10                    StateManager.INSTANCE.getClientConnectedSession());
11                 videoStreamChannel.sendVideoStreamPacket(bytes);
12             } catch (ECFException e) LogUtils.error(e);
13         }
14 }
```

Listado 6.12: Función que procesa y retransmite fotogramas a los miembros conectados al servidor

Finalmente, este sistema ha logrado solventar el problema de retransmitir el contenido tridimensional en un sólo equipo. Gracias a la integración de COLLECE 2.0, el sistema se ha beneficiado de:

- Utilizar los canales de comunicación para enviar el *streaming* a todos los miembros conectados a una misma sesión.
- Reducir la carga de procesamiento del *socket* al delegar el envío de fotogramas al resto de clientes utilizando los canales proporcionados por ECF.
- Aprovechar la infraestructura de sesiones para permitir al resto de usuarios conectados ver la visualización desde el punto de vista del usuario que lleva el dispositivo.

6.1.5 Sistema de Comunicación

Este sistema es el encargado de gestionar la comunicación entre el cliente y el servidor utilizando *sockets TCP*, proporcionados por la biblioteca *netty*. A través de este sistema ambos procesos intercambian mensajes para realizar una serie de acciones.

El servidor es el encargado de generar los recursos, puesto que es el elemento con el cual el usuario interactúa. Debido a esto, dichos recursos necesitan ser enviados de algún modo al dispositivo de RA, para que éste los procese y genere las representaciones gráficas pertinentes.

A partir de este problema nace el sistema discutido en esta sección, el cual se basa en el modelo cliente-servidor para permitir comunicar los dispositivos involucrados en la estructura distribuida. En la Figura 6.8 se muestran los elementos involucrados en el sistema.

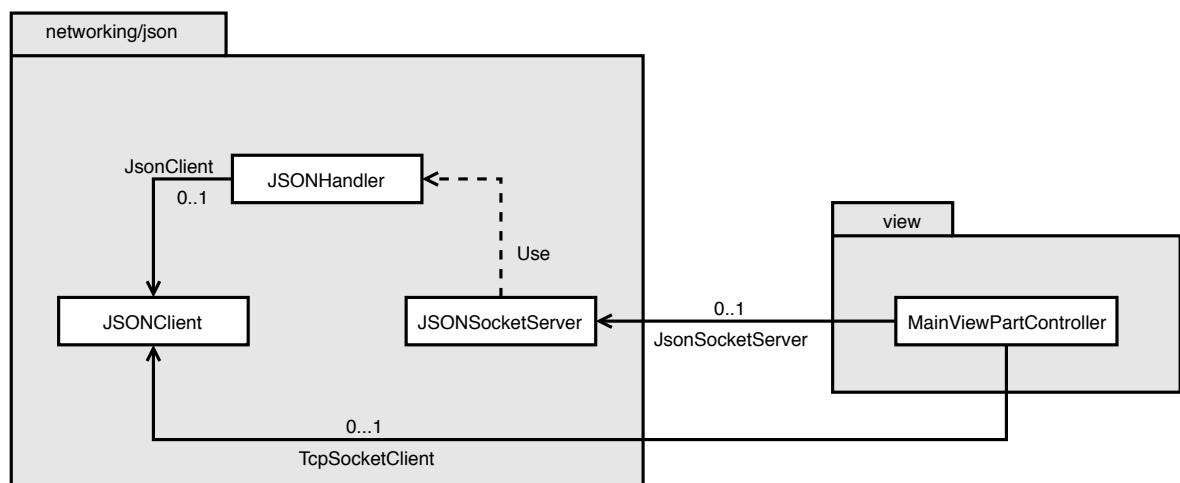


Figura 6.8: Elementos que componen el Sistema de Comunicación

La comunicación entre servidor y cliente comienza con la clase *JSONSocketServer*, la cual se encarga de abrir un puerto para establecer una conexión mediante *sockets TCP*. Como se aprecia en el diagrama, la clase *MainViewPartController* hace uso de esta clase para abrir el canal de comunicación al cargarse la vista. Cabe destacar que la utilización de *sockets TCP*

6. ARQUITECTURA

se debe a que aseguran una conexión adecuada y fiable, garantizando la recepción y envío de paquetes.

Tras establecer un puente donde intercambiar información entre servidor y cliente, era necesario crear un manejador para gestionar el envío y recibo de recursos con el cliente. Debido a esto se creó la clase *JSONHandler*, la cual proporciona funciones tanto para recibir como para comprobar el estado de la conexión con el cliente.

En cuanto al envío de información, *JSONClient* es la clase encargada para esta tarea. Esta clase cuenta con diferentes métodos para especificar al cliente si comenzar con el proceso de generación de representaciones gráficas, iniciar, o detener el proceso de retransmisión. El Listado 6.13 muestra el código en formato JSON que se envía al cliente para determinar el proceso que debe iniciar:

```
1  {"type":"stream_video", "stream_video":"true/false"};  
2  {"type":"visualization", "visualization": [...]};
```

Listado 6.13: Envío de mensaje con la especificación del proceso a realizar

Estos mensajes se envían a través de varias funciones, cada una de las cuales se asocia a un proceso distinto. De este modo, todas realizan la misma acción, enviando contenidos completamente distintos. Esto ha servido tanto para separar las diferentes funcionalidades con las que cuenta el sistema como para facilitar la extensibilidad del mismo. El Listado 6.14 muestra el código asociado a cada método:

```
1  public void sendJSON(String json) {  
2      send(json);  
3  }  
4  public void sendStartStreaming() {  
5      send(START_VIDEO_STREAM_MESSAGE);  
6  }  
7  public void sendStopStreaming() {  
8      send(STOP_VIDEO_STREAM_MESSAGE);  
9  }  
10 private void send(String json) {  
11     // Send length  
12     int jsonLength = json.getBytes(CharsetUtil.UTF_8).length;  
13     context.write(Unpooled.copyInt(jsonLength));  
14     // Send JSON  
15     context.write(Unpooled.copiedBuffer(json, CharsetUtil.UTF_8));  
16     context.flush();  
17 }
```

Listado 6.14: Envío de mensaje con la especificación del proceso a realizar

Como se aprecia en el listado, los tres primeras métodos usan *send*, el cual realiza el proceso de envío a través del canal. Este método primero envía la longitud del archivo JSON y, seguidamente, envía el contenido del mismo. De este modo, el cliente conoce el tamaño en bytes del archivo que tiene que procesar, evitando así perder información en la tarea de lectura. Esto se debe a que el uso de sockets no garantiza que la información enviada se envíe a un único paquete de red, pudiendo dividirse en varios paquetes y, por tanto, teniendo que leer varias veces del socket del cliente.

En definitiva, este sistema ha permitido gestionar las tareas relacionadas con el envío y recepción de información, además de establecer el canal de comunicación. La arquitectura elegida ha beneficiado al sistema de:

- Distribuir las funciones y responsabilidades entre dos elementos independientes, permitiendo reemplazar, reparar o actualizar uno de ellos sin verse el otro afectado por tales cambios.
- Facilitar, por separado, la extensibilidad de las capacidades de cliente y servidor.

6.1.6 Patrones de diseño

A lo largo del desarrollo del servidor se han utilizado diferentes patrones de diseño que han dado lugar a mejorar el diseño del mismo, permitiendo reutilizar partes del código y facilitando la extensión de funcionalidades. Uno de ellos

Uno de ellos es el patrón «Singleton», el cual se suele utilizar cuando es necesario tener una única instancia de un determinado tipo de objeto. Dado que el servidor cuenta con un gestor de vistas, la utilización de este diseño ha permitido garantizar que sólo exista una única instancia del objeto en cuestión, además de permitir el acceso global a dicha instancia. De este modo, el gestor ha podido operar en varias clases sin la necesidad de tener múltiples instancias del mismo objeto.

Otro de los patrones de diseño utilizados en el desarrollo del servidor, ha sido el patrón «Modelo-Vista-Controlador», el cual se utiliza para aislar el dominio de aplicación, es decir, la lógica, de la parte de presentación (IU). Dado que el servidor se compone de un vista en Eclipse, la utilización de este patrón ha permitido gestionar los eventos producidos en la estructura de las interfaces para representar la información apropiada en la vista. De este modo, el problema de que un usuario pueda realizar diferentes acciones sobre la interfaz gráfica se ha solventado mediante la implementación de este patrón.

Siguiendo en este contexto, el patrón «Observer» ha sido otra de las soluciones adoptadas para el desarrollo de la vista en Eclipse. Este patrón se utiliza para que un objeto pueda notificar y/o actualizar el estado de otros automáticamente. En este patrón se basa la entidad *Modelo* del diseño anterior, el cual se encarga de notificar los eventos producidos a las vistas.

6. ARQUITECTURA

Otro patrón de diseño a destacar, es el patrón «State», el cual se utiliza cuando el comportamiento de un objeto cambia dependiendo del estado del mismo. Sobre este patrón se basan el *Sistema de comunicación* y el *Sistema de Retransmisión*, ya que permite variar su comportamiento en función del estado interno que almacenan en un determinado momento.

Por otra parte, el patrón «Visitor» se utiliza como un mecanismo para realizar diferentes operaciones sobre una jerarquía de objetos, de forma que añadir nuevas operaciones no haga necesario cambiar las clases de los objetos sobre los que se realizan las operaciones. Este patrón ha sido utilizado por la clase *Parser*, el cual hereda todas las operaciones que proporciona *ASTVisitor* para manejar la estructura en árbol del código fuente de Java generado por AST.

Finalmente, el último patrón de diseño a destacar dentro de esta sección, es el patrón «Factory-Method», el cual se basa en la definición de una interfaz para crear instancias de objetos y permitir a las subclases decidir cómo crear dichas instancias implementando un método determinado. La utilización de este patrón ha permitido determinar a las subclases de las vistas qué métodos implementar, resolviendo en este sentido el problema de crear diferentes instancias de un mismo objeto. Este patrón ha sido utilizado por las clases *View-ViewPart* y *ViewViewPartController*

6.2 Cliente

Este elemento, que compone una parte de la arquitectura cliente-servidor, es una aplicación desarrollada para el dispositivo de RA Microsoft HoloLens. La implementación se ha llevado a cabo mediante el entorno Unity 3D usando el lenguaje C#.

Esta aplicación cuenta con nuevas formas de interacción mediante gestos, las cuales ofrecen una experiencia de usuario mucho más natural.

Además, ésta cuenta con la capacidad de reconocer el entorno, permitiendo identificar mesas, paredes, suelo y techo, entre otras muchas superficies. Esta característica está integrada en el dispositivo, sin embargo, mediante la utilización de la biblioteca MRTK, este proceso de escaneado puede ser utilizado programáticamente durante la ejecución de la aplicación. De este modo, el dispositivo conoce los cambios producidos en el entorno, dotando a la aplicación de una mayor libertad y realismo a la hora de alinear y ubicar las representaciones gráficas.

De forma similar al servidor, el desarrollo de esta aplicación se ha realizado mediante el uso de varios patrones de diseño. Los patrones utilizados en esta aplicación son: «Singleton», para mantener objetos vivos entre cambios de escena; «Observer», para notificar eventos a los manejadores de objetos; y «State», para describir la situación en la que se encuentra un objeto. Más adelante se explicará en detalle el uso de estos patrones en el sistema, teniendo en cuenta la problemática a resolver y las ventajas que proporcionan respecto al problema.

Aparte de esto, el cliente cuenta con una serie de sistemas, los cuales se encargan de realizar una serie de tareas específicas. Los sistemas utilizados por este proceso son: *Sistema de Lectura de Códigos QR*, *Sistema de Recepción de archivos JSON*, *Sistema Generador de Representaciones Gráficas* y *Sistema de Captura*.

A continuación, se describe en profundidad el desarrollo de cada uno de los sistemas anteriores, detallando su funcionamiento y, justificando y explicando las decisiones de diseño adoptadas en la implementación.

6.2.1 Sistema de Lectura de Códigos QR

Este sistema es el encargado de escanear códigos QR mediante el dispositivo de RA para obtener la dirección y puerto del servidor.

El problema consiste en que el dispositivo no cuenta con un mecanismo para obtener la información almacenada en la imagen. De este modo, la comunicación no puede establecerse, impidiendo en este sentido la generación de representaciones gráficas.

Debido a esto, se ha desarrollado un sistema capaz de extraer los fotogramas que genera la cámara del dispositivo. Cada fotograma se analiza para obtener la información almacenada tras la matriz de puntos bidimensionales.

El desarrollo de este sistema se ha llevado a cabo mediante la biblioteca *MediaFrameQr-Processing*, la cual proporciona una serie de ventajas para procesar este tipo de imágenes.

Para extraer los fotogramas de la cámara se ha hecho uso de la clase *ZXingQrCodeScanner*, en la cual se define un método que analiza estas imágenes para comprobar que se está escaneando un código QR. De ser así, ésta obtiene la información almacenada tras los puntos bidimensionales (ver Listado 6.15).

```

1  public static async void ScanFirstCameraForQrCode(
2      Action<Result> resultCallback,
3      TimeSpan timeout)
4  {
5      Result result = null;
6      var mediaFrameSourceFinder = new MediaFrameSourceFinder();
7      // Group frames in colour
8      var populated = await mediaFrameSourceFinder.PopulateAsync(
9          MediaFrameSourceFinder.ColorVideoPreviewFilter,
10         MediaFrameSourceFinder.FirstOrDefault);
11     if (populated)
12     {
13         // Obtain the device camera resource
14         var videoCaptureDevice =
15             await VideoCaptureDeviceFinder.FindFirstOrDefaultAsync();
```

6. ARQUITECTURA

```
16     if (videoCaptureDevice != null)
17     {
18         // Get frames of the camera and run
19         // ZXing to process them
20         var frameProcessor = new QrCaptureFrameProcessor(
21             mediaFrameSourceFinder,
22             videoCaptureDevice,
23             MediaEncodingSubtypes.Bgra8);
24         frameProcessor.SetVideoDeviceControllerInitialiser(
25             vd => vd.Focus.TrySetAuto(true));
26         // Process frames during the specified time
27         await frameProcessor.ProcessFramesAsync(timeout);
28         // see result obtained
29         result = frameProcessor.QrZxingResult;
30     }
31 }
32 // execute custom method
33 resultCallback(result);
34 }
```

Listado 6.15: Método que procesa los fotogramas de la cámara del dispositivo de RA

Este método, tras conseguir el resultado de escanear un código QR, ejecuta una acción en la cual se puede gestionar la información obtenida. Cabe destacar que a este método se le puede asignar un periodo de tiempo para el proceso de escaneado, el cual se define como `timeout` en los argumentos. El Listado 6.16 muestra la acción implementada cuando un código QR es reconocido.

