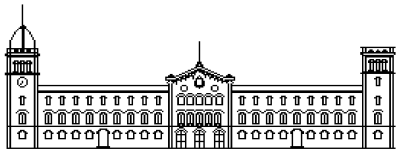

**Treball fi de carrera**

**GRAU D'ENGINYERIA INFORMÀTICA**

**Facultat de Matemàtiques**

**Universitat de Barcelona**

---

# Estudio y mejora de la usabilidad de un mundo virtual híbrido: interacción humano-agente

**Enric Mayas Márquez**

Director/s: Inmaculada Rodríguez

Pablo Almajano

Realitzat a: Departament de Matemàtica Aplicada i Anàlisi. UB

Barcelona, 20 de Junio de 2013

# RESUMEN

En la actualidad, con el auge de Internet y los avances en el campo de los gráficos en 3D los mundos virtuales han alcanzado una gran popularidad. Un mundo virtual es una simulación 3D que representa un mundo real o imaginario donde los usuarios, representados por avatares, interaccionan entre ellos y con el entorno. Estos entornos 3D abiertos gustan tanto por la inmersividad y la diversión que ofrecen como por el aspecto social que aportan al reunir a multitud de usuarios en un entorno común donde pueden participar por igual. Teniendo en cuenta no sólo su popularidad sino la multitud de usos prácticos que puede tener un mundo virtual, surgen los **mundos virtuales serios**. En éstos se pretende regular los comportamientos en el mundo virtual por tal de permitir que se desarrollen ciertas tareas, como pueden ser trámites relacionados con e-Government o e-Learning. En este trabajo hemos realizado un **estudio de usabilidad** de *v-mWater,* un mundo virtual serio e híbrido (compuesto tanto por humanos como por bots). Con los resultados que se han obtenido hemos implementado la mejora más prioritaria, y que ha consistido en utilizar **AIML** para conseguir que los bots de *v-mWater* puedan procesar lenguaje natural convirtiéndose así en **bots conversacionales**.

Nowadays, due to the increasing use of the Internet and the improvements in the field of 3D graphics virtual worlds have reached a high popularity. A virtual world is a 3D simulation that features a real or imaginary world where the users, represented by avatars, interact with each other and the environment. These open 3D environments are attractive for their immersiveness and the enjoyment they offer, as well as for the social aspect that they bring by joining multiple users together in a common environment and engaging them to participate together. Taking into account not only their popularity but the many practical uses a virtual world has, **serious virtual worlds** arise. Their aim is to regulate the behaviors in the virtual world in order to allow certain tasks to be done, which can be for example procedures related to e-Government or e-Learning. In this project we have conducted a **usability study** of *v-mWater,* a serious and hybrid (where both humans and bots interact) virtual world. With the results that have been obtained from the study, we have implemented the most beneficial improvement of *v-mWater*, that has consisted in using **AIML** to allow the bots of *v-mWater* to process natural language, turning them into **conversational bots.**

# Índice de contenido

# Índice de figuras

# Índice de tablas

# 1 Introducción

En la sociedad contemporánea se han adoptado diferentes formas de socialización online. Los denominados mundos virtuales sociales (*3D Social Virtual Worlds)* son una forma relativamente nueva de hacerlo. Son entornos virtuales donde los usuarios (representados por un avatar, su representación gráfica en el entorno) se socializan libremente en actividades y eventos [Book 2004]. Aún así, los entornos virtuales también pueden usarse para implicar a los usuarios en aplicaciones serias con una temática basada en la realidad (e-government, e-learning y e-commerce), son los llamados *Serious VWs (Virtual Worlds-VWs)*.

Los mundos virtuales sociales se conciben como entornos no estructurados que carecen de interacciones definidas y controladas, mientras que los *Serious VWs* se pueden ver como entornos estructurados donde la gente cumple con diferentes roles, y donde algunas actividades siguen protocolos bien definidos y normas que cumplen con objetivos específicos. Aunque los usuarios en dichos entornos estructurados se pueden sentir “controlados”, dado que parte de sus interacciones están controladas, a esto se le puede dar la vuelta y se pueden sentir más seguros y guiados gracias a estas regulaciones que les dirigen y coordinan sus complejas actividades. Ésto último es especialmente importante en los *Serious VWs*, donde las actividades son bastante complejas de por sí y además errores en el desarrollo de estas actividades pueden tener repercusiones graves.

Las plataformas de mundos virtuales actuales (i.e. Second Life, OpenSim), centradas principalmente en ofrecer experiencias sociales de carácter abierto, no consideran explícitamente la definición de interacciones estructuradas, ni tampoco contemplan su control en tiempo de ejecución. Por lo tanto, en éste proyecto se reivindica el uso de las Instituciones Virtuales [Bogdanovych07], las cuales combinan Instituciones Electrónicas [Esteva et al., 2004] y mundos virtuales, para diseñar entornos virtuales híbridos y estructurados.

Las Instituciones Electrónicas proporcionan una infraestructura para regular las interacciones de los usuarios. Concretamente, una Institución Electrónica es un Multi-Agent System (MAS) centrado en organizaciones que establece y regula las interacciones de los agentes participantes estableciendo una secuencia de acciones que los agentes pueden/se espera que hagan. Los mundos virtuales ofrecen una interfaz intuitiva que permite a los humanos estar al corriente del estados del MAS a la vez que participar de una manera transparente. Por híbrido se entiende que los participantes pueden ser tanto humanos como bots (avatares que representan agentes software). Ambos llevan a cabo interacciones complejas para alcanzar objetivos de la vida real (p. ej pagar facturas, asistir a un curso, comprar).

Hasta el momento no se han hecho evaluaciones sobre la usabilidad de una aplicación con una infraestructura similar al de una Institución Virtual (i.e. un entorno virtual fuertemente regulado e híbrido), que permita analizar cómo los usuarios perciben la estructuración (regulación) de ciertas actividades en el entorno y también cómo perciben la interacción con los bots. Este proyecto pretende indagar en estas líneas de trabajo no exploradas. Para ello consta de dos partes bien diferenciadas.

La primera parte es un análisis de usabilidad del prototipo *v-mWater*. *v-mWater* (a *v*irtual *m*arket based on trading *Water)* es un prototipo de una aplicación de e-Government. *v-mWater* es una institución virtual que implementa un mercado virtual del agua. Los resultados de esta primera parte han sido publicados en la conferencia GRAPP (International Conference on Computer Graphics Theory and Applications ) 2012.

La segunda parte se centra en implementar una mejora de *v-mWater* según los resultados obtenidos en el análisis de usabilidad mencionado anteriormente. Para ello se deberá integrar el código de dicha mejora en la plataforma que permite la ejecución de una institución virtual, *Virtual Institutions eXEcution Environment* (VIXEE) [Tescak et al., 2011].

A continuación se resumen el resto de apartados de la memoria de este proyecto fin de carrera. El apartado de *Conceptos relacionados* introduce conceptos y procesos relacionados con el ámbito del proyecto. A continuación se introducen los objetivos del proyecto. En el apartado *Antecedentes* se presentan trabajos relacionados con el proyecto. En el apartado *Prototipo v-mWater & infraestructura de ejecución* de ejecución se explica en detalle el entorno sobre el cual se va a realizar el proyecto. El apartado *Estudio de usabilidad de v-mWater* se explica la planificación y los resultados del test de usabilidad realizado. En el apartado *Lenguaje AIML* se introduce el lenguaje AIML así como para que se ha usado y de que forma. En los apartados *Análisis* y *Diseño* se describe con diagramas UML la mejora introducida en el prototipo, después de realizar la evaluación de usabilidad. En concreto, el diagrama de casos de uso, el modelo de dominio, y los diagramas de clases y de secuencia respectivamente. El apartado *Implementación* presenta detalles sobre la implementación final así como muestras de los resultados obtenidos. En el apartado *Conclusiones y trabajo futuro* se resume el trabajo realizado, sus logros y aportaciones. Por último los apéndices incluyen manuales de usuario, técnico y de instalación, así como el material que se ha utilizado en las sesiones de test con usuarios.