```
1 public void OnScan()
2 {
3     ...
4     ZXingQRCodeScanner.ScanFirstCameraForQRCode(result =>
5     {
6         UnityEngine.WSA.Application.InvokeOnAppThread(() =>
7
8             if (result != null && !scanned)
9             {
10                 scanned = true;
11                 capture AudioSource.Play();
12                 StartCoroutine(WaitWhileAudioIsPlaying(result));
13             }
14         }, false);
15 }
```

```

15     }, null);
16     ...
17 }
```

Listado 6.16: Método que recupera el valor de un código QR

El método `onScan` reproduce un audio que indica al usuario que el código QR ha sido reconocido. Tras esto, se ejecuta `StartCoroutine(WaitWhileAudioIsPlaying(result))`, un método que se invoca para proceder a establecer la conexión con el dispositivo de RA. Cabe indicar que el proceso de escaneado se realiza mientras que un código QR no sea reconocido, ya que la variable `timeout` está a `null`. Después de esto, se libera el recurso de la cámara para, más tarde, obtener los fotogramas que se utilizarán en el proceso de retransmisión.

En definitiva, este sistema ha permitido gestionar la lectura de códigos QR con el dispositivo de RA. La utilización de la biblioteca *MediaFrameQrProcessing* ha facilitado:

- Obtener el recurso de la cámara del dispositivo de RA.
- Extraer los fotogramas de la cámara.
- Procesar los fotogramas extraídos.
- Obtener el resultado tras escanear un código QR.

6.2.2 Sistema de Recepción de archivos JSON

Este sistema es el encargado de recibir los archivos en formato JSON que el *Sistema de Comunicación* envía al cliente.

El servidor genera archivos de intercambio de datos, los cuales se obtienen de analizar y procesar el código fuente de Java. Estos recursos son enviados al cliente, el cual debe encargarse de procesarlos para componer las representaciones gráficas correspondientes y construir la visualización del método. Sin embargo, el tamaño de estos archivo puede variar, dificultando en este sentido el procesamiento del mismo. Este problema dificulta la creación de las representaciones gráficas, ya que si un recurso es demasiado grande, su información se puede perder.

Debido a esto, el sistema expuesto en esta sección define una estructura para solventar este problema. Para ello, el cliente primero recupera el tamaño del archivo, con el objetivo de utilizarlo posteriormente como referencia de lectura. De este modo, el cliente conoce los bytes que debe leer del archivo a procesar. La Figura 6.9 muestra los elementos que intervienen en el funcionamiento del sistema en cuestión.

Este sistema utiliza *sockets* TCP para establecer una conexión con el servidor, dado que permite asegurar una conexión fiable. La clase con la que se establece la conexión entre cliente y servidor es *TcpNetworkClient*. Esta clase ha sido desarrollada con el fin de mantener una estructura escalable a la hora de añadir nuevas funcionalidades al sistema. Ésta proporciona

6. ARQUITECTURA

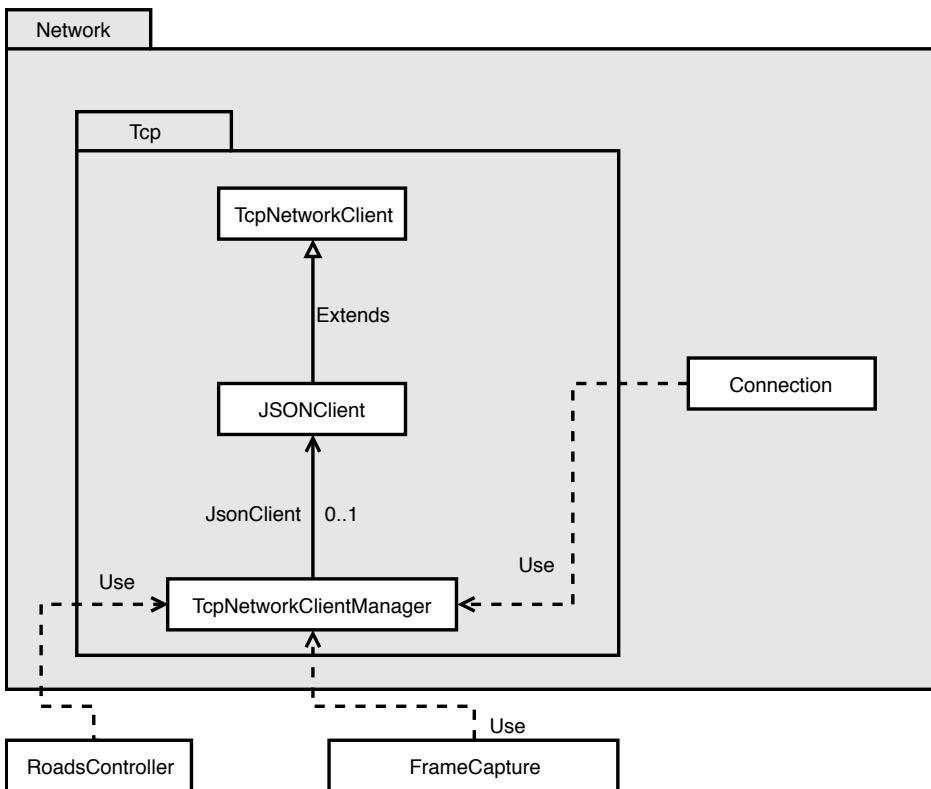


Figura 6.9: Elementos que componen el Sistema de Recepción de archivos JSON

un conjunto de funciones, las cuales permiten enviar mensajes al servidor, recibir mensajes del servidor, y conectar el cliente con la dirección extraída del código QR. Estas funciones se realizan a través de *sockets*, utilizando la biblioteca *Windows.Networking.Socket*. Una característica importante a destacar de esta clase, es la utilización de eventos, los cuales permiten notificar a las clases hijas que un mensaje ha sido recibido.

JSONClient hereda de la clase anterior, la cual obtiene la conexión con el servidor. De este modo, cada vez que el servidor envía un mensaje, el evento es capturado para comenzar a leer el contenido de dicho mensaje. Como se comentó anteriormente, el servidor envía en bytes el tamaño del archivo con el fin de leer el contenido completo del mismo. El método que realiza esta acción se muestra en el Listado 6.17:

```

1 protected override async void ReceiveAsync()
2 {
3     while (receiveFlag)
4     {
5         try
6         {
7             // read length of payload
8             int payloadLength = await ReadPayloadLength();
9             if (payloadLength <= 0)
10                 continue;
  
```

```

12         // read the content of the message
13         byte[] payload = await ReadPayload(payloadLength);

15         if (payload.Length <= 0)
16             continue;

18         OnReceiveInvoke(payload);
19     }
20     catch (Exception ex)
21     {
22         throw new Exception(ex.Message);
23     }
24 }
25 }
```

Listado 6.17: Lectura asíncrona de mensajes

Para mantener la conexión durante toda la ejecución de la aplicación se hace uso de la clase *TcpNetworkClientManager*, la cual se basa en el patrón «Singleton» para mantener vivo el objeto entre los cambios de escena (los objetos en Unity 3D se destruyen cuando se realiza el cambio de una escena a otra). Además, ésta gestiona los eventos producidos por la clase *JSONClient*, los cuales son encapsulados en varios métodos para mejorar la comprensión del código (aquí es donde se hace uso del patrón «Observer»).

Por último, es importante destacar las ventajas que ha proporcionado la programación orientada a eventos para gestionar el recibo de mensajes enviados desde servidor. Los beneficios obtenidos son:

- Separación de datos del diseño.
- Reutilización de código.
- Obtención de un código más simple y comprensible.
- Adquisición de un modo sencillo el mensaje enviado por el servidor.
- Facilidad de extensión de funcionalidades.

6.2.3 Sistema de Captura

Este sistema es el encargado de recoger los fotogramas de la cámara del dispositivo de RA para llevar a cabo la retransmisión del contenido que éste genera.

El problema consiste en que las representaciones gráficas creadas por el dispositivo sólo pueden ser visualizadas por el usuario que lo utiliza. De este modo, el número de personas

6. ARQUITECTURA

que pueden beneficiarse de las ayudas que este proyecto brinda en torno al aprendizaje de la programación se reduce.

Debido a esto, la solución desarrollada ha consistido en la creación de un sistema capaz de recoger los fotogramas de la cámara del dispositivo para enviarlos al servidor y mostrarlos sobre una vista. En la Figura 6.10 se reflejan los elementos que intervienen en el funcionamiento del sistema:

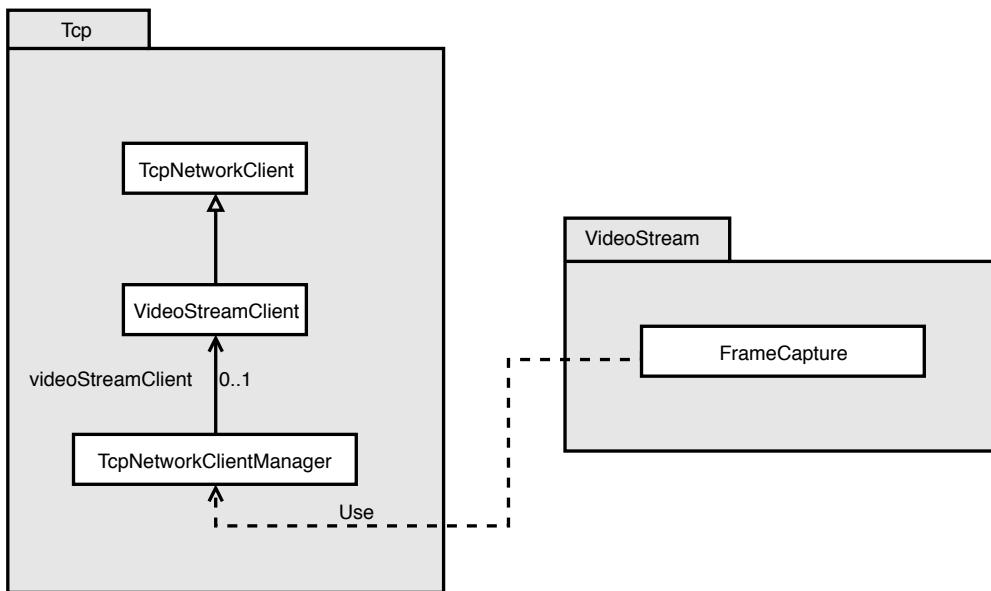


Figura 6.10: Elementos que componen el Sistema de Captura

Este sistema, al igual que el anterior, utiliza *sockets* TCP, dado que permite asegurar que los paquetes no se perderán durante el proceso de envío (ver § 6.1.4 para conocer el planteamiento que ha desembocado en esta solución). La clase con la que se establece la conexión entre cliente y servidor es *VideoStreamClient*, la cual hereda de *TcpNetworkClient* para llevar a cabo el proceso de retransmisión.

Para realizar la recogida de fotogramas de la cámara, se ha utilizado la biblioteca *HoloLensCameraStream*, la cual es usada en la clase *FrameCapture*. Esta clase se encarga de gestionar el estado en el que se encuentra cada fotograma durante la retransmisión. Cabe destacar que estos estados son los mismos que se utilizan en el *Sistema de Retransmisión*, codificados con los mismos valores para ser utilizados como acuse de recibo cuando el servidor responda a los envíos del cliente.

```
1 public enum VideoStreamStates : Int16
2 {
3     NONE = -3,
4     ACQUIRING_FRAME = -2,
5     READY = -1,
6     FRAME_ACQUIRED = 0,
```

```

7     TYPE_RECEIVED = 1,
8     LENGTH_RECEIVED = 2,
9     FILE_RECEIVED = 4
10    }

```

Listado 6.18: Estados que especifican la situación en la que se encuentra un fotograma

Esta clase, a través de *TcpNetworkClientManager*, recibe los archivo JSON que contienen las especificación de comenzar la retransmisión. Cuando este mensaje llega, el estado **NONE** cambia a **ACQUIRING_FRAME**, el cual describe la acción de adquirir un nuevo fotograma.

Después de este paso, los siguientes mensajes que se reciban contendrán el valor asociado a cada estado, los cuales se gestionan a través de una maquina de estados (ver Figura 6.11). Tanto el *Sistema de Retransmisión* como éste intercambiarán mensajes hasta alcanzar el estado **FILE RECEIVED**, el cual especifica que el fotograma ha sido procesado correctamente. Este proceso se repite continuamente hasta que llegue un archivo en formato JSON que especifique detener la retransmisión.

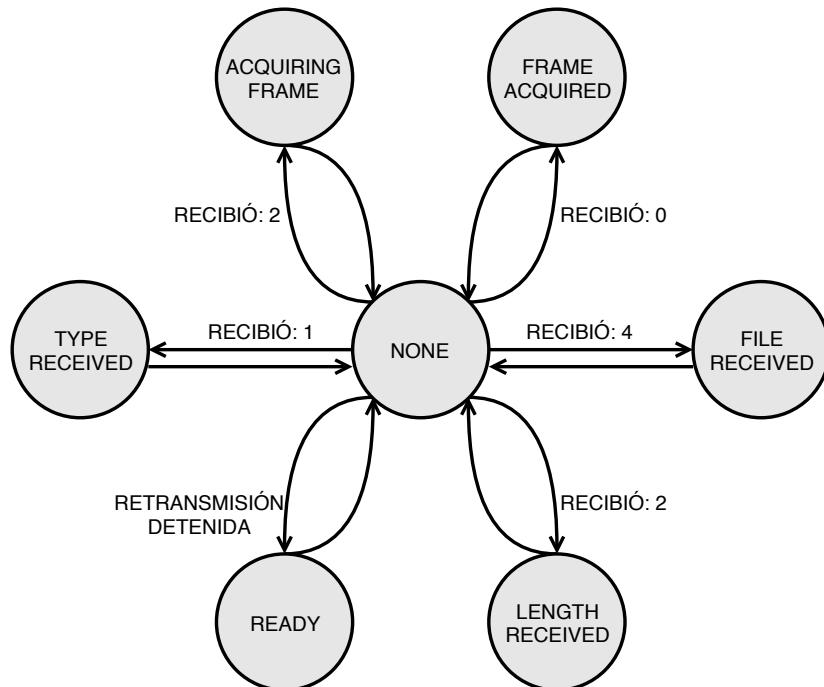


Figura 6.11: Máquina de estados para procesar los fotogramas

En definitiva, este sistema ha permitido llevar a cabo la retransmisión del contenido tridimensional generado por el dispositivo de RA. La utilización de la biblioteca *HoloLensCameraStream* ha facilitado:

- Obtener el recurso de la cámara del dispositivo de RA.
- Obtener en tiempo real los fotogramas de la cámara.

6.2.4 Sistema Generador de Representaciones Gráficas

Este sistema es el encargado de generar mediante archivos en formato JSON las correspondientes representaciones gráficas que son visualizadas a través del dispositivo de RA.

El problema reside en que el usuario puede visualizar un programa o algoritmo arbitrario en tiempo de ejecución, lo que impide tener un modelo previo que represente su estructura.

Debido a esto, se ha desarrollado el sistema discutido en esta sección, el cual permite construir representaciones gráficas tras el procesamiento de archivos JSON generados por el servidor. La Figura 6.12 representa los elementos que intervienen en el funcionamiento del sistema.

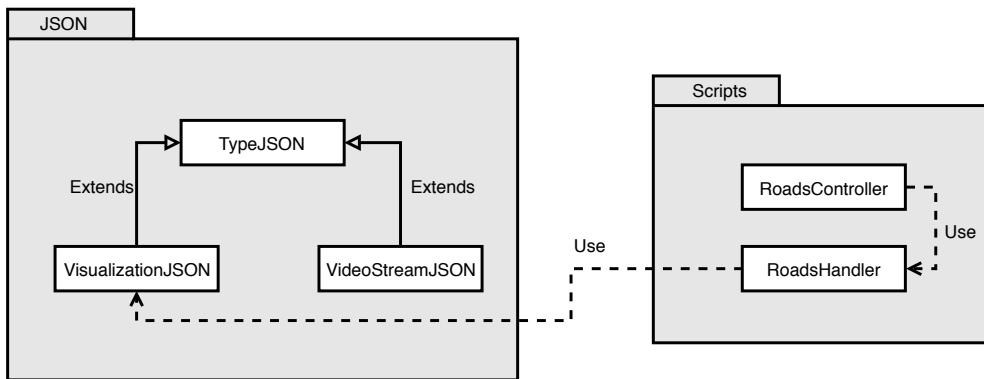


Figura 6.12: Elementos que componen el Sistema Generador de Representaciones Gráficas

Junto al desarrollo de este sistema, se han generado un conjunto de modelos tridimensionales, los cuales representan: definición de una función, instrucción de condición, bucles, sentencia de retorno y expresiones, entre otros. A continuación, se presentan los modelos desarrollados junto a una breve descripción de cada uno. Cabe destacar que el diseño de estos modelos se muestra en el capítulo resultados (ver § 7).

- **Node_FunctionIn:** representa el comienzo de definición de una función. Este modelo muestra el nombre de la función y los pasados a la misma.
- **Node_FunctionOut:** representa el fin de una función. Este modelo refleja la variable que devuelve una función (en el caso de que se devuelva algo).
- **Node_LoopIn:** representa el comienzo de un bucle. Este modelo muestra la condición de parada.
- **Node_LoopOut:** representa el fin de un bucle.
- **Node_IfIn:** representa el comienzo de una sentencia `if ... then ... else`. Este modelo muestra la condición que se evalúa en la sentencia.
- **Node_IfOut:** representa el fin de la sentencia `if ... then ... else`.
- **Node_Expression:** representa la transcripción literal de declaración evaluada.

- **Connector_IfInL:** conector que sirve de enlace a la rama izquierda de *Node_IfIn*.
- **Connector_IfInLE:** conector que sirve de enlace a las instrucciones dentro de la rama izquierda de *Node_IfIn*.
- **Connector_Node_IfInR:** conector que sirve de enlace a la rama derecha de *Node_-IfIn*.
- **Connector_IfInRE:** conector que sirve de enlace a las instrucciones dentro de la rama derecha de *Node_IfIn*.
- **Connector_If_OutL:** conector que sirve de enlace a la rama izquierda de *Node_IfOut*.
- **Connector_If_OutLE:** conector que sirve de enlace a las instrucciones dentro de la rama izquierda de *Node_IfOut*.
- **Connector_If_OutR:** conector que sirve de enlace a la rama derecha de *Node_IfOut*.
- **Connector_If_OutRE:** conector que sirve de enlace a las instrucciones dentro de la rama derecha de *Node_IfOut*.
- **Connector_HorizontalIn:** conector que sirve de enlace a *Loop_IfIn*.
- **Connector_HorizontalOut:** conector que sirve de enlace a *Loop_IfOut*.
- **Connector_Vertical:** conector que sirve para completar las ramas de la sentencia if
... then ... else.
- **Connector_Double:** conector que sirve para completar la instrucción bucle.

Estos modelos son generados en tiempo de ejecución mediante la clase *RoadsHandler* (incluida en el paquete *Scripts*), la cual contiene un conjunto de métodos que añaden las instrucciones anteriores. Mediante un procesamiento del archivo JSON, estas instrucciones se generan dando lugar a una representación gráfica unificada. El método que se encarga de este procesamiento se muestra en el Listado 6.19.