# 2 Conceptos relacionados

El desarrollo de este proyecto se engloba dentro de los campos de la Interacción Persona-Computador y de Inteligencia Artificial.

La Interacción Persona-Computador es la disciplina interesada en el diseño, evaluación y implementación de sistemas computacionales interactivos, para su uso por los humanos, y en el estudio de los fenómenos que les rodean. El objetivo es que la interacción sea más eficiente, minimizar errores, incrementar la satisfacción, disminuir la frustración y, en definitiva, hacer más productivas las tareas que rodean a las personas y los computadores. En concreto, la usabilidad se refiere a la facilidad con que las personas pueden utilizar una herramienta particular o cualquier otro objeto fabricado para su uso por humanos, con el fin de alcanzar un objetivo concreto.

Por otra parte, la Inteligencia Artificial (IA) es una rama de las ciencias computacionales que se dedica al estudio y diseño de programas de ordenador que intentan imitar la inteligencia humana. En concreto, este proyecto se focaliza en una rama de la IA, los sistemas multi-agente o MAS organizativos. En éstos se cuenta con múltiples agentes los cuales deben cooperar (o competir) por tal de conseguir sus objetivos y por tanto deben tomar consciencia de que existe una cierta sociedad. En concreto, las *Instituciones Electrónicas*, son una forma de implementar convenciones en cuanto a la interacción de agentes en un entorno abierto.

Las *Instituciones Electrónicas* están formadas por varios elementos. Los roles permiten predefinir las acciones que pueden realizar diferentes grupos de agentes, según a cual de estos *roles* se acojan. Los *protocolos* definen la estructura que deben seguir las interacciones entre los agentes por tal de que todos los agentes de la Institución Electrónica se puedan entender entre ellos. Las escenas definen las acciones a realizar en cada estado concreto. A lo largo de este proyecto se hará referencia a las *escenas* también por el nombre *actividades*. Por último, la *Performative Structure* (PS) define cómo se relacionan las diferentes escenas.

Las *Instituciones Virtuales* combinan una *Institución Electrónica* con un mundo virtual. De esta manera, se obtiene un entorno abierto 3D el cual esta regulado por una *Institución Electrónica,* y por lo tanto dentro del mundo virtual existirán protocolos y otras normas que decidirán que acciones tienen que consecuencias y en que ocasiones. Un ejemplo de *Institución Virtual* es *v-mWater*, prototipo sobre el cual se hará el estudio de usabilidad y donde se implementará la posterior mejora.

Por último se ha utilizado el Artificial Intelligence Markup Language [AIML]. AIML es un lenguaje compatible con XML y que permite de manera fácil crear bots conversacionales que utilizan lenguaje natural para

comunicarse con los humanos. Esto es, un humano podría mantener una conversación con un “bot inteligente”. Esto se consigue relacionando una serie de entradas (expresiones en lenguaje natural que proporcionaría el humano) con unas salidas o respuestas dadas por el agente software. Esto es, si en la base de conocimiento del bot existe un tag “pattern” que coincide con el mensaje introducido por el usuario, el bot devuelve una respuesta que de define con el tag “template”.

# 3 Objetivos

El objetivo de este proyecto es realizar una evaluación de usabilidad del prototipo *v-mWater* y implementar una mejora en dicho prototipo basada en los resultados de la evaluación con usuarios.

En concreto, la evaluación de usabilidad requiere una planificación exhaustiva del test:

- Identificar a los usuarios (*testers*) de la aplicación, reconociendo los diferentes grupos de usuarios y sus características.
- Buscar un grupo de voluntarios para realizar el testeo, intentando incluir integrantes de cada grupo de usuarios identificados, y establecer lugar, día y hora del test.
- Crear el documento de consentimiento del test, que tendrán que firmar los usuarios, y diseñar una encuesta de satisfacción que realizarán los usuarios después del test.
- Estudiar los aspectos relacionados con la usabilidad (objetivos de usabilidad) que se quieren analizar en el prototipo, por tal de diseñar un test que pueda dar información lo más completa posible sobre todos estos aspectos.
- Durante la realización de los tests se hará uso de herramientas adecuadas para la recolección y representación de los datos obtenidos.
- Analizar los datos obtenidos y sacar conclusiones sobre buenas propiedades y problemas de usabilidad del sistema.

Una vez analizados los resultados de la evaluación de usabilidad, se buscará implementar la mejora que se considere más prioritaria en el prototipo. Para ello se estudiará el código fuente del prototipo por tal de entender su funcionamiento.

La hipótesis que se baraja en este proyecto es que la mayor dificultad que encontraran los usuarios será en el proceso de interacción con los bots (agentes software). Si como resultado de los test, corroboramos esta hipótesis, el principal objetivo de la mejora será implementar procesamiento de lenguaje natural en los agentes de la institución.

# 4 Antecedentes

## 4.1 Trabajos relacionados

En esta sección se revisan trabajos previos aplicados a interacciones estructuradas en entornos Multi-Usuario/Agente y la evaluación de usabilidad en entornos virtuales. La regulación ha sido sujeto de estudio tanto en el campo de los sistemas multi-agente (MAS) como en el de la interacción persona-computador.

En el campo de los MAS, varios estudios se han centrado en sociedades de agentes y han propuesto metodologías e infraestructuras para regular y coordinar las interacciones de los agentes. [Dignum et al., 2002] [Esteva et al., 2004]. Concretamente, Cranefield et al. adaptó una herramienta, originariamente desarrollada para estructurar interacciones sociales entre agentes software, para modelar y hacer un seguimiento de las reglas de “expectativas sociales” en el mundo virtual Second Life (SL) tales como, por ejemplo, “nadie debería volar” [Cranefield and Li, 2009]. Usaron lógica temporal para implementar el sistema regulativo. En nuestro caso, se utilizan Instituciones Electrónicas, un ejemplo bien conocido de Organization Centered MAS (OCMAS), para regular las interacciones de los participantes en entornos virtuales 3D híbridos.

Varias investigaciones en HCI (Human Computer Interaction) se han centrado en mecanismos de regulación para aplicaciones groupware, i.e. CSCW (Computer Supported Collaborative Work). En general, estos mecanismos definen roles, actividades y métodos de interacción para aplicaciones colaborativas. Un trabajo previo ha usado reglas sociales (y las condiciones para ejecutarlas) para controlar las interacciones entre los miembros de un grupo de trabajo [Mezura-Godoy and Tal-bot, 2001]. Otro trabajo ha propuesto mecanismos de regulación para tratar aspectos sociales del trabajo colaborativo tales como la localización donde la actividad toma lugar, actividades colaborativas en lo referente a escenarios, y a los propios participantes [Ferraris and Martel, 2000]. A nivel conceptual el modelo de regulación utilizado por las Instituciones Electrónicas, esto es, actividades, protocolos y roles, comparte similitudes con aquellos aplicados para aplicaciones groupware.

Relacionado con las regulaciones de actividades en entornos virtuales, Paredes [Paredes and Martins, 2007] propuso el modelo Social Theatres. Este modelo regula las interacciones sociales en un entorno virtual basándose en el concepto de *teatro* (un espacio donde actores interpretan papeles y siguen un *workflow* de interacciones bien definido y regulado por una serie de reglas). En trabajos posteriores, llevaron a cabo una encuesta para evaluar las preferencias de los usuarios sobre las interfaces utilizadas. Ésto permitió diseñar una interfaz 3D basada en el modelo de

Social Theatres y las preferencias de los usuarios [Guerra et al., 2008]. Recientemente han propuesto una arquitectura software multi-capa implementando el modelo Social Theatres [Paredes and Martis, 2010]. Aunque se ha diseñado para ser adaptable, esta arquitectura presenta algunas limitaciones en la adaptación dinámica de reglas. Por el contrario, las Instituciones Virtuales heredan propiedades de auto-adaptación de las Instituciones Electrónicas [Trescak et al., 2011]. Otra diferencia clave entre las Virtual Institutions y el modelo Social Theatres es que este último se basa en un entorno web (dependiendo de servicios web), mientras que las primeras son independientes de la tecnología que implementa el entorno virtual.