```

1  private GameObject Backward(List<Visualization> children, GameObject _object, GameObject
2   parent)
3   {
4     GameObject _gameObject = null;
5     int size = children.Count;
6     for (int i = 0; i < size; i++)
7     {
8       Visualization visualization = children[i];
9       // Associate _object to _gameObject when the last one is not null
10      if (_gameObject != null)
11        _object = _gameObject;
12
13      _gameObject = IdentifyGameObjectToReturn(visualization, _object, parent);

```

6. ARQUITECTURA

```
14         // Check if children list exists
15         if (visualization.children != null)
16             _gameObject = Backward(visualization.children, _gameObject,
17                                   _gameObject);
18
19         // Check if true list exists
20         if (visualization.@true != null)
21             Backward(visualization.@true, _gameObject, _gameObject);
22
23         // Check if false list exists
24         if (visualization.@false != null)
25             _gameObject = Backward(visualization.@false, _gameObject,
26                                   _gameObject);
27     }
28
29     // By existing the children list, we get the parent GameObject
30     // to paint the roads that follow the loop
31     if (parent.name.Equals(ConstantsRoads.LOOP_IN))
32         _gameObject = parent;
33
34     return _gameObject;
35 }
```

Listado 6.19: Método recursivo para generar las representaciones gráficas

Este método, para generar las representaciones gráficas, hace uso de la clase *VisualizationJSON* (incluida en el paquete *JSON*), la cual deserializa el contenido del archivo JSON en una lista de listas que facilita el procesamiento del mismo. A través de un recorrido de la lista, se obtienen los elementos para identificar el modelo que se tiene que generar. Para ello, se usa el método *IdentifyGameObjectToReturn*, el cual invoca el método que permite añadir el objeto identificado.

Para añadir los objetos y que éstos sean colocados en su posición correspondiente, se han diseñado unas marcas en los objetos, las cuales son: *joiner_in*, *joiner_out* y *joiner_fill*. El primero sirve para identificar el comienzo de un objeto; el segundo permite conocer la posición final del mismo; y el último permite rellenar las partes incompletas de los modelos. Aparte de lo anterior, esto permite que nuevos modelos puedan incluirse en el proyecto, evitando reorganizar partes del código. La Figura 6.13 muestra un ejemplo de estas marcas.

Concluyendo en cuanto al proceso de creación de carreteras, el método que permite generar las representaciones gráficas hace uso de la clase *RoadsController* (incluida en el paquete *Scripts*), la cual recibe el archivo JSON enviado por el servidor. Cuando un mensaje llega,



Figura 6.13: Ejemplo que refleja las marcas de la representación 3D de la sentencia bucle: *joiner_in* (verde), *joiner_out* (azul), *joiner_fill* (rojo)

primero se verifica que contenga el tipo `visualization`, lo que conlleva comenzar con el procesamiento del archivo. Seguidamente, se crean las carreteras mediante la llamada al método `createRoads`, de la clase `RoadsHandler`. Este método se encarga de crear por separado el comienzo y fin de función, para, posteriormente, invocar al método `Backwards`, que tiene la tarea de crear las representaciones asociadas a las instrucciones dentro del bloque del método. Una vez las carreteras son creadas, se añaden unos atributos para que el usuario puede llevar a cabo las acciones de escalar, rotar o trasladar (ver Listado 6.20):

```

1  private void TcpNetworkClientManager_OnReceiveJSON(byte[] data)
2  {
3      string message = Encoding.UTF8.GetString(data);
4
5      // Convert JSON into an object to retrieve its type
6      var typeObject = JsonConvert.DeserializeObject<TypeJSON>(message);
7
8      // If the type of JSON file is not visualization, we discard this task
9      if (!typeObject.type.Equals(ConstantsJSON.VISUALIZATION))
10         return;
11
12     // Convert JSON into a real type
13     var visualizationObject =
14         JsonConvert.DeserializeObject<VisualizationJSON>(message);
15
16     // Create roads
17     GameObject roads = roadsHandler.CreateRoads(visualizationObject.visualization);

```

6. ARQUITECTURA