En lo que se refiere a análisis de usabilidad de entornos virtuales, Bowman et al. analizo una serie de aspectos tales como el entorno físico, el usuario, el evaluador y el tipo de evaluación de usabilidad, y propuso una nueva clasificación para las evaluaciones: evaluación secuencial y *testbed*. La evaluación secuencial incluye evaluaciones heurísticas, formativas/exploratorias y aditivas, se hace en el contexto de una aplicación particular y puede tener resultados tanto cualitativos como cuantitativos. Un *testbed* se realiza en un contexto evaluativo mas genérico, y normalmente tiene resultados cuantitativos obtenidos mediante la creación de *testbeds* los cuales implican todos los aspectos importantes de la tarea en cuanto a interacción. [Bowman et al., 2002a] [Bowman et al., 1999]. Hay varios investigadores que han propuesto diferentes frameworks de evaluación para entornos virtuales colaborativos [Tsiatsos et al., 2010] [Tromp et al., 2003]. El método que se seguirá en este proyecto es el de evaluación secuencial, principalmente formativo ya que se ha observado que los usuarios interactuan en este entorno híbrido, pero también sumativo ya que se toman algunas medidas de tiempo y errores a la hora de realizar las tareas.

En cuanto a la parte de implementación, existen trabajos con AIML que han creado extensas librerías con diferentes grupos de vocabulario por tal de equipar a un bot (mediante el chat de una página web) con lenguaje natural [AIML]. Otro ejemplo sería Pandorabots [Pandorabots], un servicio web que nos permite crear nuestros propios bots conversacionales, para luego incluirlos por ejemplo en nuestra páginas web. Existe incluso la posibilidad de incluir estos bots en Second Life [Pandorabots in Second Life]. También hay aproximaciones para integrar el lenguaje AIML a distintos lenguajes de programación, un ejemplo es Program# [AIMLbot] que permite usar lenguaje AIML en el lenguaje C# mediante la creación de clases que se ocupan del procesado de los mensajes. Program# se mencionará mas adelante, ya que la librería AIMLbot.dll se ha obtenido de allí.

## 4.2 Conclusión

Existen numerosos trabajos previos tanto en el campo de entornos estructurados poblados por agentes como en el campo de evaluación de usabilidad de entornos virtuales pero ninguno de ellos proporciona información sobre la experiencia de los usuarios en este tipo de sistemas. En este proyecto se propone realizar un estudio de usabilidad de *v-mWater*, un sistema híbrido, poblado por humanos y agentes software, y así obtener datos que ayuden a proponer mejoras en el sistema.

Tanto la parte de análisis de usabilidad como la de implementación de mejoras permitirán mejorar notablemente el prototipo, y en general obtener resultados sobre la usabilidad y el interés por parte de los usuarios en aplicar mundos virtuales al e-Goverment (lo cual se puede extrapolar en cierta medida a cualquier otro Serious Virtual World). La parte de implementación también sera aplicable a otras *Instituciones Virtuales* que quieran adquirir sus funcionalidades, ya que se va a diseñar e implementar pensando en que sea reusable y portable.

# 5 Prototipo *v-mWater* & infraestructura de ejecución

## 5.1 Introducción a v-mWater

*v-mWater* es un mercado virtual del agua. Es una simplificación de m-Water [Giret et al., 2011] implementado como una institución virtual la cual modela un mercado electrónico de derechos del agua en el dominio de la agricultura [Almajano et al., 2012].

En nuestro mercado, los participantes negocian con derechos del agua. En un contexto de agricultura, *un derecho del agua* se refiere al derecho de un regante a usar agua de fuentes de agua públicas (p.ej. un pantano). Esta asociado a tierras de cultivo y el volumen de riego se especifica en metros cúbicos ($m^3$).

Al principio de la *época de riego*, las autoridades públicas estiman las reservas de agua y asignan cierta cantidad de agua por cada *derecho del agua.* Los *derechos de agua comerciables* contienen la cantidad sobrante de agua que los regantes esperan tener y deciden venderlos. Un *acuerdo* es el resultado de una negociación donde un vendedor acuerda con un comprador traspasar parte de su agua durante un periodo de tiempo acotado a cambio de cierta cantidad de dinero.

Nuestro mercado permite la entrada a regantes propietarios de *derechos del agua* en la cuenca hidrográfica (i.e. tierras de cultivo).

## 5.2 Especificación de las interacciones

Se utiliza una *Institución Electrónica* para estructurar las interacciones de los participantes en el entorno virtual. Una *Institución Electrónica* esta definida por los siguientes componentes: una *ontologia,* la cual especifica conceptos de dominio; los *roles* que pueden adoptar los participantes; varias *dialogic activities,* las cuales agrupan las interacciones de los participantes; protocolos *que definen las interacciones que se realizan en las dialogic activities*; y una *performative structure* que define los movimientos legales de los diferentes *roles* entre *activities*. Más concretamente, una *performative structure* se especifica como un grafo de *nodos* que representan tanto *activities* como *transiciones* y están unidos por *arcos* dirigidos etiquetados con los roles que tienen permiso para realizarlas.

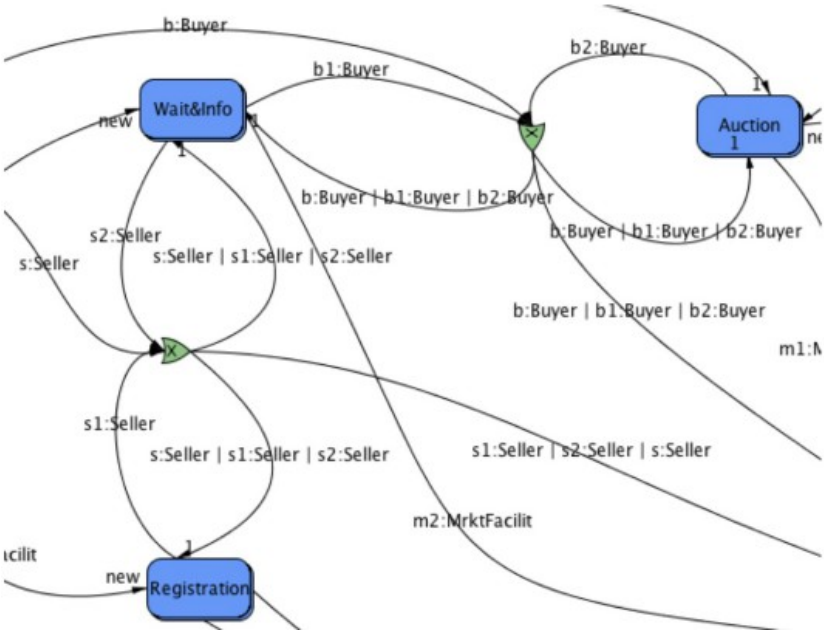


*Figura 1: Extracto de la performative structure de v-mWater. Las cajas representan actividades (Wait&Info, Registration, Auction1) y las puertas lógicas representan transiciones.*

En la *ontologia* de nuestro escenario del mercado del agua se incluyen conceptos tales como *derecho del agua, tierra* o *agreement.* Además, los participantes (tanto agentes software como humanos) pueden desempeñar diferentes roles. Por lo tanto, un *comprador* representa un adquisidor de derechos del agua, un *vendedor* es un mercader de derechos del agua, un *market facilitator* es responsable de cada actividad del mercado, un *basin authority* corresponde a la entidad legal la cual valida los *acuerdos*, y un *institution manager* está al cargo de controlar el acceso al mercado.

Para acceder a la institución, un agente debe “logearse” dando su nombre y el rol que quiera desempeñar. Los agentes que se hayan logeado con éxito se colocan en una actividad inicial por defecto. Desde esta actividad, los agentes en *v-mWater* pueden unirse a tres actividades diferentes (ver la *performative structure* de la Figura 1): en la actividad *Registration* se registran los derechos del agua, para ser negociados posteriormente; en la actividad *Waiting and Information*, los participantes intercambian impresiones sobre el mercado y obtienen información sobre las negociaciones; y por último, la negociación por los derechos del agua tiene lugar en la actividad *Auction*. La subasta sigue un protocolo de subasta japonés multi-unidad, un protocolo de precio ascendiente que toma como precio de salida el precio registrado por el vendedor. Luego, los compradores hacen pujas mientras sigan interesados en adquirir los derechos del agua al precio que tengan actualmente.

Los participantes y la especificación de elementos de una *Institución Electrónica* tienen sus correspondientes representaciones (visuales) en el entorno virtual 3D. A modo de ejemplo, los participantes se representan mediante avatares mientras que las actividades se representan mediante salas con puertas por tal de mantener el acceso a ellas controlado (ver Figura 2). La siguiente sección se centra en la infraestructura que permite ejecutar un entorno virtual 3D estructurado.


*Figura 2: Vista aérea inicial de v-mWater, con las tres salas (actividades) visibles.*

## 5.3 Infraestructura

Hemos usado VIXEE, *Virtual Institutions eXEcution Environment* [Trescak et al., 2013], una infraestructura robusta para conectar una *Institución Electrónica* a diferentes mundos virtuales. Eso permite validar las interacciones de los mundos virtuales las cuales tienen significado a nivel institucional (i.e. se contemplan en las especificación de la Institución Electrónica), y actualizar los estados de ambos mundos virtuales y Institución Electrónica para mantener una dependencia casual. También contempla la manipulación dinámica de contenido de mundo virtual.

La arquitectura de VIXEE consiste en 3 capas: la capa de control normativo, la capa de interacción visual y la de conexión casual (también referida como *middleware* en este documento), todas ellas mostradas en la Figura 3. También se proporciona una descripción del flujo de comunicación el cual permite las interacciones humano-agente.

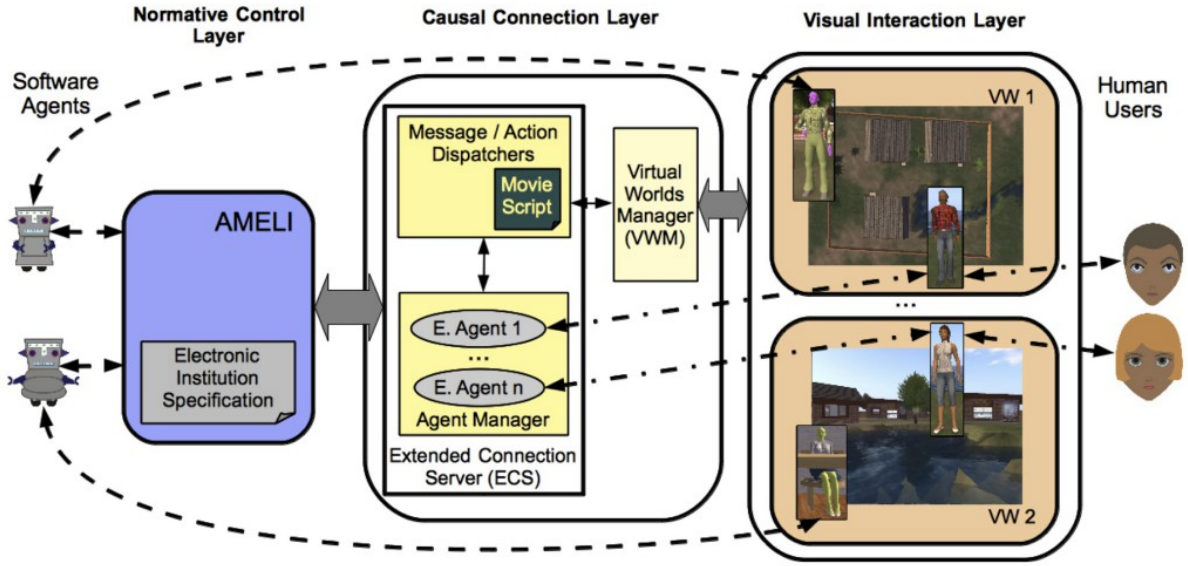


*Figura 3: Arquitectura VIXEE. La Casual Connection Layer se muestra como un middleware entre la Normative Control Layer (poblada por agentes) y la Visual Interaction Layer (poblada por personajes 3D virtuales).*

La *Normative Control Layer* a la izquierda de la Figura 3 se encarga de estructurar las interacciones. Esta compuesta por una especificación de la Institución Electrónica y AMELI [Esteva et al., 2004], el motor de ejecución de las EIs. Para generar una *especificación de la Institución Electrónica* se usa ISLANDER [Esteva et al., 2002], un editor de especificaciones de Instituciones Electrónicas que facilita la tarea. Luego, AMELI interpreta la especificación por tal de mediar y coordinar la participación de cada agente dentro del MAS. Los agentes software (iconos con aspecto de robot en la izquierda de la Figura 3) tienen una conexión directa con AMELI, el cual tiene una comunicación bidireccional con el middleware.

La *Visual Interaction Layer* representa varios mundos virtuales 3D. Los usuarios humanos (iconos con caras de persona en la derecha de la Figura 3) participan en el sistema por medio del control de avatares (i.e. personajes virtuales 3D) que les representan en el entorno virtual. Además, los agentes software del Normative Control Layer se pueden visualizar como bots en el mundo virtual (las flechas en la Figura 3 conectan los iconos de robot en la izquierda con los personajes bot dentro de esta capa).

La *Casual Connection Layer* -o middleware- constituye el componente principal de VIXEE (ver Figura 3). Casual connection se refiere a una relación estado-consistencia, tal que los cambios en el estado se propagan en ambos sentidos de la comunicación. En un sentido de comunicación, conecta los humanos participantes de diferentes mundos virtuales a la *Normative Control Layer* con el objetivo de regular sus acciones. En la comunicación inversa, permite la visualización de los agentes software

participantes como bots en los mundos virtuales (representando su presencia así como sus acciones y sus consecuencias). Esta capa está dividida entre Extended Connection Server (ECS) y un Virtual World Manager (VWM).

El ECS (caja izquierda del middleware en la Figura 3) media todas las comunicaciones con AMELI. Permite la conexión de múltiples mundos virtuales a una Institución Electrónica. De esta manera, los usuarios de diferentes mundos virtuales pueden participar juntos en la misma institución virtual. Además, la ECS es capaz de capturar los mensajes AMELI que activan la generación del entorno 3D inicial (p.ej. construir habitaciones y su amueblado) y resetear el mundo a un estado predefinido (p.ej. resetear los paneles de información o cerrar todas las puertas). Los principales elementos del ECS son el *Agent Manager* y los *Message/Action Dispatchers.*

El *Agent Manager* crea un External Agent (E. Agent en la Figura 3) por cada avatar conectado (controlado por un humano). El external agent esta conectado a la Institución Electrónica con el propósito de traducir las interacciones producidas por el humano en el mundo virtual. Por lo tanto, AMELI percibe a todos los integrantes como agentes software.

Los *Message/Action Dispatchers* (encima de *Agent Manager* en la Figura 3) se hacen cargo de los mensajes AMELI y las acciones del mundo virtual. Utilizan un mecanismo llamado *Movie Script* por tal de definir como se mapean los mensajes AMELI a acciones mundo virtual, y viceversa. Por un lado, un mensaje generado por AMELI provoca una acción mundo virtual por tal de que la visualización de todos los mundos virtuales conectados se actualicen. Por otra parte, por cada acción institucional provocada por un avatar humano en el mundo virtual (regulado por la Institución Electrónica), un dispatcher envía el correspondiente mensaje AMELI a través de un agente externo. Cabe mencionar que es en este módulo de VIXEE donde se debería integrar el código necesario para implementar un bot conversacional en una Institución Virtual.

El VWM (caja derecha dentro de middleware en la Figura 3) se hace cargo de las comunicaciones entre mundos virtuales y ECS y actualiza de manera dinámica las representaciones 3D de todos los mundos virtuales conectados. Esto se hace a través de *Message/Action dispatchers*. Por un lado, el VWM recibe acciones concretas (correspondientes a mensajes AMELI) del *Message dispatcher* para actualizar el contenido de los mundos virtuales de manera acorde. Por otro parte, VWM filtra las acciones del mundo virtual y envía aquellas con valor institucional para el *Action dispatcher* que las va a comunicar y, si es necesario, actualiza AMELI.