```
19 // Add to rads collider and plane  
20 // Align them in front of the user  
21 ...  
22 }
```

Listado 6.20: Método que gestiona la creación de las representaciones gráficas

Finalmente y, opuesto al proceso de creación de carreteras, es importante mostrar la creación de unas marcas, diferentes a las anteriormente explicadas, para tener una mayor visión de aquellas condiciones o expresiones que están incompletas por el espacio que ocupan. Para ello, se ha propuesto desplegar papeles encima de los modelos clicables (señales y cajas) para reflejar la totalidad de la condición o expresión. Para realizar este cometido, se han diseñado dos marcas, las cuales son: *paper_in* y *paper_out*. La primera sirve para identificar la posición original en la que se encuentra el papel, mientras que la segunda permite colocar éste por encima de la caja o señal. En la Figura 6.14 se refleja la posición de estas marcas.

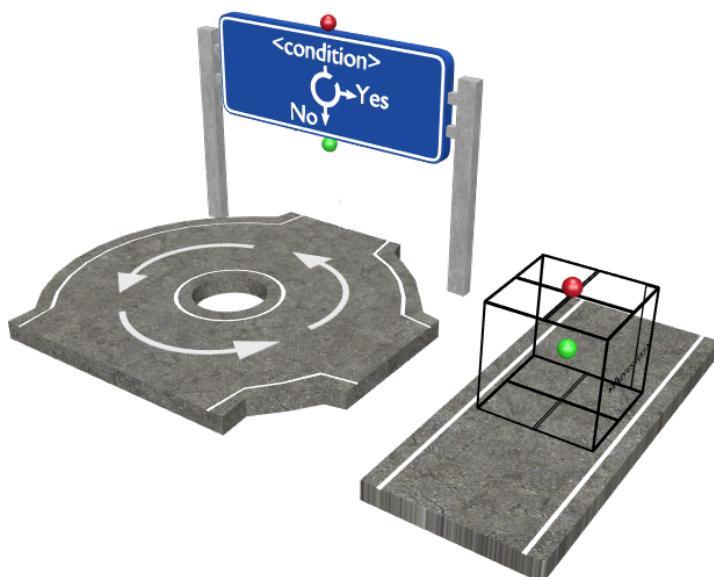


Figura 6.14: Ejemplo que refleja las marcas para el papel de la representación 3D de la sentencia bucle y expresión: *paper_in* (verde) y *paper_out* (rojo)

En definitiva, este sistema ha conseguido solventar el problema de construir representaciones gráficas de forma eficiente en tiempo de ejecución. Las ventajas que éste ha proporcionado a la plataforma son:

- Generación automática de representaciones gráficas.
- Posibilidad de representar un conjunto de programas simultáneamente.

6.2.5 Otras características

El cliente, aparte de los sistemas anteriormente mencionados, cuenta con una serie de características que mejoran la experiencia de uso. En esta sección se describen, explicando, por un lado, el objetivo de cada una ellas y detallando, por otro lado, la implementación desarrollada. Los siguientes puntos reflejan las características introducidas en el sistema.

- **Sombras:** dado que la plataforma utiliza RA para superponer objetos virtuales sobre el espacio físico real, se han creado sombras con el fin de dar un mayor grado de profundidad yrealismo a la escena. Para ello, se crea en tiempo de ejecución un plano negro en la base de las representaciones gráficas, el cual recibe las sombras que se generan cuando la luz de la escena incide sobre los modelos. El motivo por el cual el plano es de color negro, se debe a que el dispositivo de RA toma dicho color como transparente, permitiendo en este sentido mostrar sólo las sombras durante la visualización, haciendo que el plano sea invisible a los ojos del usuario.
- **Text To Speech (TTS):** el dispositivo de RA integra el asistente de voz de Microsoft, Cortana, el cual permite dictar comandos de voz mediante el uso de la biblioteca MRTK. Gracias a esto, diferentes frases se han definido para que el asistente las reproduzca y, de este modo, facilite al usuario la realización de ciertas tareas. Cabe destacar que, por el momento, la reproducción de voz sólo se puede realizar en inglés, dado que el dispositivo no cuenta con su versión en castellano.
- **Reconocimiento de voz:** el usuario, durante la ejecución del cliente, puede hacer uso de una serie comandos de voz, los cuales disparan una serie de acciones. Esto se ha conseguido mediante la biblioteca MRTK, la cual permite integrar este tipo de funcionalidad. Cabe destacar que el reconocimiento de comandos de voz es en inglés, dado que el dispositivo no cuenta con su versión en castellano. En § E.2 se muestra una lista con los comandos de voz que el usuario puede ejecutar.

6.2.6 Patrones de diseño

De igual modo que en el desarrollo del servidor, el cliente también ha hecho uso de diferentes patrones de diseño, los cuales han permitido mejorar el diseño del mismo y reutilizar partes del código, facilitando en este sentido la expansión de funcionalidades. A continuación, se describen los patrones utilizados junto a las ventajas que han proporcionado al problema en cuestión.

En primer lugar, el primer patrón a destacar dentro de esta sección, es el patrón «Singleton». Este patrón se ha utilizado para mantener una única instancia de objetos relacionados con la comunicación y acceder de forma global en diferentes clases del mismo. Además, este diseño ha permitido mantener objetos vivos a lo largo de la ejecución (mantener el objeto en ejecución durante cambios de escena en Unity 3D).

Siguiendo en este contexto, el patrón «Observer» ha sido otra de las soluciones adoptadas

6. ARQUITECTURA

para el desarrollo del cliente. Dado que varios mensajes pueden llegar de la parte del servidor, la utilización de este patrón ha permitido notificar eventos a los objetos que esperan dichos mensajes.

Finalmente, otro patrón de diseño a destacar dentro de esta sección es el patrón «State». Este patrón ha permitido al *Sistema de Recepción de archivos JSON* y al *Sistema de Captura* realizar una acción determinada en función del estado interno que ambos almacenan en un determinado momento.

Capítulo 7

Resultados

Tras describir la problemática, los objetivos planteados, la metodología empleada y las decisiones de diseño adoptadas para el desarrollo del trabajo, se exponen los resultados conseguidos en términos del aspecto final del sistema. Aparte de esto, se muestra una estadística del código desarrollado, junto a unos resultados tras someter la plataforma a ciertas pruebas. Por último, se expone una estimación de costes del proyecto de acuerdo con la situación actual del mercado.

7.1 Aspecto y funcionalidad

Antes de mostrar el aspecto del sistema y explicar su funcionalidad a través de un ejemplo, primero es necesario exponer los diseños elaborados tanto del cliente como del servidor.

Respecto al cliente, varios modelos se han desarrollado para representar sentencias de programación a través de una notación de carreteras y señales de tráfico. Por esto, de cara a una mejor comprensión de los modelos, se muestra, a continuación, una serie de figuras que ilustran la metáfora asociada a cada sentencia de programación.

En la Figura 7.1 se muestra la metáfora empleada para representar la instrucción bucle, cuyo diseño se asocia al de una rotonda. Esta analogía refleja la relación que ambos elementos muestran a la hora de realizar una acción repetidas veces.



Figura 7.1: Sentencia bucle

7. RESULTADOS

Otra metáfora diseñada para este sistema se muestra en la Figura 7.2, en la cual se representa la sentencia `if ... then ... else` mediante una bifurcación. Este diseño intenta mostrar la división de la ejecución en función de la condición de la instrucción.

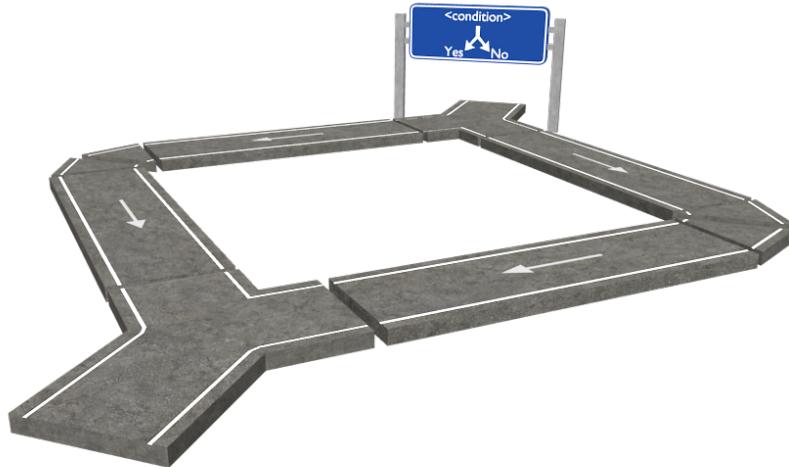


Figura 7.2: Sentencia `if ... then ... else`

Por último, la Figura 7.2 refleja la metáfora empleada para representar la definición de función y sentencia de retorno. Este diseño intenta mostrar la relación de ambos elementos con la señal de comienzo de autovía y fin de población.

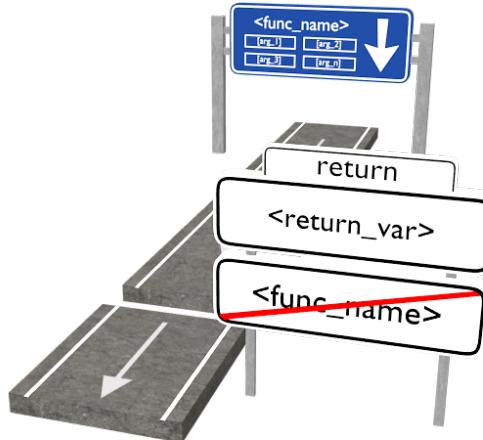


Figura 7.3: Definición de función y sentencia de retorno

En relación con el servidor, se ha diseñado una interfaz para permitir al usuario interactuar de una forma más cómoda con el sistema. Esta interfaz cuenta con tres vistas, las cuales permiten mostrar el código QR que almacena la dirección de red del servidor, retransmitir el contenido del dispositivo de RA y reflejar la conversión del código fuente de Java a formato JSON. La Figura 7.4 muestra las pestañas que componen la vista del plug-in.

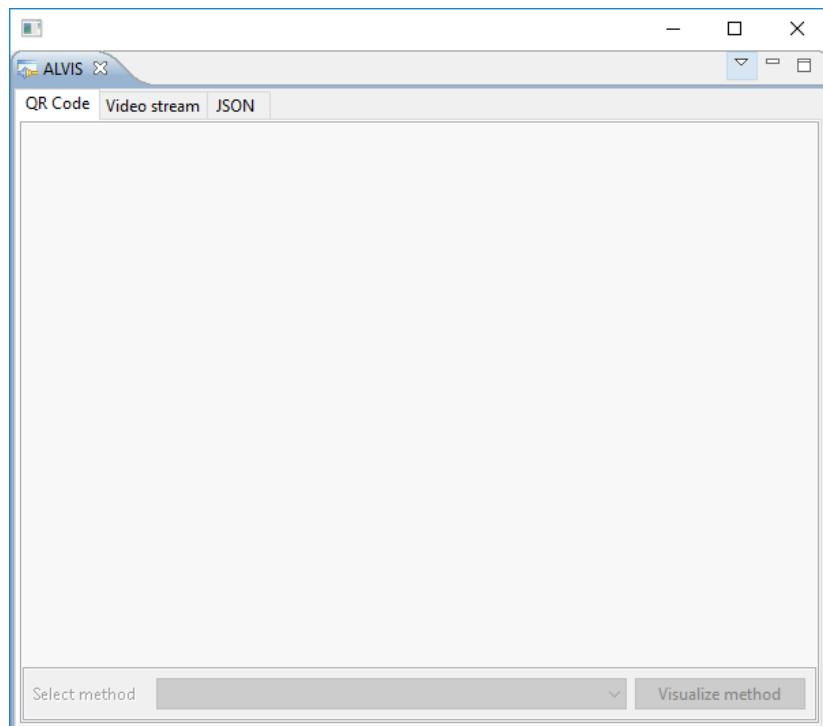


Figura 7.4: Diseño de la vista del plug-in de Eclipse

Una vez se han mostrado los diseños desarrollados para cada elemento, se describe de forma resumida la funcionalidad principal del sistema, acompañando mediante imágenes el aspecto tanto del plug-in desplegado en la parte del servidor como de la aplicación ejecutada en la parte del cliente.

Con respecto al servidor, en la Figura 7.5a se refleja la pestaña encargada de mostrar el código QR que contiene la dirección de red del mismo. A través de esta ventana, el usuario se conecta con el cliente y selecciona el método que desea visualizar. Estas funciones se realizan con los botones inferiores de la pestaña, los cuales permanecen deshabilitados hasta que la conexión no esté establecida.

Para retransmitir el contenido del dispositivo de RA se utiliza la pestaña de la Figura 7.5b, que permite a través de los botones inferiores comenzar o detener el proceso. De igual forma que en la pestaña anterior, este botón permanece deshabilitado hasta que la conexión no esté establecida.

Respecto de la última pestaña del plug-in, la Figura 7.5c muestra el código JSON que se genera tras el procesamiento de un método, el cual ha sido previamente elegido por el usuario en la pestaña del código QR.

En cuanto al cliente, la aplicación comienza mostrando una serie de imágenes a través de una animación que representan las entidades involucradas en el desarrollo de este proyecto. En la Figura 7.5d se muestra un ejemplo de estas imágenes.

En la Figura 7.5e se refleja el resultado de incluir un proceso de carga al desaparecer

7. RESULTADOS

las imágenes anteriores, cuyo fin es el de proporcionar un feedback al usuario de que una operación asíncrona se está ejecutando en segundo plano (carga de la siguiente escena).

Tras finalizar el proceso de carga que se refleja en la imagen anterior, la Figura 7.5f muestra el menú con las acciones que la aplicación puede realizar, las cuales son: escanear un código QR o salir de la aplicación.

En el caso de comenzar a analizar un código QR, la Figura 7.5g refleja un marco para facilitar al usuario el proceso de escaneado de dichos códigos.

Después de esto, una vez el usuario ha seleccionado el método que desea visualizar, el dispositivo genera la representación gráfica en el espacio físico real (ver Figura 7.5h). Cada representación tiene asociado un menú, el cual se muestra en la Figura 7.5i. Mediante esto, el usuario puede modificar y rotar dicha representación (ver Figura 7.5k), además de cambiar su posición (ver Figura 7.5j). Aparte de esto, el sistema soporta crear varias representaciones gráficas en el mismo espacio físico, lo que permite realizar la visualización de varios programas o algoritmos simultáneamente. En la Figura 7.5ñ se muestran un ejemplo de la representación de dos algoritmos ubicados sobre una mesa.

Por otro lado, en aquellos modelos, los cuales cuentan con una señal o caja, se refleja la expresión o condición asociada a la instrucción que éstos representan (ver Figura 7.5l). Además, estos modelos tienen otra peculiaridad frente a los demás, puesto que integran un papel que es desplegado cuando el usuario realiza el gesto *Air Tap* sobre ellos (en § E.2 se proporciona información sobre la utilización de este gesto). En la Figura 7.5m y 7.5n se muestra el resultado obtenido tras implementar la interacción del usuario con los modelos.

Finalmente, cabe destacar que las representaciones mostradas en el espacio físico incorporan sombras, las cuales dan un mayorrealismo a la visualización. Éstas se reflejan durante la ejecución de la aplicación, es decir, mientras un usuario utiliza el dispositivo. Sin embargo, a la hora de capturar imágenes o realizar la retransmisión, las sombras se proyectan sobre un plano negro, impidiendo en este sentido su representación. Debido a esto, se ha prescindido reflejar el plano en las figuras, con el motivo de representar con mayor claridad los modelos de las carreteras.



Figura 7.5: Grid que representa mediante imágenes el aspecto final del sistema

7. RESULTADOS

7.2 Estadística del proyecto

En esta sección se exponen las estadísticas extraídas de los repositorios utilizados para el desarrollo del proyecto, los cuales hacen referencia tanto a la aplicación del cliente como del servidor. A continuación, por un lado, se muestran los resultados obtenidos tras emplear la herramienta *CLI cloc*¹, la cual ha permitido contabilizar el número de líneas de código del proyecto. Por otro lado, se presenta mediante *Awesome Graphs*, un plug-in de BitBucket, una serie de gráficos que reflejan el esfuerzo y trabajo dedicados en el desarrollo del trabajo.

En el Cuadro 7.1 se muestra los lenguajes utilizados tanto en la parte del cliente como en la parte del servidor, reflejando el número de archivos, comentarios, líneas en blanco y líneas de código asociados a cada lenguaje.

Lenguaje	Archivos	Espacios en blanco	Comentarios	Código
C#	32	588	634	2540
Java	14	137	432	956
Total	46	725	1066	3496

Cuadro 7.1: Estadísticas del código del proyecto entre cliente y servidor

Por otro lado, las Figuras 7.6 y 7.7 reflejan las contribuciones realizadas para el desarrollo tanto del cliente como del servidor, respectivamente. En ambas figuras, el eje Y representa el número de *commits* realizados en un periodo de tiempo específico, mientras que el eje X refleja los meses que abarca el desarrollo del sistema.

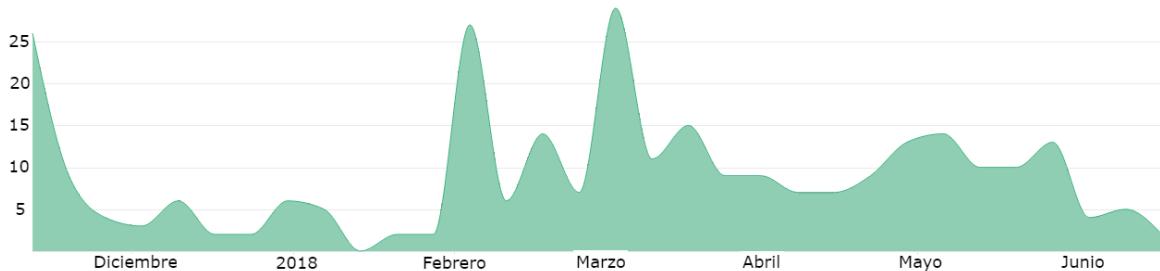


Figura 7.6: Diagrama que muestra las contribuciones de la aplicación del cliente

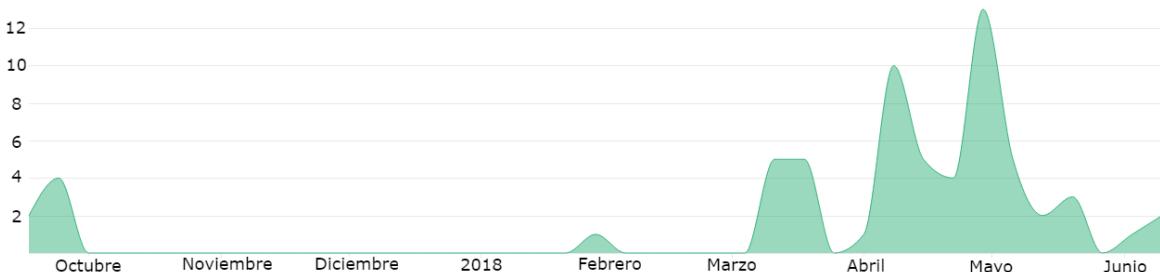


Figura 7.7: Diagrama que muestra las contribuciones de la aplicación del servidor

¹<http://cloc.sourceforge.net/>

7.3 Costes

El desarrollo de este proyecto abarca desde octubre de 2017 hasta junio de 2018. En el transcurso del año 2017, se trabajó en el proyecto a media jornada durante 5 días a la semana, mientras que a comienzos del año 2018 se trabajó a jornada completa durante seis días a la semana. Por lo tanto, esto supone un total de 700 horas de trabajo desarrollado, aproximadamente. Cabe destacar que en estas horas no se ha incluido el tiempo empleado para la elaboración del presente documento.

El sueldo del desarrollador se ha estimado en función del salario actual que obtienen los profesionales del sector informático. El salario de estos profesionales, según *Infojobs*², es de unos 30 €/hora, el cual se ha obtenido a través de varias búsquedas de ofertas de trabajo, comprobando el sueldo neto que las empresas actuales ofrecen.

En el Cuadro 7.2 se muestra un desglose de los gastos aproximados durante el desarrollo. Por un lado, se incluyen los gastos de hardware para el despliegue del proyecto y, por otro lado, se refleja el sueldo del programador teniendo en cuenta la estimación anterior.

Recurso	Cantidad	Coste
Sueldo programador (700 horas)	1	21.000,00€
Computador personal	1	600,00€
Microsoft HoloLens	1	3.299,00€
Total		24.899,00€

Cuadro 7.2: Desglose de costes de AsgAR

7.4 Medidas de rendimiento

En esta sección se presenta un análisis de la eficiencia y rendimiento de AsgAR a través de la medición de los tiempos que el sistema emplea para realizar las tareas más críticas:

- Procesamiento del código fuente de Java.
- Envío/recepción de datos por red.
- Creación de representaciones gráficas en el dispositivo de RA.
- Dibujado de fotogramas en la vista.

Para la primera tarea se ha realizado una prueba que ha consistido en medir cuál es el tiempo que el sistema invierte en procesar un programa o algoritmo en función de su tamaño (número de líneas de código). Para ello, se ha utilizado el algoritmo de la burbuja (ver Listado 7.1), con el fin de obtener datos reales introduciendo un procedimiento complejo de procesar. Durante la prueba, el algoritmo ha sido repetido en cada iteración para incrementar el número de líneas de código y comprobar el tiempo que emplea el sistema encargado en el análisis

²<https://www.infojobs.net/>

7. RESULTADOS

del código fuente de Java. La primera iteración comienza con el algoritmo repetido 10 veces, obteniendo un total de 100 líneas de código, mientras que la última iteración el algoritmo cuenta con 400 líneas, dando lugar a un total de 40 iteraciones. Los resultados obtenidos tras la realización de esta prueba se reflejan en la Figura 7.8 (proceso realizado con el equipo descrito en § 5.4.1).