El VWM se compone de un proxy para cada mundo virtual conectado. Ya que diferentes plataformas de mundo virtual pueden requerir lenguajes de programación específicos, estos proxies permiten usar dicho lenguaje para comunicarse con el ECS. En nuestro escenario de muestra usamos la librería de *OpenMetaverse* [OpenMetaverse, 2012] para manipular el contenido de *OpenSimulator,* un servidor de mundos virtuales 3D.

# 6 Estudio de usabilidad de v-mWater

A continuación se describe el proceso seguido para evaluar *v-mWater*, un entorno virtual estructurado e híbrido. Se evaluará a partir de un *test plan* que ha sido ampliamente usado [Rubin and Chisnell, 2008]. Por lo tanto, primero se definirán los objetivos generales del test y luego se derivaran cuestiones de investigación (research questions) de éstos. A continuación, se detallarán los participantes del test y la metodología a seguir durante éste. Por último, se describen y se analizan los resultados obtenidos.

## 6.1 Objetivos del test

El principal objetivo del test de usabilidad es determinar la utilidad de nuestro entorno virtual, esto es, el grado en el que los usuarios pueden alcanzar sus objetivos y su voluntad de usar un sistema de este tipo para tareas de e-goverment. Este objetivo se puede dividir en los siguientes sub-objetivos:

- Determinar la *efectividad* de *v-mWater*, i.e. en qué medida su interfaz 3D se comporta como los usuarios esperan y la facilidad con la que los usuarios pueden usarla para hacer aquello que pretenden.
- Determinar la eficiencia de v-mWater, i.e. la rapidez con que los objetivos de los usuarios pueden cumplir sus objetivos de manera precisa y completa.
- Identificar *problemas/errores que los usuarios encuentran/cometen* mientras están inmersos en este entorno virtual 3D híbrido y normativo (estructurado).
- Determinar *la satisfacción de los usuarios*, i.e. sus opiniones, sensaciones y experiencias.
- Generar un cierto debate sobre la hipótesis de que las *características de los usuarios* edad, genero y habilidad (en el manejo de ordenadores así como con entornos virtuales) podría afectar tanto la *habilidad de alcanzar su objetivo como la experiencia d*el usuario.

Con todos estos objetivos en mente, hemos definido una tarea para el test que consiste en buscar información sobre las últimas transacciones en el mercado y luego registrar un derecho del agua para su venta. Esta tarea estructurada esta compuesta de cuatro subtareas que involucran algunas de las actividades descritas en la sección 5.1:

i. *Entender la tarea y deducir el plan* (dos de las tres habitaciones deben ser visitadas en un orden específico) requeridos para realizar la tarea.

ii. *Obtener información específica sobre las transacciones del mercado* en la habitación *Waiting and Information*. Esto se puede lograr mirando el panel de información o sino preguntando al bot de Información que se encuentra en el mostrador.

iii. *Resolver el precio de registro correcto,* el cual debe ser 5€ mayor que el precio de la transacción más reciente.

iv. Registrar el derecho del agua en la habitación Registration, hablando con el Registration bot, que se encuentra en el mostrador de esta misma habitación.

## 6.2 Cuestiones de investigación

Siendo v-mWater un prototipo funcional, hemos querido contestar algunos interrogantes relacionados con cómo de usable es, cómo de útil resulta ser este entorno virtual para diferentes usuarios, y de manera mas genérica, la disposición del usuario para realizar tareas de e-Government en entornos virtuales.

Dados los objetivos del test dados en la sección anterior, proponemos una serie de cuestiones de investigación que derivan de éstos. Estas preguntas están dividas en dos categorías. La primera categoría esta estrechamente relacionada con la tarea que se les plantea a los usuarios en el entorno virtual:

RQ1: Obtención de la información ¿Con que rapidez encuentra el usuario la información que necesita una vez entra a la sala *Waiting and Information?* ¿Era dicha información fácil de interpretar? ¿Cómo ha obtenido el usuario la información? (leyendo el panel o bien interactuando con el agente).

RQ2: Interacción humano-bot ¿Es el panel de registro (y el bot) fácil de encontrar? ¿Como de complaciente ha sido la interacción con el bot? ¿Valora el usuario el hecho de poder distinguir entre bots y humanos?

RQ3: Cumplimiento de la tarea ¿Que obstáculos encuentran los vendedores de camino a la sala *Registration* dentro del entorno virtual? ¿Que errores cometen a la hora de registrar su derecho del agua? ¿Cuantos usuarios completaron la tarea?

Las respuestas a las preguntas de esta primera categoría nos dan datos sobre la efectividad, eficiencia y errores haciendo tareas concretas en el entorno. La segunda categoría es mas general y se centra en la habilidad

del usuario y sus estrategias para moverse por el entorno 3D, la *aprendibilidad* para usuarios novatos, y la utilidad percibida así como la disposición a usar *entornos virtuales* para procedimientos de e-Government online:

**RQ4: Influencia del perfil de usuario** ¿Influencia el perfil de usuario (edad, genero y experiencia con ordenadores y entornos virtuales) en la dificultad que se percibe en la realización de la tarea, la satisfacción del usuario y su inmersividad?

**RQ5: Navegabilidad del entorno virtual** ¿Que estrategia emplea el usuario para desplazarse entre habitaciones? ¿Se da cuenta (y utiliza) el usuario de que existe una función de teleportarse? ¿Incluso si la descubre, prefiere el usuario moverse a pie y investigar el espacio 3D?

RQ6: Aplicabilidad al e-Government ¿Que piensan los usuarios de las aplicaciones de e-Government 3D después del test? ¿Las usarían en un futuro?

## 6.3 Participantes

Hemos reclutado diez participantes. Representan un espectro amplio de la población en lo que se refiere a las características de edad, genero, habilidad con los ordenadores y experiencia en entornos/juegos 3D. Las edades de los participantes van de 18 a 54 años, dentro de este grupo encontramos a usuarios que han crecido con los ordenadores y usuarios que no, por lo tanto podemos estudiar si la edad influencia la eficiencia, la facilidad que aparenta el entorno virtual, su utilidad y la predisposición a utilizar dicho entorno virtual 3D híbrido para tareas relacionadas con e-Government. También prestamos especial atención a las habilidades con ordenadores y a la experiencia en entornos virtuales 3D por parte de los usuarios, ya que puede influenciar su habilidad para realizar las tareas que se les piden.

La Tabla 1 muestra detalles sobre la edad, el genero, las habilidades con el ordenador ('básico', 'medio', 'avanzado') y la experiencia en entornos virtuales/juegos ('ninguna', 'poca', 'mucha') por parte de los usuarios.

La clasificación para las habilidades con ordenadores era: 'basic' para participantes que utilizan el ordenador únicamente para las cosas mas básicas, tales como navegar, editar texto, etc.; 'medium' para los usuarios con un mínimo conocimiento del funcionamiento interno del ordenador los cuales lo utilizan de manera mas compleja para cosas como jugar; y 'advanced' para participantes los cuales trabajan de manera profesional con ordenadores, i.e. programadores.

En cuanto a las habilidades en entornos virtuales, 'none' para los

usuarios que nunca han usado un entorno virtual, 'some' para usuarios que lo han probado en alguna ocasión, y 'high' para usuarios que utilizan a menudo un entorno virtual. Cabe destacar que aunque la mayoría de habilidades están uniformemente distribuidas, la experiencia con entornos virtuales esta fuertemente inclinada hacia los novatos en entornos virtuales.

| Name | Age | Gender | PC exp | VE exp |
|---|---|---|---|---|
| P1 | 18 | Female | Medium | Some |
| P2 | 19 | Female | Medium | High |
| P3 | 20 | Male | Advanced | Some |
| P4 | 25 | Female | Medium | None |
| P5 | 25 | Female | Medium | None |
| P6 | 28 | Female | Advanced | None |
| P7 | 39 | Male | Advanced | None |
| P8 | 40 | Male | Medium | None |
| P9 | 53 | Male | Basic | None |
| P10 | 54 | Female | Basic | None |

*Tabla 1: Lista de los participantes clasificados según edad (age) , genero (gender) y experiencia (exp).*

## 6.4 Metodología

El estudio de usabilidad que se ha llevado a cabo era principalmente *exploratorio*, pero también *sumatorio*. Hemos usado el método de Formative Evaluation [Bowman et al., 2002b], el cual encaja con nuestros intereses en el estado del prototipo (además, es el primer test que realizamos con *v-mWater*) y estamos interesados principalmente en encontrar datos relevantes y cualitativos sobre la usabilidad del sistema. Aun así, ya que la aplicación en sí es un prototipo ya funcional, también hemos tenido en consideración algunas medidas cuantitativas.

El equipo evaluador se ha compuesto de un moderador y un observador. El primero ha guiado al usuario cuando este lo ha necesitado, animándole a pensar en voz alta, introduciendo la sesión de test, y dando al usuario cualquier material que necesitara [Error: No se encuentra la fuente de referencia, Error: No se encuentra la fuente de referencia] así como material para el propio equipo de test [Apéndice E: Guión del moderador]. El observador tomaba las notas que fueran necesarias durante la sesión de test.

El test ha tenido lugar en lugares familiares al usuario: la mitad de los participantes lo hicieron en su casa y la otra mitad en su lugar de trabajo,

en una habitación separada. El equipo requerido ha sido dos ordenadores, uno que actuaba como servidor de mundos virtuales y el segundo, desde el cual el usuario realizaba el test, actuando solamente como cliente de mundos virtuales y grabando tanto la pantalla como el sonido durante la tarea.

A todos los participantes se les ha pedido realizar la tarea descrita anteriormente. En concreto, se les ha dicho: *"Actúa como vendedor, y registra un derecho del agua 5€ mas alto que el precio de la última transacción que se haya realzado".* Cabe recordar que, para realizar la tarea correctamente, los participantes deberían visitar la sala *Waiting and Information,* comprobar el precio de la última transacción (preguntando al bot llamado "Information Manager" o comprobando dicha información en un panel de información situado en una de las paredes de la sala), y luego dirigirse hacia la sala *Registration* y registrar un derecho del agua al precio correcto interactuando con el bot Registration.

El protocolo del test ha consistido en cuatro fases:

1. *Entrevista pre-test:* Le hemos dado la bienvenida al usuario, introduciéndole al test y preguntándole preguntas relacionadas con su experiencia con (y opinión sobre) el e-Government.
2. *Entrenamiento: El* usuario ha "participado" en una sesión previa al test, para enseñarle a interactuar con los objetos y los avatares, enseñarle a distinguir usuarios humanos de bots, interactuar con los bots en concreto (chateando mediante un sistema de comandos). Este entrenamiento ha sido totalmente guiado, ya que el moderador ha explicado al usuario qué hacer a cada paso de la demo, excepto al final, cuando el usuario ha sido libre de curiosear con la demo hasta que elija empezar con la tarea (saliendo el escenario de la demo).
3. *Test:* El usuario ha realizado la tarea del test sin recibir asistencia a no ser que se haya quedado sin recursos. Mientras, el moderador le ha animado a pensar en voz alta (le ha sugerido que describa sus acciones y pensamientos mientras hacía el test).
4. *Cuestionario post-test:* Al usuario se le ha dado un cuestionario con preguntas sobre *v-mWater* y también sobre la aplicación de los entornos virtuales a tareas de e-Government. (véase la Tabla 2 y la Figura 4).

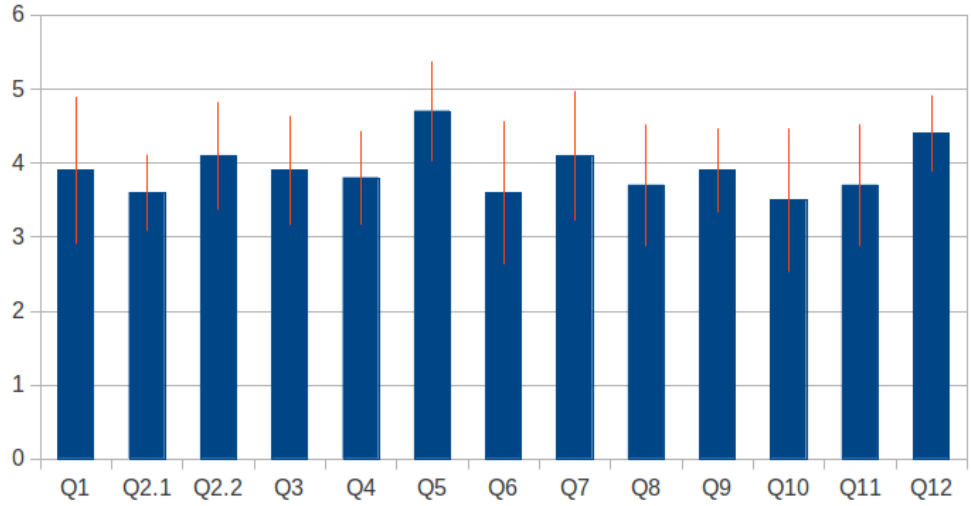


*Figura 4: Resultados del cuestionario post-test. Eje X: preguntas de la Tabla 2. Eje Y: media (y desviación estándar)*

| Question Number | Brief description |
|---|---|
| Q1 | Situatedness and movement in 3D |
| Q2 (Q2.1, Q2.2) | VE walking (2.1) and teleport (2.2) comfortability |
| Q3 | Info gathering (panel/bot) |
| Q4 | Human-bot interaction |
| Q5 | Bot visual distinction |
| Q6 | Chat-based bot communication |
| Q7 | Task easiness |
| Q8 | Immersiveness in 3D |
| Q9 | Improved opinion of 3D VWs |
| Q10 | Likeliness of future usage |
| Q11 | 3D interface usefulness |
| Q12 | Overall system opinion |
| open question | User's comments |

*Tabla 2: Cuestionario post-test*

## 6.5 Resultados y discusión

En esta sección se discuten las cuestiones de usabilidad identificadas una vez hecho el análisis de los datos obtenidos durante el test. Se hará un repaso a las cuestiones de investigación definidas en la sección 6.2. Las respuestas a cada una de ellas vienen de diferentes fuentes: una combinación del cuestionario post-test; comentarios hechos por los

usuarios; notas tomadas por el observador; y la revisión de las grabaciones tomadas durante el test tanto de voz como de pantalla, esto es, mientras los participantes realizaban la tarea.

La Tabla 2 resume las doce preguntas del cuestionario post-test, mientras que Gráfico 1 muestra las respuestas de los usuarios. Aquí, el eje X representa cada una de las preguntas del cuestionario post-test y el eje Y muestra la media de las respuestas considerando la escala de cinco puntos de *Likert*. Esta escala proporciona cinco diferentes alternativas respecto al éxito de la aplicación ('muy mala'/'mala'/'normal'/'buena'/'muy buena'), donde 'muy mala' corresponde a 1, y 'muy buena' a 5. Se muestra también la desviación estándar.

En general, los resultados cuantitativos obtenidos de los cuestionarios han sido muy satisfactorios, siendo todas las medias obtenidas mayores que 3.5 (con desviación estándar menor de 1.0). Las respuestas con mejor puntuación (con valores mayores de 4.0) se han obtenido de las preguntas sobre la clara distinción entre bots y personajes controlados por humanos (Q5) y la satisfacción del usuario (Q12). Por otra parte, las preguntas con respuestas más bajas (con valores de 3.5) corresponden a la comodidad a la hora de caminar por el entorno (Q2.1), el sistema de comandos usado para chatear con los bots (Q6), y la idea de usar un entorno virtual 3D para realizar tareas similares a la del test (Q10).

Tanto de las medidas cualitativas que el usuario dio en la pregunta abierta (de comentarios) del cuestionario post-test como cuando se ha contrastado con el equipo de evaluación, se han extraído un número de aspectos relevantes del sistema *v-mWater.*

Primero, a los usuarios les gusta su aprendibilidad, su inmersividad, y como el escenario facilita el completar la tarea. Además, a los usuarios les gusta la visualización 3D aunque la idea de usar entornos virtuales para tareas del día a día les parece una idea un tanto futurista ya que les cuesta imaginárselo, no tienen costumbre de usarlos, y en algunos casos no confiarían totalmente en estos métodos. Así mismo, la opinión general del sistema era positiva y algunos usuarios aclararon que no se encontraban del todo cómodos con la aplicación, pero que se acostumbrarían fácilmente a ella, ya que era muy aprendible y segura de usar.