```
1 public static void bubblesort(int nums[]) {
2     for(int i = 0; i < nums.length; i++) {
3         for(int j = 1; j < nums.length - 1; j++) {
4             if(nums[j] > nums[j+1]) {
5                 int aux = nums[j];
6                 nums[j] = nums[j + 1];
7                 nums[j+1] = aux;
8             }
9         }
10    }
11 }
```

Listado 7.1: Algoritmo de la burbuja

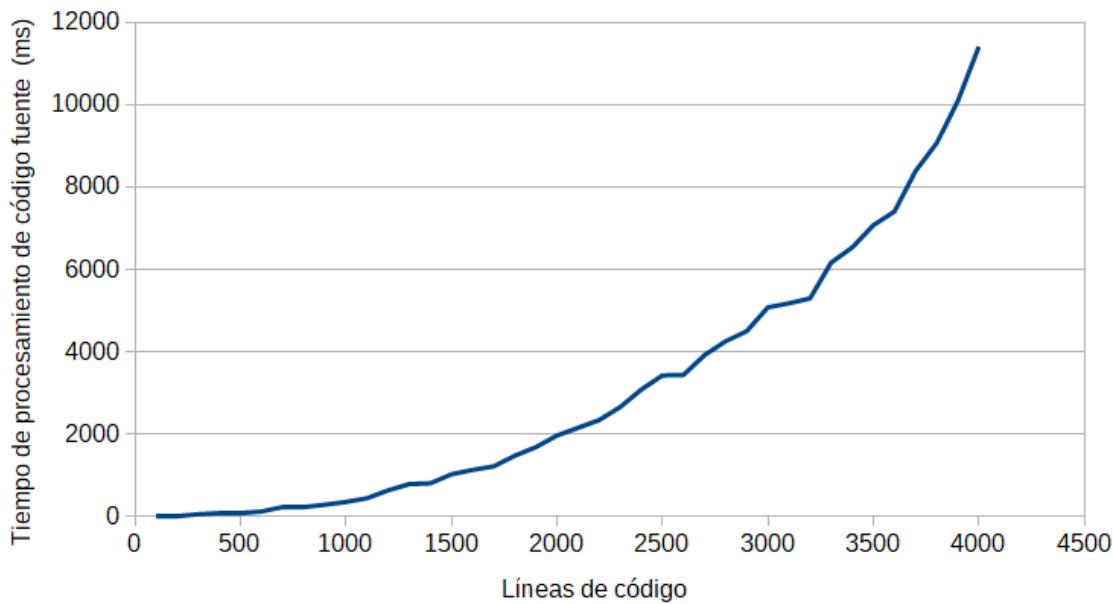


Figura 7.8: Tiempo de procesamiento del código fuente en función del nº de líneas de código

En relación con lo anterior, se ha realizado una prueba para conocer los tiempos de envío y recepción entre cliente y servidor teniendo en cuenta el tamaño del paquete transmitido (líneas de código). Para ello, se ha seguido el procedimiento anteriormente comentado, utilizando el mismo algoritmo y repitiendo el número de líneas de código en cada iteración para aumentar el número de bytes del archivo. En la Figura 7.9 se muestra un gráfico donde se refleja el tiempo que tardan los archivos en enviarse desde el servidor al cliente.

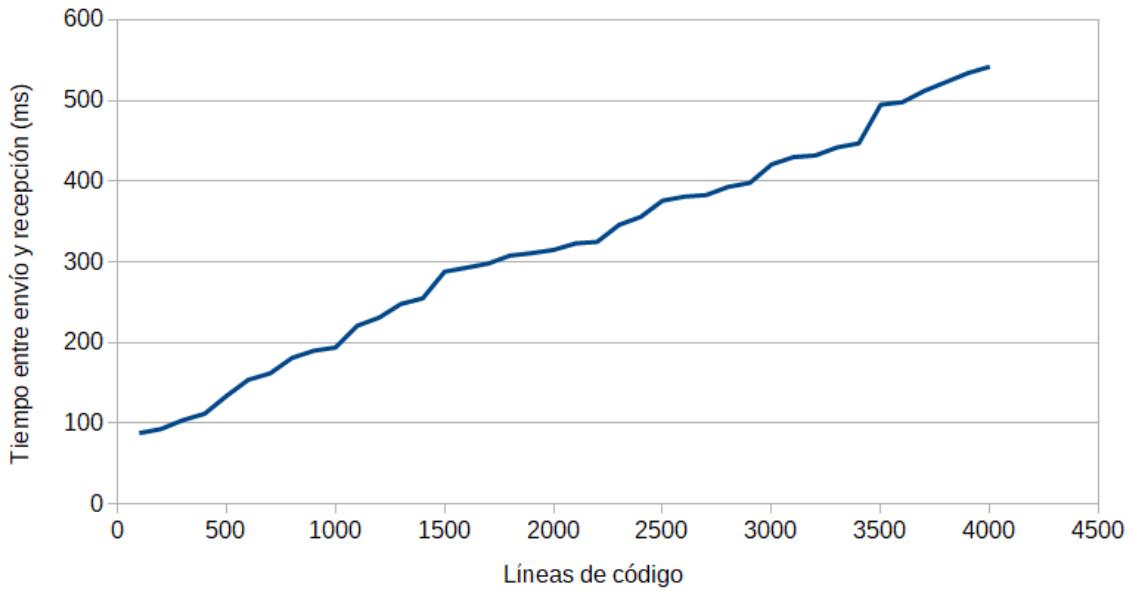


Figura 7.9: Tiempo de envío y recepción de paquetes en función del tamaño del archivo

Siguiendo en este contexto, otro análisis se ha llevado a cabo para medir el tiempo que emplea el dispositivo de RA en generar las representaciones gráficas correspondientes. Para realizar esta prueba se han seguido la lógica de los procedimientos anteriores, aumentando las líneas de código del algoritmo en cada iteración con el fin de comprobar el tiempo que invierte el sistema generador de representaciones gráficas en crear la visualización. El resultado obtenido tras la realización de esta prueba se refleja en la Figura 7.10.

Por último, en cuanto a la retransmisión de los fotogramas se ha realizado un análisis con el fin de medir el tiempo desde que un fotograma es capturado por el dispositivo de RA hasta que es dibujado en la vista de Eclipse. Para ello, se ha hecho uso del algoritmo de las pruebas anteriores, con la diferencia de que éste permanece en su estado original, es decir, sin repetir el número de líneas de código. El objetivo del análisis es comprobar que el tiempo que emplea el sistema en llevar a cabo este proceso es constante, es decir, la retransmisión del contenido generado por el dispositivo se realiza de forma lineal. En la Figura 7.11 se refleja el resultado obtenido tras el análisis.

Como podemos observar en las tres primeras pruebas, el número de líneas de código de un algoritmo o programa influyen respecto al tiempo que el sistema emplea en generar una representación gráfica. No obstante, esto ocurre cuando el archivo es muy pesado, algo que no suele ser frecuente, puesto que el sistema no está pensado para soportar programas o algoritmos con grandes cantidades de líneas de código. Respecto a la última prueba, el resultado conseguido es el esperado, dado que los tiempos reflejados en el diagrama muestran que la acción de recuperar y dibujar fotogramas es prácticamente la misma durante todo el proceso de retransmisión.

7. RESULTADOS

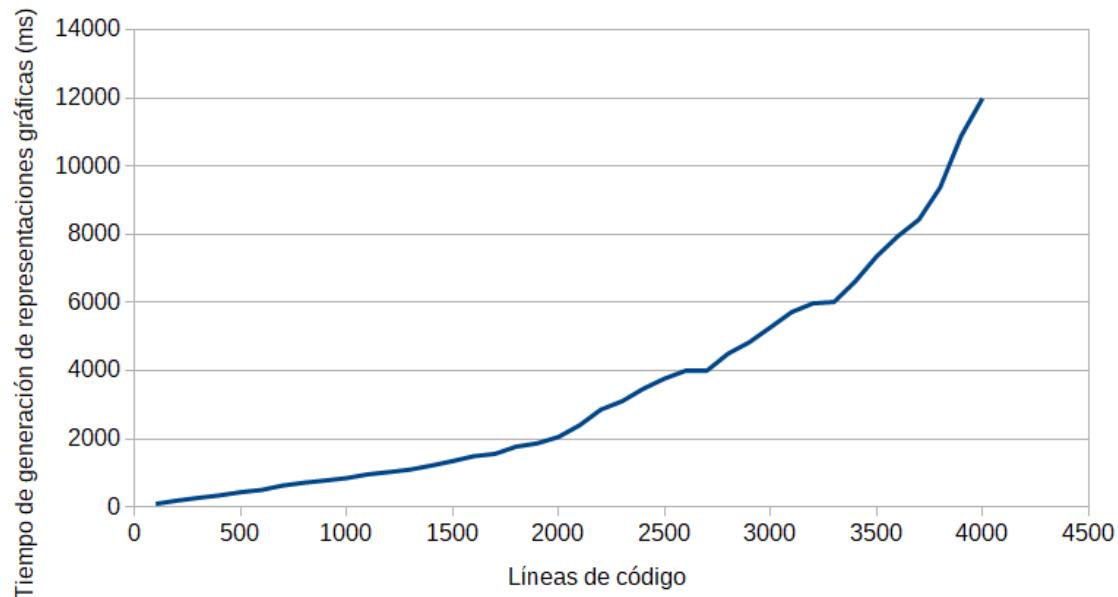


Figura 7.10: Tiempo de generación de representaciones gráficas en función del nº de líneas de código

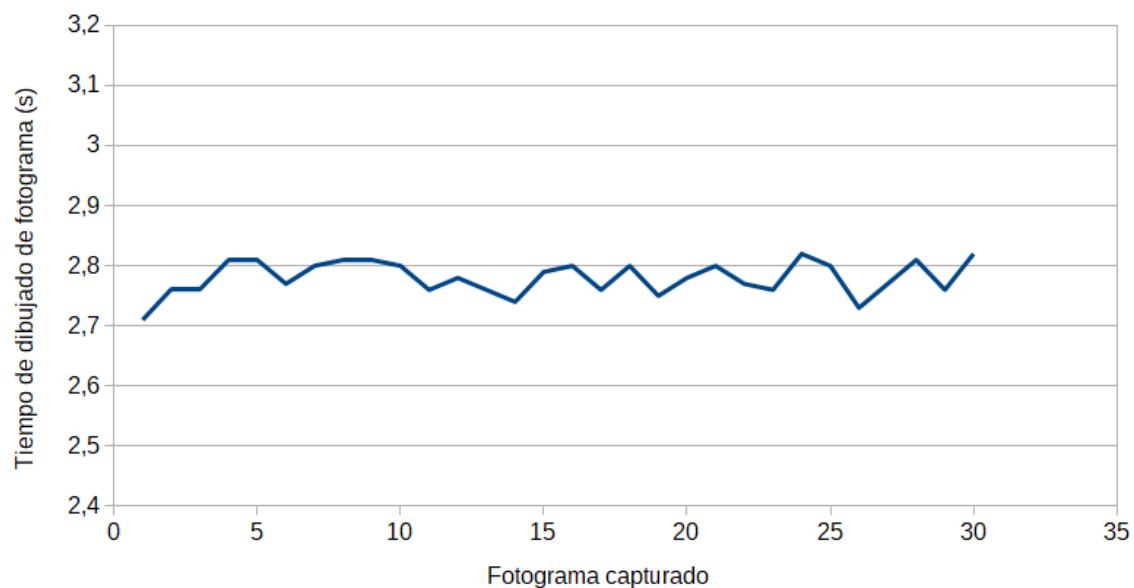


Figura 7.11: Tiempo en generar y pintar un fotograma

Capítulo 8

Conclusiones

Este trabajo presenta un breve estudio sobre la arquitectura propuesta para el aprendizaje de la programación, un entorno de RA diseñado para la generación automática de representaciones gráficas con el fin de visualizar tanto programas como algoritmos en tiempo real. Éste se integró como un plug-in de COLLECE 2.0, el cual ha facilitado el desarrollo de nuevas funcionalidades debido al soporte que proporciona. Además, la elección del dispositivo de RA de Microsoft ha sido todo un acierto, puesto que la inclusión de modelos tridimensionales en el espacio físico focaliza la atención del estudiante y mejora su experiencia de aprendizaje. Por lo tanto, tras los resultados plasmados en el capítulo de resultados (ver § 7), se resumen las conclusiones más importantes junto a una breve interpretación de cada uno de ellos. Además, se incluyen unas posibles líneas de trabajo futuras que nacen a partir de este trabajo.

8.1 Conclusiones

En primer lugar, uno de los logros más importantes que se han llevado a cabo durante la elaboración del trabajo, ha sido la **integración de COLLECE 2.0** con el plug-in creado para la automatización de la generación de representaciones gráficas. Alcanzar este objetivo ha permitido facilitar en una gran medida el desarrollo de otros componentes con el fin de mejorar el aprendizaje de la programación.

En segundo lugar, dado que se usa un dispositivo para sumergir al alumno en un entorno holográfico tridimensional, ha sido un éxito la implementación de la **retransmisión en vídeo** del contenido que el dispositivo visualiza en la plataforma Eclipse. Esto ha logrado que aquellos alumnos que no cuenten con el dispositivo puedan introducirse de algún modo en el entorno de aprendizaje de RM.

En tercer lugar, una parte decisiva para la creación de las representaciones gráficas, ha sido el **procesamiento del código fuente** de Java. Este logro ha permitido la construcción tanto de programas como de algoritmos en 3D sin la necesidad de invertir esfuerzo y tiempo.

En cuarto lugar, el proyecto desarrollado basa su implementación en varios **patrones de diseño**, los cuales son: «Singleton», «Observer», «Modelo-Vista-Controlador», «State», «Factory Method» y «Visitor». El objetivo de la utilización de estos mecanismos ha sido ase-

8. CONCLUSIONES

gurar la eficiencia, mantenibilidad y escalabilidad del proyecto de futuras ampliaciones.

En último lugar, cabe destacar la importancia de la **arquitectura de red** desarrollada, la cual permite el envío y recepción de archivos con el contenido esencial. La realización de este objetivo ha facilitado la comunicación entre los entornos discutidos, de tal forma que ambos puedan intercambiar información para el correcto funcionamiento del sistema.

8.2 Líneas de trabajo futuras

En cuanto a las líneas de trabajo futuras, a continuación se presentan una serie de mejoras y evoluciones que pueden sacar el máximo partido al proyecto desarrollado en este trabajo.

Una de las evoluciones que se pretenden abordar próximamente es la **visualización dinámica** de las representaciones basadas en carreteras y señales de tráfico. En este momento, las visualizaciones generadas son estáticas, las cuales necesitan ser previamente evaluadas para obtener constancia de su efectividad. El objetivo de esta mejora es componer tanto programas como algoritmos que puedan ser evaluados dinámicamente utilizando la Java Virtual Machine (JVM) y las herramientas de depuración que proporciona Eclipse, con el fin de representar la ejecución de un programa como si de un depurador interactivo se tratara y permitir al estudiante interactuar con los elementos de la representación. Con esto se pretende que el usuario pueda examinar el valor de las variables, la pila de llamadas que han desembocado en la situación actual y el interior de una función, entre otras funciones. De este modo, se espera facilitar la comprensión de los alumnos sobre una ejecución particular del programa o algoritmo. Se estima que el coste temporal para la realización de esta tarea sea de dos meses, aproximadamente, puesto que es necesario realizar un estudio de las ventajas que ofrece el depurador integrado en la herramienta de Eclipse, junto a la modificación de algunos scripts del sistema, entre otras muchas tareas.

Otra extensión que puede ser interesante incluir en este trabajo es la integración del sistema a una **plataforma móvil**, de tal modo que éste pueda ser más accesible a los estudiantes. Lo que se pretende con esta idea es utilizar frameworks de desarrollo disponible que permitan realizar un tracking sin marcas en dispositivos iOS y Android, como *ARKit* y *ARCore*, respectivamente. Esto sin duda brindará al sistema de una mayor flexibilidad, puesto que hoy en día el dispositivo más extendido a nivel mundial es el smartphone. Se estima que esta tarea tenga un coste temporal de tres semanas, aproximadamente, ya que es fácil la integración de estas bibliotecas en el sistema debido a la arquitectura escalable desarrollada.

Respecto a la generación de representaciones gráficas, el diseño está planteado para que sentencias `if ... then ... else`, `switch ... case` o bucles `for` o `while` sean creadas en tiempo de ejecución. Por lo tanto, la **agregación de nuevas abstracciones** como bucles `do ... while` o sentencias `try ... catch` permitirían abarcar un mayor número de programas o algoritmos que pueden ser visualizados para mejorar la comprensión de los alumnos. Se estima que el coste temporal para realizar esta tarea sea alrededor de un mes, puesto que primero es

necesario llevar a cabo una serie de diseños para evaluar la constancia de su efectividad.

Una posible mejora, en cuanto a las modificaciones que se pueden realizar para perfeccionar el sistema de posibles fallos, sería entorno al **procesamiento del código fuente** en Java. En el caso de la sentencia `switch ... case`, ésta no se procesa cuando carece de sentencias `break`. Por lo tanto, la visualización de la misma no se realiza. La solución que debe llevarse a cabo para solventar este problema, es crear una estructura de datos que permita analizar de forma sencilla y eficiente el contenido de la sentencia para generar el código deseado. Se estima que la duración para la realización de esta tarea sea alrededor de una semana.

Por otro lado, aunque la arquitectura de red planteada funciona correctamente, mejoras respecto a la **retransmisión de vídeo** deberían desarrollarse, ya que el contenido mostrado sufre un pequeño retardo que impide su correcta reproducción. Este problema podría solventarse usando algún protocolo de transmisión de vídeo en tiempo real, el cual se encargue de enviar únicamente las diferencias de un fotograma con respecto del que se envió anteriormente, evitando mandar fotogramas completos. Se estima que el coste temporal para llevar a cabo la realización de esta tarea sea de tres semanas, puesto que es necesario elaborar un estudio del protocolo que mejor se adecúe a las características de este sistema.

Finalmente, una posible mejora podría realizarse en torno a **optimizar el código que se encarga de generar las representaciones gráficas** en el dispositivo de RA. Éstas son procesadas mediante un algoritmo recursivo, el cual puede producir un desbordamiento de la pila del sistema en el caso de procesar programas o algoritmos de grandes dimensiones. La solución para solventar este problema sería modificar el algoritmo recursivo a iterativo, ya que, aunque es difícil que esta situación pueda darse, es una buena solución en cuanto a mejorar la eficiencia del procedimiento. Se estima que el tiempo empleado para llevar a cabo esta tarea sea de una semana.

8.3 Conclusión personal

El desarrollo de AsgAR, plataforma discutida en este documento, ha supuesto el broche final a mis 4 años de grado, en los cuales he adquirido una serie de conocimientos y habilidades que me han permitido elaborar el proyecto expuesto en este trabajo. No obstante, a pesar de las destrezas adquiridas, no todo ha sido un «camino de rosas», puesto que he tenido que enfrentarme a nuevos desafíos que han implicado el aprendizaje de tecnologías novedosas, dispositivos en auge como las gafas de RA de Microsoft, etc.