Los criterios de usabilidad, tales como la efectividad, eficiencia y los errores se han analizado contestando las cuestiones de investigación de la primera categoría introducida en la sección 6.2.

### RQ1: Obtención de la información

La información que el usuario debía obtener en la subtarea ii) se podía obtener de dos formas: el panel de información y el bot Information Manager, ambos localizados en la sala *Waiting and Information.* Durante el test, la mayoría de los usuarios, excepto dos

de ellos que no entraron en la sala, caminaron directamente hacia el panel de información y/o el mostrador (donde se encontraba el bot). Dichos usuarios pudieron leer con facilidad la información de ambas fuentes. Las respuestas a Q1 y Q3, ambas con una media cercana a 4, refuerzan esta afirmación.

### RQ2: Interacción humano-bot

Los usuarios deberían interactuar con bots en diferentes subtareas (ii y iv). La alta media de Q4 indica que el usuario tenia una buena impresión general sobre la interacción humano-bot. Aun así, Q6 denota que los usuarios no estaban cómodos con la manera de hacerlo, un sistema basado en comandos que se ha usado para dialogar con el bot. Analizando Q5, con una media de 4.7, podemos decir que los participantes lo encontraron prácticamente imprescindible saber cuándo se encontraban delante de un bot.

### RQ3: Cumplimiento de la tarea

En general, los participantes encontraron fácil completar la tarea (tal y como indica Q7 con una media de 4), y les llevó una media de 4.46 minutos. Los usuarios no han encontrado ningún obstáculo que les impidiera completar la tarea. Hablando de los errores, algunos usuarios se equivocaban de dirección en ocasiones (iban a una sala que no era), pero siempre se daban cuenta de su error y eran capaces de ir a la sala correcta.

Otro tipo de error se relaciona con la interacción basada en chat con los bots; como Q6 indica, donde la media es de 3.6. Los usuarios con poca experiencia con ordenadores tuvieron ciertos problemas para interactuar con los bots a causa del estricto sistema de comandos. Aun así, los usuarios encontraron este sistema de comunicación altamente aprendible (aunque no evidente). En relación con la efectividad de la aplicación se ha medido revisando las grabaciones de pantalla. Considerando la estructura de la tarea detallada en la sección 6.1, el porcentaje de usuarios que completaron las subtareas fueron:

i. Un 80% entendieron bien la tarea. Sólo el 20% de los usuarios no fueron conscientes de que debían comprobar los precios antes de registrar su derecho del agua.
ii. El 80% de los usuarios obtuvieron la información correctamente (el resto se saltaron este paso, tal y como se ha mencionado en i)).
iii. El 70% de los usuarios calcularon el precio correctamente.
iv. Un 100% de los usuarios completaron la subtarea de registrarse, esto es, todos registraron su derecho del agua.

A continuación damos una breve discusión a cerca de la influencia del perfil de los usuarios en cuanto a la dificultad percibida de la tarea, la satisfacción, la utilidad y la inmersividad, y analizamos aspectos de usabilidad mas generales de nuestro sistema tales como la habilidad de los usuarios para moverse por el entorno 3D; o la utilidad percibida de los entornos virtuales para procedimientos online de e-Government.

### RQ4: Influencia del perfil de usuario

Esta pregunta se ha contestado analizando los resultados del cuestionario post-test en lo que se refiere a las características de cada usuario. Desde el punto de vista de la edad, los participantes estaban bien balanceados. A medida que la edad sube también lo hace la dificultad a la hora de usar la aplicación, aunque la satisfacción también se incrementa. Sorprendentemente, los usuarios mas jóvenes encontraron la aplicación menos útil que los más mayores (esto puede ser dado por sus expectativas mas altas de un entorno virtual 3D). En relación con la experiencia de los usuarios con los ordenadores, los usuarios con la experiencia mas baja lo tuvieron claramente mas difícil para usar los controles direccionales a la hora de moverse por el espacio 3D. Además, este grupo de usuarios encontró difícil tanto la interacción con los *bots* como el completar la tarea. De manera similar, la inmersión crece a medida que lo hace la experiencia con ordenadores.

### RQ5: Navegabilidad del entorno virtual

Moverse por nuestro entorno virtual ha demostrado ser relativamente fácil, ya que la opinión media de los usuarios ha sido de 4 (Q1). No se aprecia en las grabaciones que se dedicaran a deambular en ninguna ocasión. Los usuarios que se dieron cuenta de que se podían *teleportar* lo encontraron cómodo, y lo reflejaron en el cuestionario post-test (Q2.2) y también en algunos de sus comentarios.

### RQ6: Aplicabilidad al e-Government

La opinión de los usuarios sobre los entornos virtuales ha mejorado después de la realización del test (Q9), ya que respondieron con una media de 4. Cuando se les preguntaba sobre su disponibilidad a la hora de utilizar un sistema similar con fines parecidos en un futuro (Q10), los usuarios respondieron con una media de 3.5, lo cual indica que tienen una opinión relativamente buena sobre la utilidad de la aplicación. Por último, los usuarios indicaron que la interfaz 3D les ha ayudado a alcanzar sus objetivos durante el test, tal y como muestra Q11 con una media de 4.

## 6.6 Conclusión

Los resultados de la evaluación de usabilidad han provisto un feeback realmente útil sobre *v-mWater*. *v-mWater se* percibe como una aplicación útil y poderosa que podría facilitar tareas diarias en un futuro. A los usuarios les ha gustado su aprendibilidad, su inmersividad y como el escenario y sus elementos han facilitado el completar tareas. En general, los usuarios han completado correctamente la tarea que se les ha propuesto y han sido capaces de dirigirse a los lugares acertados en el escenario. Después del test, los usuarios han visto su opinión mejorada respecto los entornos virtuales 3D. Además, la opinión en general sobre las interacciones humano-bot son positivas.

Aun así, se han detectado algunas limitaciones intrínsecas en cuanto a las interfaces de dialogo y sus interacciones. Varios usuarios no se han mostrado cómodos con el sistema de dialogo con los bots basado en comandos. Por lo tanto, la mejora prioritaria que se puede realizar al prototipo apunta hacia definir nuevas formas de interacción humano-bot, usando lenguaje natural y técnicas multi-modales basadas en feedback por voz, sonido o táctil las cuales soportan los dispositivos orientados a jugar.

## 6.7 Estudio de la mejora a implementar

Como se ha dicho en el apartado 6.6, la principal necesidad que destaca el estudio de usabilidad es la de incorporar un nuevo sistema de interacción con los bots, que reemplace al sistema de comandos actual.

Lo que ha parecido “molestar” principalmente a los usuarios no ha sido el hecho de usar el “chat” sino tener que hablar de una manera pre-definida (comandos). Los usuarios olvidaban que debían introducir el comando “help” para poder saber que servicios ofrecía el bot y cómo acceder a ellos. Se muestra un ejemplo a continuación, en la Figura 5:


*Figura 5: Tester 1 le pregunta a Registration Manager cuales son sus comandos.*

Luego han aparecido problemas con el formato del comando, es decir el formato del mensaje que se le envía al bot ya que el mas mínimo error en este caso por ejemplo invalidaba el registro de los derechos del agua y el usuario no siempre era consciente de porqué había introducido mal el comando. Por ejemplo, hubo un usuario que introdujo el comando “regiter wr1 90”. Al ver que no había respuesta lo intento de nuevo, miró la ayuda otra vez y siguió intentándolo. No fue hasta unos minutos mas tarde que, casi frustrado, se percató de que había puesto “regiter” en lugar de “register”.

Otros usuarios hablaban a los bots usando lenguaje natural aunque se les había indicado en el entrenamiento previo al test que a los bots siempre se les debía “saludar” con el comando “help”.

Por lo tanto vemos dos grandes problemas con el sistema de comandos. El primero, una estructura demasiado rígida a la hora de interactuar con el bot. La segunda, una falta de feedback al usuario en muchas ocasiones.