También me gustaría dejar constancia que el abordar un proyecto de tales magnitudes me ha hecho ver el desarrollo desde otro punto de vista, el cual no suele calar durante la realización de trabajos o prácticas en el grado. La utilización de técnicas de ingeniería del software, tales como los patrones de diseño, han permitido diseñar un sistema eficiente, mantenable y fiable, consiguiendo una arquitectura robusta y escalable.

8. CONCLUSIONES

Además, gracias al desarrollo de este proyecto he podido conocer ciertas herramientas que antes desconocía, como es el caso de Unity 3D, con la cual se ha desarrollado todo lo relacionado con las representaciones basadas en carreteras y señales de tráfico. Esto me ha permitido adquirir un conjunto de conocimientos, los cuales me son útiles para seguir formándome en un campo apasionante, el cual es la RA.

Finalizando esta conclusión, quiero dejar unas líneas respecto a mis pensamientos sobre el proyecto en cuestión. AsgAR se ha creado para estudiantes de niveles universitarios de Ingeniería Informática, el cual pretende ser una ayuda en cuanto al aprendizaje de la programación se refiere. Para ello, este trabajo ha requerido de una dedicación constante, el cual ha dado lugar al resultado plasmado en este documento (ver § 7). Por lo tanto, tras el esfuerzo oculto en estas líneas y el beneficio que supondría a los alumnos, me gustaría ver este sistema utilizarse en las diferentes aulas de la ESI, el cual sirviera a los estudiantes de primer año a contemplar el grado como algo divertido, donde el aprendizaje y el disfrute estén en armonía.

Capítulo 9

Conclusions

This document presents a brief study of the proposed architecture for the programming learning, an AR environment designed for the automatic generation of graphical representation to visualize both programs and algorithms in real time. It was integrated as a COLLECE 2.0 plug-in, which has facilitated the development of new features due to the support it provides. In addition, the option of selecting the Microsoft AR device has been totally a success, since the inclusion of three-dimensional models in the physical space focuses the attention of the students and improves their learning experience. Therefore, after the results shown in the results chapter (see § 7), the most important conclusions are summarized along a brief interpretation of each of them. Moreover, there are some possible future working lines that are born from this project.

9.1 Conclusions

First of all, one of the most important achievements during the elaboration of this project has been the **integration of COLLECE 2.0** with the plug-in created for the automation of the generation of graphic representations. Achieving this goal has greatly facilitated the development of other components to improve the programming learning process.

Secondly, since a device is used to immerse the student in a three-dimensional holographic environment, the implementation of the **video streaming** of the content that the device displays on the Eclipse platform has been totally a success. It has meant that those students who do not get the device can somehow be introduced into a MR learning environment.

Thirdly, a decisive part of the creation of the graphical representations has been the **processing of the Java source code**. This achievement has allowed the construction of both programs and 3D algorithms without the need to invest effort and time.

Fourthly, the developed project bases its implementation on several **design patterns**, which are: «Singleton», «Observer», «Model-View-Controller», «State», «Factory Method» and «Visitor». The objective of the use of these mechanisms has been to ensure the efficiency, maintainability and scalability of the project for future expansions.

Finally, it is important to mention the **network architecture** developed, which allows to

9. CONCLUSIONS

send and receive files with the essential content. The achievement of this objective has facilitated the communication between the environments discussed, so that both can exchange information for the proper performance of the system.

9.2 Future lines of work

This section presents a set of improvements and developments that can take advantage of the developed project that is exposed in this document.

One of the developments that we intend to address in the near future is the **dynamic visualization** of road-based graphical representation. At this moment, the generated visualizations are static, which need to be previously evaluated to obtain proof of their effectiveness. The objective of this improvement is to compose both programs and algorithms that can be dynamically evaluated using the JVM and debugging tools provided by Eclipse, in order to represent the execution of a program as if it were an interactive debugger and allow the student to interact with elements of the representation. This is intended to enable the user to examine the value of the variables, the stack of calls that have led to the current situation, and the block of a function, among other things. In this way, it is expected to facilitate students' understanding of a particular implementation of the program or algorithm. It is estimated that the temporary cost for the implementation of this task is approximately two months, since it is necessary to carry out a study of the advantages offered by the debugger integrated in the Eclipse tool, together with the modification of some system scripts, among many others other things.

Another extension that may be interesting to include in this project is the integration of the system into a **mobile platform**, so that it can be more accessible to students. The aim of this idea is to use available development frameworks that allow for unmarked tracking on iOS and Android devices, such as *ARKit* and *ARCore*, respectively. It will undoubtedly give the system greater flexibility, since, nowadays, the most widespread device in the world is the smartphone. It is estimated that this task has a temporary cost of approximately three weeks, since it is easy to integrate these libraries into the system due to the developed scalable architecture.

With regard to the generation of graphic representations, the design is proposed so that `if ... then ... else, switch ... case or, for or while loops` be created at runtime. Therefore, the **aggregation of new abstractions** as loops `do... while` or `try ... catch` statements will make it possible to cover a greater number of programs or algorithms that can be visualized to improve the understanding of the students. It is estimated that the temporary cost to carry out this task takes about a month, since a set of designs is first necessary to evaluate the constancy of their effectiveness.

A possible improvement, in terms of the modifications that can be made to improve the system of possible failures, would be towards the **processing of the source code** in Java.

In the case of the `switch ... case` statement, it is not processed in the absence of `break` statement. Therefore, it is not displayed. The solution that must be implemented to solve this problem, is to create a data structure that allows for simple and efficient analysis of the content of the statement to generate the desired code. It is estimated that the duration of this task will be about one week.

On the other hand, although the proposed network architecture works correctly, improvements with respect to the **video streaming** should be developed, as the displayed content suffers a small delay that prevents its proper playback. This problem could be solved by using some kind of real-time video transmission protocol, which takes over sending only the differences of one frame with respect to a one that was previously sent, avoiding sending full frames. It is estimated that the temporary cost for this task should take three weeks to be completed, as it is necessary to prepare a study of the protocol that is best suited to the characteristics of this system.

Finally, a possible improvement could be made around **optimizing the code that is responsible for generating the graphical representations** in the AR device. Graphical representation are processed using a recursive algorithm, which can cause the system stack overflow when processing large programs or algorithms. The solution to solve this problem would be to modify the recursive algorithm to a iterative one, since, although it is difficult that this situation can be given, it is a good solution in terms of improving the efficiency of the procedure. It is estimated that the duration of this task will be about one week.

9.3 Personal conclusion

The development of AsgAR, the platform discussed in this document, has meant the end of my 4 years of degree, in which I have acquired a series of knowledge and skills that have allowed me to elaborate the project exposed in this document. However, despite the skills acquired, not everything has been a “bed of roses”, as I have had to face new challenges that involved learning of new technologies, booming devices like Microsoft’s RA glasses, etc.

I would also like to put on record that approaching a project of such magnitude has made me see the development from another point of view, which is not usually taken into account for the performance of work or practices during the degree. The use of software engineering techniques, such as design patterns, have enabled to design an efficient, maintainable and reliable system, getting a robust and scalable architecture.

In addition, thanks to the development of this project I have been able to get to know certain tools that I did not know before, such as Unity 3D, which has been used to develop everything related to representations based on roads and traffic signs. It has allowed me to acquire a set of knowledge, which is useful to me to continue my training in an exciting field, which is the AR technology.

Finalizing this conclusion, I would like to leave a few lines regarding my thoughts on the project in question. AsgAR has been created for students at university levels of Computer Science, which is intended to be an aid to the programming learning. For this reason, this work has required a constant dedication, which has given rise to the result reflected in this document (see § 7). Therefore, I would like to see this system being used in some ESI classrooms, which would serve novices to contemplate the degree as something fun, where the learning process and enjoyment be in harmony.

ANEXOS

Anexo A

Características Microsoft HoloLens

Especificaciones técnicas	HoloLens
Óptica	Lentes holográficas transparentes Motores ligeros de 2 HD 16:9 Calibración automática según la distancia pupilar Resolución holográfica: 2.3 M puntos de luz totales Densidad holográfica: 2500 > radianes (puntos de luz por radián)
Sensores	1 IMU 4 Cámaras de compresión del entorno 1 Cámara de profundidad 1 Cámara de fotos de 2 MP y vídeo HQ Captura de RM 4 Micrófonos 1 sensor de luz ambiental
Entendimiento humano	Sonido espacial Seguimiento de la mirada Entrada de gestos Compatibilidad con voz
Entrada	Conector 3.5 mm para audio Micro USB 2.0
Salida	Altavoces integrados LED para el estado de la batería Botones para brillo y audio Botón de encendido
Conectividad	Wi-Fi 802.11ac Bluetooth 4.1 LED
Procesadores	Arquitectura de 32 bits de Intel Unidad de procesamiento diseñada para Microsoft Holographic (HPU 1.0)
Batería	2-3 horas de uso activo
memoria	Flash de 64 GB 2 GB de RAM
Sistema operativo	Windows 10

Cuadro A.1: Características del dispositivo de RA Microsoft HoloLens [hol18]

Anexo B

Contenido del CD

En este anexo se detalla el contenido del CD que acompaña a este documento.

- **código fuente:** directorio con los archivos fuente completos siguiendo la estructura detallada en el Anexo C.
- **documentación:** directorio con la documentación del proyecto, el PDF generado y los archivos fuentes L^AT_EX para su construcción.
- **binarios:** directorio donde se encuentran los archivos que permiten instalar la plataforma descrita en el presente documento (en el Anexo D se encuentran los pasos a seguir).
- **vídeo:** directorio que contiene un vídeo explicativo acerca del funcionamiento del sistema completo.
- **README.txt:** archivo que contiene toda la información relacionada con el contenido del CD.

Anexo C

Código Fuente

C.1 Cliente

La aplicación cliente, es decir, aquella que se ejecuta en el dispositivo de RA, presenta la siguiente estructura de directorios:

- **Assets:** contiene todos los recursos utilizados en el proyecto.
 - **Animations:** contiene las animaciones utilizadas por los modelos.
 - **Animator:** contiene las máquinas de estados que controlan las animaciones usadas por un objeto.
 - **External:** contiene las bibliotecas utilizadas para el desarrollo de la aplicación.
 - **Models:** almacena los modelos 3D generados con la herramienta Blender.
 - **Plugins:** contiene los componentes externos usados que extienden las capacidades de la aplicación.
 - **Prefabs:** contiene los objetos construidos en las escenas del proyecto con valores y atributos específicos.
 - **Resources:** contiene los recursos que son creados en tiempo de ejecución.
 - **Scenes:** contiene las escenas de la aplicación.
 - **Scripts:** contiene el código fuente C# de la aplicación.
 - **Shaders:** contiene los programas *vertex* y *fragment shader* aplicados a los modelos de la aplicación.
 - **UI:** almacena los modelos utilizados por la IU de la aplicación.
- **ProjectSettings:** contiene la configuración del entorno Unity 3D.

C.2 Servidor

En cuanto a la aplicación del servidor, es decir, aquella que se ejecuta en el entorno de Eclipse, presenta la siguiente estructura de directorios:

- **META-INF:** almacena los metadatos con el contenido del plug-in.
- **lib:** contiene las bibliotecas utilizadas en el desarrollo del plug-in.
- **src:** contiene el código fuente en Java del plug-in de Eclipse.

Anexo D

Instalación

En el presente anexo se incluyen las instrucciones a seguir para la instalación y despliegue del cliente y servidor.

D.1 Instalación del cliente

En esta sección se presupone que la persona interesada en instalar la aplicación del cliente cuenta con el dispositivo de RA Microsoft HoloLens. Antes de comenzar con este proceso, es necesario habilitar el **Modo de desarrollador** (Developer mode) en el dispositivo. Esta opción se encuentra en *Settings → Update & Security → For Developers*. Tras este paso, se reflejan, a continuación, los pasos que el usuario debe de seguir.

- Configura el dispositivo para usar Windows Device Portal¹ (ver *configurar HoloLens para usar Windows Device Portal* en el link).
- En un equipo, intenta conectarte al dispositivo mediante Wi-Fi o USB¹ (ver *conectarse vía USB o vía WI-FI* en el link).
- Crea un nombre de usuario y contraseña¹ (ver *crear usuario y contraseña* en el link) si es la primera vez que te conectas a Windows Device Portal, o escribe el nombre de usuario y contraseña que ya hay configurado.
- En Windows Device Portal, haz clic en **Apps** (ver Figura D.1). Fuera del cometido de esta tarea, con el objetivo de ampliar información sobre esta web, Windows Device Portal cuenta con más opciones, en las cuales se encuentra: **Mixed Reality Capture**, para retransmitir en tiempo real el contenido que visualiza el dispositivo, capturar imágenes, hologramas y audio; **3D View**, para observar cómo HoloLens interpreta nuestro entorno; y **System Performance**, para ver en tiempo real medidas de rendimiento del sistema, entre otras muchas opciones.
- En **Install app**, selecciona el archivo con extensión .appx que se encuentra en el directorio binarios.
- En **Deploy**, haz clic en **Go** para implementar el paquete de la aplicación.

¹https://docs.microsoft.com/es-es/windows/mixed-reality/using-the-windows-device-portal#setting-up_hololens_to_use_windows_device_portal

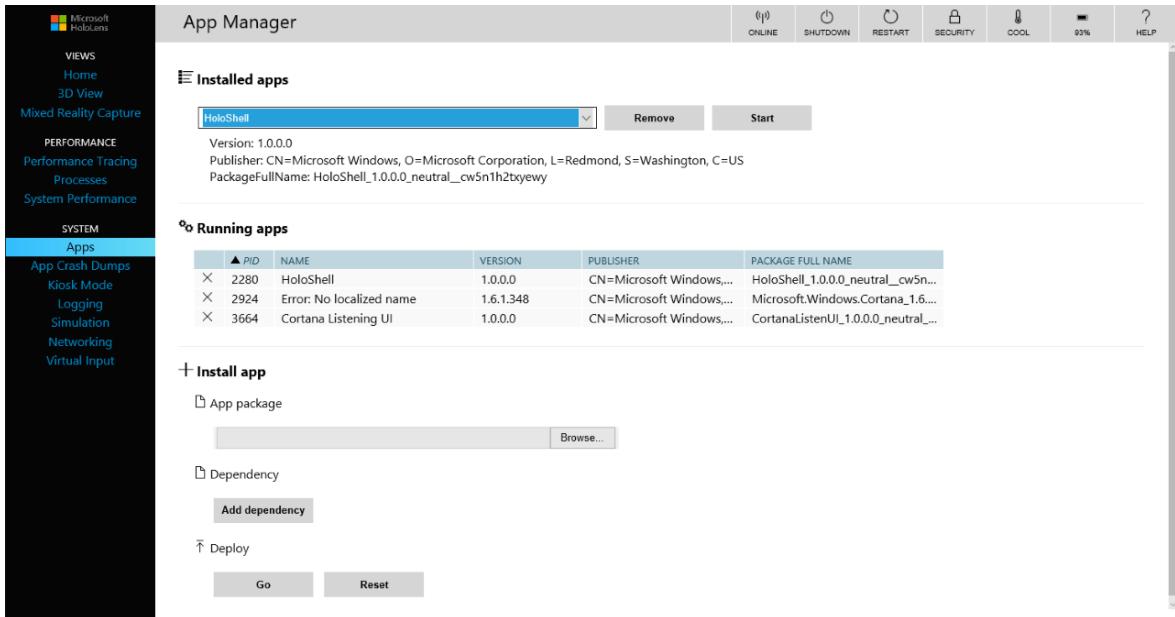


Figura D.1: Windows Device Portal

D.2 Instalación del servidor

En esta sección se presupone que la persona interesada en instalar esta aplicación cuenta con la versión Neon de Eclipse², en la cual se ha desarrollado la aplicación del servidor (plugin de Eclipse). A continuación, se muestran los pasos a seguir para llevar a cabo una correcta instalación (estos pasos son los mismos para las plataformas Windows, Mac y Linux).

- En Eclipse Neon, ir a *Help → Install New Software...*
- En el campo **Work with**, introduce la dirección para instalar COLLECE 2.0³.
- Selecciona e instala la categoría Collece-2.0.
- Una vez se ha instalado el plugin anterior, busca el archivo con la extensión .jar en el directorio binarios.
- Tras esto, busca el directorio donde se instaló Eclipse.
- En el directorio plugins, introduce el archivo anteriormente mencionado.
- Una vez hecho esto, reinicia Eclipse para guardar los cambios y comenzar a usar las funcionalidades que este plugin ofrece.

²<http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/neon3>

³<http://chico.esi.uclm.es/iapro/collece-2.0-plugin/>

Anexo E

Manual de usuario

El presente anexo constituye el manual de usuario tanto para el cliente como para el servidor, incluyendo las instrucciones necesarias para una correcta utilización del sistema.

E.1 Servidor

En esta sección se muestra la información necesaria para ejecutar y utilizar el plug-in desarrollado para Eclipse. A continuación, se describen los pasos que el usuario debe seguir.

Para desplegar el servidor, el usuario debe abrir Eclipse e ir a *Window → Show View → Other...* y seleccionar *ALVIS → ALVIS*. Tras esto, un vista aparecerá en la parte derecha del editor, la cual consta de tres pestañas: **QR Code**, **Video stream**, **JSON**.

La pestaña **QR Code** contiene dos botones, uno para seleccionar métodos de la clase activa del editor y otro para enviar dicho método a la aplicación del cliente. Es importante destacar que estos botones aparecen deshabilitados hasta que el cliente no se conecte con el servidor. Una vez el botón **Visualize method** haya sido pulsado, la traducción del código Java aparecerá en la pestaña **JSON**.

En el caso en que el usuario quiera retransmitir el contenido del dispositivo de RA, debe ir a la pestaña **Video stream** y pulsar el botón de **Start Video Streaming**. Aparte, varios usuarios pueden ver esta retransmisión en tiempo real utilizando el plug-in de COLLECE 2.0. Para desplegarlo, el usuario debe ir a *File → New → Other* y seleccionar *COLLECE → server*. A continuación deberá seleccionar el directorio donde COLLECE 2.0 almacenará la configuración e información asociada al servidor, junto al puerto de escucha de los clientes. Por último, se finaliza el asistente para tener el servidor de COLLECE 2.0 funcionando. Todo esto en cuanto al despliegue del servidor.

Por otro lado, para que el usuario pueda conectarse a un servidor de COLLECE 2.0, debe ir a *File → New → Other...* y seleccionar *COLLECE → client*. En la primera vista del asistente, el usuario puede iniciar sesión o registrarse en el servidor si no lo ha hecho antes. Una vez iniciada la sesión con los datos del usuario, éste debe elegir una sesión de trabajo y unirse a ella dándole al botón **Join**. A continuación, el asistente finalizará y los usuarios conectados a la sesión podrán visualizar, en tiempo real, el contenido del dispositivo de RA.

E.2 Cliente

En esta sección se muestra la información necesaria para ejecutar y utilizar la aplicación desarrollada en la parte del cliente. Sin embargo, antes de comenzar a describir los pasos para desplegar la aplicación, se explica, a continuación, cómo utilizar el dispositivo de RA.

Moverse por este dispositivo es diferente a usar otros sistemas que utilizan Windows. Por ejemplo, la acción de mover un cursor con un ratón se simula a través de la mirada del usuario y, la acción de clicar un ratón se simula mediante gestos con las manos. Los gestos que este dispositivo incorpora son: **Bloom Gesture** y **Air Tap**.

Bloom Gesture se utiliza para abrir tanto el menú principal del dispositivo como cerrar aplicaciones. La Figura E.1 refleja los pasos para realizar el gesto de forma correcta.



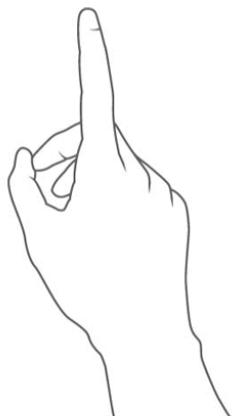
(a)



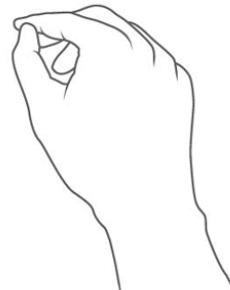
(b)

Figura E.1: (a) comienzo y (b) fin del gesto *Bloom*

En cuanto al gesto *Air tap*, éste simula la acción de clicar el botón izquierdo de un ratón. La Figura E.2 muestra los pasos a seguir para realizar el gesto de forma correcta.



(a)



(b)

Figura E.2: (a) comienzo y (b) fin del gesto *Air Tap*

Tras explicar cómo utilizar el dispositivo de RA, se procede a describir los pasos para lanzar la aplicación que se ejecuta en la parte del cliente.

Para ello, el usuario inicialmente debe hacer uso del gesto *Bloom*, el cual permite desplegar el menú del dispositivo. Una vez hecho esto, se busca la aplicación que se pretende lanzar y se hace uso del gesto *Air Tap* para comenzar con la ejecución.

La aplicación comienza con una serie de imágenes que reflejan las organizaciones involucradas en el desarrollo del proyecto. Tras esto, la aplicación muestra un menú, donde el usuario tiene la opción de escanear un código QR o salir de ella. Es posible que el código QR sea detectado, pero no permita establecer la conexión con el dispositivo de RA debido a la configuración del *firewall* de Windows. Para solucionar esto, vamos al cuadro de búsqueda de Cortana, escribimos *Firewall de Windows Defender* y clicamos en *Activar o desactivar el Firewall de Windows Defender*. Dependiendo en el tipo de red que estemos, desactivamos el *firewall* para poder establecer la comunicación entre cliente y servidor. Tras el escaneo del código que genera el plug-in de Eclipse, el dispositivo se mantendrá a la espera hasta recibir un mensaje del servidor. Este mensaje se procesará internamente dando lugar a un modelo compuesto por carreteras y señales de tráfico. Mediante un menú asociado al modelo, el usuario podrá realizar las acciones de mover, rotar y trasladar. Además, se podrá eliminar la representación siempre que el usuario lo considere oportuno.

Aparte de esto, el usuario puede interactuar con algunas instrucciones, las cuales están compuestos por una señal o caja. Clicando sobre éstas, el usuario puede desplegar un panel interactivo, reflejando la información completa que almacena.

Por otro lado, este sistema ofrece un conjunto de comandos de voz, los cuales permiten disparar una serie de acciones. Los comandos definidos para este sistema son:

- **Show surface:** permite mostrar a través de un material las superficies reconocidas por el dispositivo. Este comando puede ser utilizado una vez cliente y servidor estén conectados.
- **Hide surface:** oculta el material habilitado con el comando anterior. Este comando puede ser utilizado una vez cliente y servidor estén conectados.
- **Show shadows:** permite mostrar las sombras de las representaciones gráficas sobre el espacio físico real. Este comando puede ser utilizado mientras una representación gráfica se esté visualizando.
- **Hide shadows:** oculta las sombras habilitadas con el comando anterior. Este comando se puede utilizar mientras una representación gráfica esté visualizándose.

Por último, cabe destacar que el sistema permite la generación simultánea de representaciones gráficas, posibilitando tener múltiples programas o algoritmos en un mismo entorno.

Anexo F

Diagramas UML

F.1 Servidor

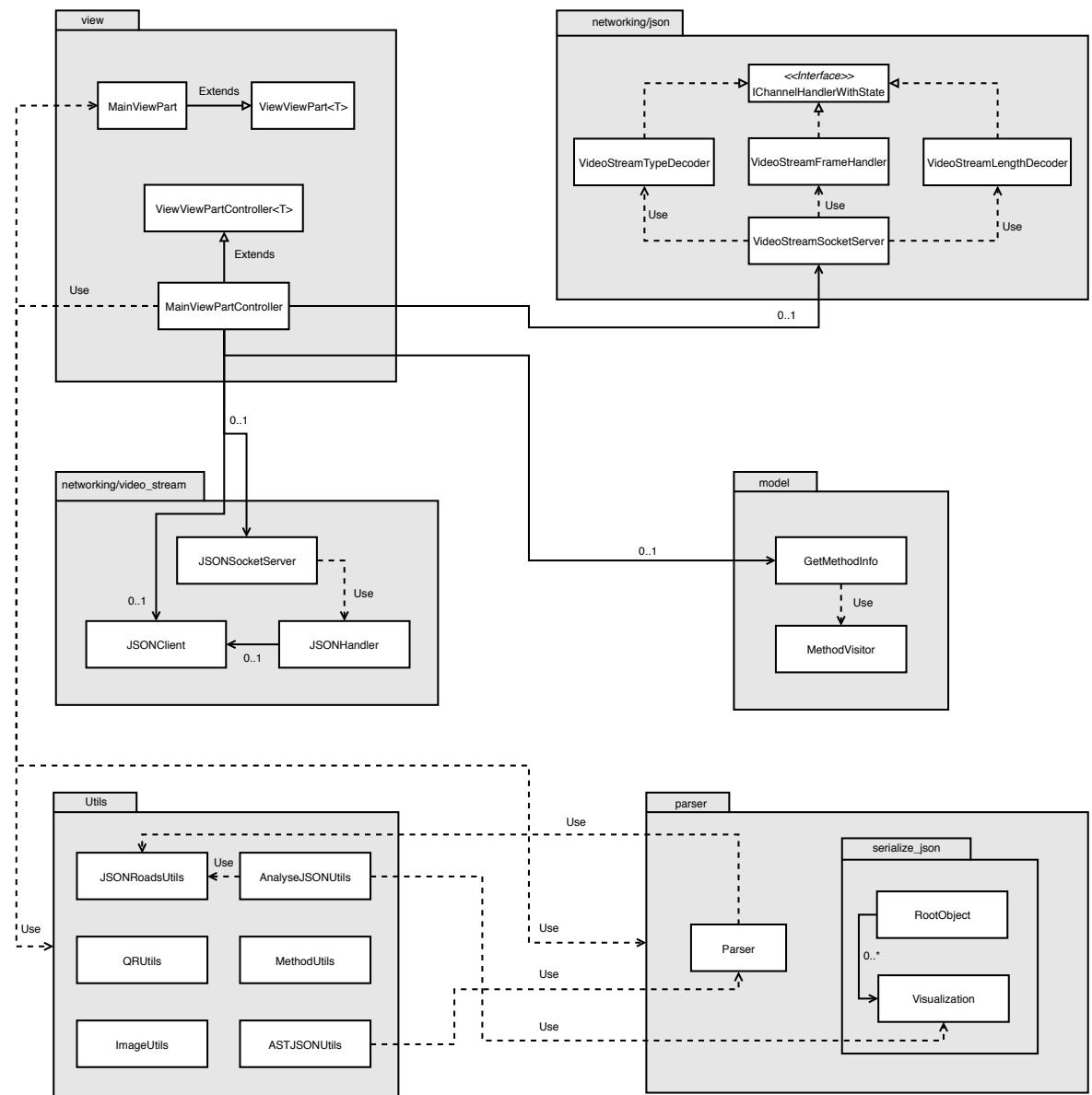


Figura F.1: Diagrama UML del plug-in de Eclipse

F.2 Cliente

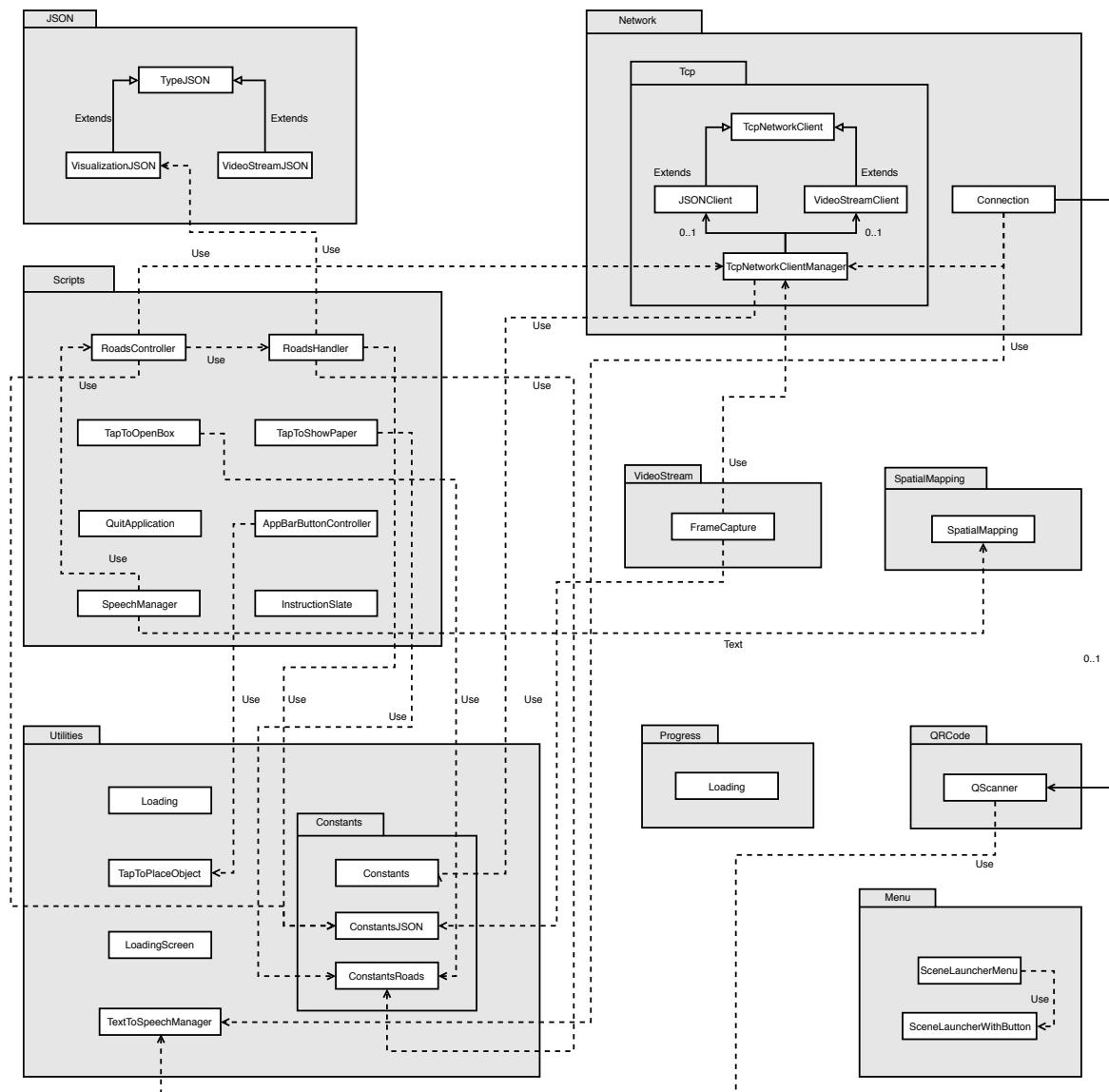


Figura F.2: Diagrama UML de la aplicación del cliente

Anexo G

Autoría de imágenes

El presente anexo incluye las fuentes de cada una de las imágenes utilizadas a lo largo del trabajo, con el fin de no distraer al lector mostrando elementos ajenos al contexto de este documento.

- **Figura 1.1:** obtenida de [MK94].
- **Figura 1.2:** obtenida de [mix18].
- **Figura 4.1:** obtenida de [VHP17].
- **Figura 4.2:** obtenida de <https://tinyurl.com/ycjs3a92>.
- **Figura 4.3:** obtenida de [KLH97].
- **Figura 4.4:** obtenida de [LBAU03].
- **Figura 4.5:** obtenida de [TC12].
- **Figura 4.6:** obtenida de [TCC17].
- **Figura 4.7:** obtenida de <https://tinyurl.com/y8h9ld78>.
- **Figura 4.8:** obtenida de <https://tinyurl.com/y9j36rvk>.
- **Figura 4.9:** obtenida de <https://tinyurl.com/y8rpc263>.
- **Figura 4.10:** obtenida de [Sut68].
- **Figura 4.11a:** obtenida de <https://tinyurl.com/y8uk2vea>.
- **Figura 4.11b:** obtenida <https://tinyurl.com/yaе47mm4>.
- **Figura 4.12a:** obtenida de <https://tinyurl.com/yb26hm6j>.
- **Figura 4.12b:** obtenida de <https://tinyurl.com/y6wmu82f>.
- **Figura 4.12c:** obtenida de <https://tinyurl.com/y7ube2f4>.
- **Figura 4.13:** obtenida de <https://tinyurl.com/yb68yjy5>.
- **Figura 5.1:** obtenida de <https://tinyurl.com/ycye7xgy>.
- **Figura D.1:** obtenida de <https://tinyurl.com/ydgryq7p>.
- **Figura E.1a:** obtenida de <https://tinyurl.com/y747pv63>.

- **Figura E.1b:** obtenida de <https://tinyurl.com/y747pv63>.
- **Figura E.2a:** obtenida de <https://tinyurl.com/y747pv63>.
- **Figura E.2b:** obtenida de <https://tinyurl.com/y747pv63>.

Anexo H

GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <<http://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version. 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

5. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

6. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

7. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

8. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

9. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

10. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

Referencias

- [Azu97] Ronald T Azuma. A survey of augmented reality. *Presence: Teleoperators & Virtual Environments*, 6(4):355–385, 1997.
- [B⁺56] Benjamin S Bloom et al. Taxonomy of educational objectives. vol. 1: Cognitive domain. *New York: McKay*, pages 20–24, 1956.
- [BKP01] Mark Billinghurst, Hirokazu Kato, and Ivan Poupyrev. The magicbook: a transitional ar interface. *Computers & Graphics*, 25(5):745–753, 2001.
- [BN84] Andrew Birrell and Bruce Jay Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems (TOCS)*, 2(1):39–59, 1984.
- [Bro88] Marc H Brown. *Algorithm animation*. MIT Press, Cambridge, 1988.
- [CB16] Julio Cabero and Julio Barroso. The educational possibilities of augmented reality. *Journal of New Approaches in Educational Research*, 5(1):44, 2016.
- [Coc08] Alistair Cockburn. Using both incremental and iterative development. *STSC CrossTalk (USAF Software Technology Support Center)*, 21(5):27–30, 2008.
- [CZ11] Pierre Caserta and Olivier Zendra. Visualization of the static aspects of software: A survey. *IEEE transactions on visualization and computer graphics*, 17(7):913–933, 2011.
- [DCP01] Wanda Dann, Stephen Cooper, and Randy Pausch. Using visualization to teach novices recursion. *ACM SIGCSE Bulletin*, 33(3):109–112, 2001.
- [DSEB15] Phil Diegmann, Manuel Schmidt, Sven Eynden, and Dirk Basten. Benefits of augmented reality in educational environments-a systematic literature review. *Wirtschaftsinformatik*, 3(6):1542–1556, 2015.
- [FMS93] Steven Feiner, Blair Macintyre, and Dorée Seligmann. Knowledge-based augmented reality. *Communications of the ACM*, 36(7):53–62, 1993.

- [FR14] Roy Fielding and Julian Reschke. Hypertext transfer protocol (http/1.1): Message syntax and routing. <https://tools.ietf.org/pdf/rfc7230.pdf>, 2014. Última consulta en 05/06/2018.
- [GVAC13] Carlos González, David Vallejo, Javier Albusac, and José Jesús Castro. *Realidad aumentada. Un enfoque práctico con ARTOolkit y Blender*. Identic, Ciudad Real, 2013.
- [HDS02] Christopher D Hundhausen, Sarah A Douglas, and John T Stasko. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing*, 13(3):259–290, 2002.
- [hol18] Hololens. <https://www.microsoft.com/es-es/p/microsoft-hololens-development-edition/8xf18pqz17ts?activetab=pivot:reviewstab>, 2018. Última consulta en 28/05/2018.
- [KB99] Hirokazu Kato and Mark Billinghurst. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *Proceeding of the 2nd IEEE and ACM International Workshop on Augmented Reality*, pages 85–94. IEEE, 1999.
- [KLH97] Charles Kann, Robert W Lindeman, and Rachelle Heller. Integrating algorithm animation into a learning environment. *Computers & Education*, 28(4):223–228, 1997.
- [KMT⁺17] Pooya Khaloo, Mehran Maghoumi, Eugene Taranta, David Bettner, and Joseph Laviola. Code park: A new 3d code visualization tool. In *Proceedings of the 5th IEEE Working Conference on Software Visualization for Understanding and Analysis*, pages 43–53. IEEE, 2017.
- [LBAU03] Ronit Ben-Bassat Levy, Mordechai Ben-Ari, and Pekka A Uronen. The jeliot 2000 program animation system. *Computers & Education*, 40(1):1–15, 2003.
- [LGH⁺96] Mark A Livingston, William F Garrett, Gentaro Hirota, Mary C Whitton, Etta D Pisano, Henry Fuchs, et al. Technologies for augmented reality systems: Realizing ultrasound-guided needle biopsies. In *Proceedings of the 23rd annual conference on computer graphics and interactive techniques*, pages 439–446. ACM, 1996.
- [LPPV17] Faraón Llorens Largo, Francisco José García Peñalvo, Xavier Molero Prieto, and Eduardo Vendrell Vidal. La enseñanza de la informática, la programación y el pensamiento computacional en los estudios preuniversitarios. *Education in the Knowledge Society (EKS)*, 18(2):7–17, 2017.

- [mix18] Mixed reality. https://developer.microsoft.com/en-us/windows/mixed-reality/mixed_reality, 2018. Última consulta en 22/05/2018.
- [MK94] Paul Milgram and Fumio Kishino. A taxonomy of mixed reality visual displays. *IEICE TRANSACTIONS on Information and Systems*, 77(12):1321–1329, 1994.
- [MTUK95] Paul Milgram, Haruo Takemura, Akira Utsumi, and Fumio Kishino. Augmented reality: A class of displays on the reality-virtuality continuum. In *Telemanipulator and telepresence technologies*, volume 2351, pages 282–293. International Society for Optics and Photonics, 1995.
- [Mye90] Brad A Myers. Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing*, 1(1):97–123, 1990.
- [MZC12] Carlo Dal Mutto, Pietro Zanuttigh, and Guido M Cortelazzo. *Time-of-flight cameras and microsoft kinect (TM)*. Springer Publishing Company, Incorporated, New York, 2012.
- [Pad82] MA Padlipsky. Perspective on the arpanet reference model. <https://tools.ietf.org/pdf/rfc871.pdf>, 1982. Última consulta en 05/06/2018.
- [PBS93] Blaine A Price, Ronald M Baecker, and Ian S Small. A principled taxonomy of software visualization. *Journal of Visual Languages & Computing*, 4(3):211–266, 1993.
- [PCGA15] Anasol Peña, Vic Callaghan, Michael Gardner, and Mohammed J Alhaddad. Experiments with collaborative blended-reality laboratory technology for distance learners. 2015.
- [Ros93] Louis B Rosenberg. Virtual fixtures: Perceptual tools for telerobotic manipulation. In *Virtual Reality Annual International Symposium*, pages 76–82. IEEE, 1993.
- [RSKM98] Dirk Reiners, Didier Stricker, Gudrun Klinker, and Stefan Müller. Augmented reality for construction tasks: Doorlock assembly. *Proc. IEEE and ACM IWAR*, 98(1):31–46, 1998.
- [SGG⁺18] Santiago Sánchez, Carlos González, María Ángeles García, David Vallejo, María del Carmen Lacave, Miguel Ángel Redondo, and Ana Isabel Molina. Applying mixed reality techniques for the visualization of programs and algorithms in a programming learning environment. In *Proceedings of the 10th International Conference on Mobile, Hybrid, On-line Learning*, pages 84–89. IARIA, 2018.

- [Sut63] Ivan Edward Sutherland. *SKETCHPAD-a man-machine graphical interface*. PhD thesis, PhD thesis, MIT, 1963.
- [Sut68] Ivan Edward Sutherland. A head-mounted three dimensional display. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 757–764. ACM, 1968.
- [TC12] Chin-Hung Teng and Jr-Yi Chen. An augmented reality environment for learning opengl programming. In *Proceedings of the 9th International Conference on Ubiquitous Intelligence & Computing and Autonomic & Trusted Computing (UIC/ATC)*, pages 996–1001. IEEE, 2012.
- [TCC17] Chin-Hung Teng, Jr-Yi Chen, and Zhi-Hong Chen. Impact of augmented reality on programming language learning: Efficiency and perception. *Journal of Educational Computing Research*, pages 254–271, 2017.
- [Tör09] Gábor Törley. Algorithm visualization in programming education. *Journal of applied multimedia*, 4(3):68–80, 2009.
- [TV04] Alexandru Telea and Lucian Voinea. A framework for interactive visualization of component-based software. In *Proceedings of the 30th Euromicro Conference*, pages 567–574. IEEE, 2004.
- [VF06] David Vallejo Fernández. Documentación de zeroICE. <http://www.esi.uclm.es/www/dvallejo/docs/ICE/docICE.pdf>, 2006. Última consulta en 04/06/2018.
- [VHP17] J Ángel Velázquez, Isidoro Hernán, and Maximiliano Paredes. Evaluating the effect of program visualization on student motivation. *IEEE Transactions on Education*, 60(3):238–245, 2017.
- [VKP10] Dirk Willem Van Krevelen and Ronald Poelman. A survey of augmented reality technologies, applications and limitations. *International journal of virtual reality*, 9(2):1, 2010.
- [Win71] Joel M Winett. Definition of a socket. <https://buildbot.tools.ietf.org/pdf/rfc147.pdf>, 1971. Última consulta en 05/06/2018.
- [WL07] Richard Wettel and Michele Lanza. Visualizing software systems as cities. In *Proceedings of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pages 92–99. IEEE, 2007.

Este documento fue editado y tipografiado con L^AT_EX empleando
la clase **esi-tfg** (versión 0.20180819) que se puede encontrar en:
https://bitbucket.org/arco_group/esi-tfg