La solución que consideramos más efectiva y eficiente reside en **implementar procesamiento de lenguaje natural** en los bots. Con ésto, no sólo serán mas abiertos a la hora de recibir mensajes, sino que podrán dar respuestas más acordes a cada situación. El sistema idóneo para hacerlo es “convertir” los actuales bots en “bots conversacionales”, esto es, bots que pueden mantener una conversación con los usuarios humanos. Para ello se propone utilizar **AIML** (Artificial Inteligence Markup Language).

En las siguientes secciones se introduce el lenguaje AIML y se describe cómo se ha integrado la funcionalidad de *bots* conversaciones (AIML) en el prototipo *v-mWater*.

# 7 Lenguaje AIML

Tal y como se introdujo en la sección 2, el lenguaje AIML es un lenguaje compatible con XML que permite de manera fácil crear bots conversacionales que utilizan lenguaje natural para comunicarse con los humanos.

AIMLbot.dll es una librería de código abierto obtenida de Program# [AIMLbot] que permite utilizar el lenguaje AIML desde código C#. La manera en que permite hacerlo es proporcionando una serie de clases con métodos que facilitan el procesamiento de mensajes mediante AIML. Las clases *Bot*, *Request*, *Result* y *User* son las más importantes y las que se van a usar de manera directa a la hora de implementar la mejora de *v-mWater*. La clase *Bot* contiene mantiene una referencia al conocimiento del bot conversacional (esto es, las rutas a los ficheros AIML) así como las conversaciones (almacenadas) que se van manteniendo con él. La clase *User* identifica a un usuario concreto que manda mensajes a un *Bot* específico. Estos mensajes del *User* para el *Bot* se envían mediante un *Request.* Como respuesta a ese *Request,* el *Bot* genera un *Result* que contiene la respuesta del bot.

Para ello, los tags <pattern> y <template> relacionan un mensaje de entrada del usuario y uno de salida del bot conversacional, respectivamente. Estos dos tags se agrupan por parejas dentro de los tags <category>, que representan una unidad de conocimiento del bot conversacional. A continuación, en la Figura 6 vemos un ejemplo del flujo normal durante una conversación entre el humano y el bot conversacional:

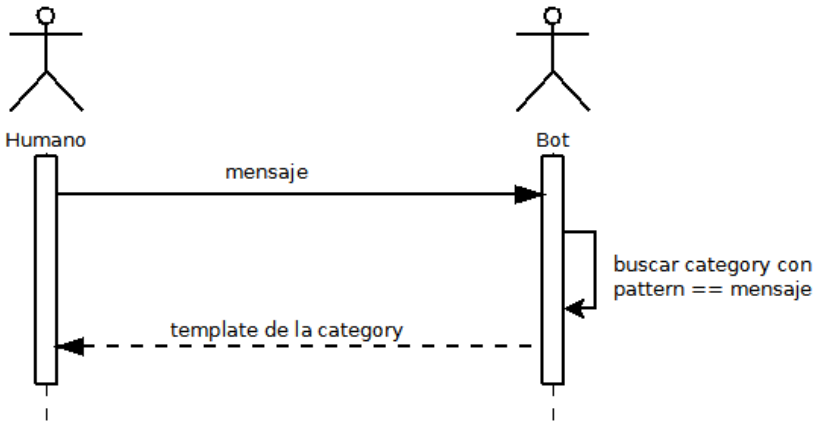


*Figura 6: Esquema de la interacción entre humano y bot conversacional (AIML).*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<aiml version="1.0">
<category>
        <pattern> ¿Fumas?</pattern> <!-- mensaje del humano -->
        <template>No, la verdad.</template>           <!--
respuesta del bot -->
</category>
</aiml>
```

El carácter * es especialmente importante ya que permite destacar palabras a buscar en una frase independientemente del resto. Por ejemplo, un *pattern* "* vecino *" corresponde tanto a las frases "Hola vecino, ¿qué tal?" como "mi vecino es de Burgos".

En este proyecto, el carácter * sera usado con gran frecuencia por tal de buscar frases que contengan palabras concretas provenientes del protocolo de las escenas/actividades que forman la *Performative Structure* de una Institución Electrónica. Por ejemplo, en la Figura 7 podemos ver el protocolo de la actividad de registro de *v-mWater*. Vemos también que para poder pasar del estado inicial (W0) al siguiente estado (W1) se debe hacer un *request* de nombre "register" y proporcionando los datos "quantity", "idWR", "idArea", y "price". Los recuadros amarillos encima de cada estado indican los roles que participan, marcando con + aquellos que pueden entrar a ese estado del protocolo de la escena y con – aquellos que pueden salir.

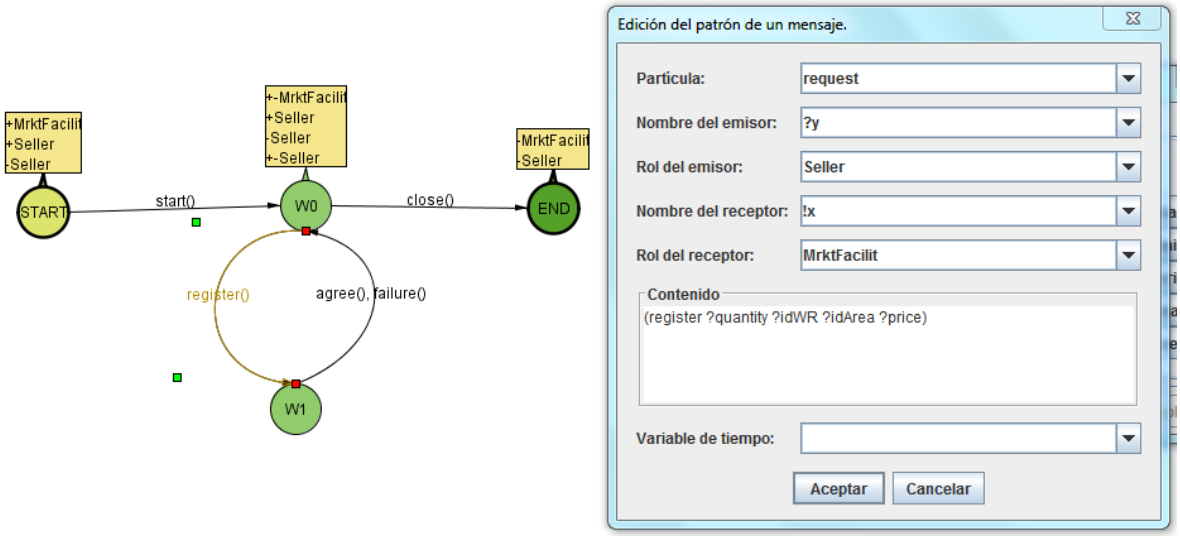


*Figura 7: Ejemplo en el que se visualiza el arco para pasar de W0 a W1*

Gracias a esta información sabemos pues que en la actividad de registro el usuario (actuando como *Seller*), va a hacer una petición "register" al bot llamado *MrktFacilit* con los datos que hemos señalado.

El tag <that> sirve para especificar la última respuesta que ha debido dar el "bot" para que se corresponda con el *pattern* especificado. Por

ejemplo: Supongamos que tenemos el *pattern* “Tengo 22 años” que contiene un *that* “¿Cuantos años tienes?”. Para que se active, el usuario deberá decirle al bot “Tengo 22 años” justo después de que el bot le haya dicho “¿Cuantos años tienes?” y entonces el bot responderá al usuario Ah! Pues aún eres joven... ¿estás estudiando?.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<aiml version="1.0">
<category>
<pattern> Tengo 22 años </pattern>  <!-- mensaje del humano -->
<that> Cuantos años tienes? </that>
<template> Ah! Pues aún eres joven... ¿estás estudiando?
</template>                         <!-- respuesta del bot -->
</category>
</aiml>
```

En este proyecto el tag <that> nos será útil para poder construir cadenas de mensajes cuando el usuario pida algo a un agente que requiera un proceso, como por ejemplo el de pedirle datos.

De manera similar, el tag <topic> permite condicionar las respuestas que da AIML según el tema (topic) de que se esté hablando en la conversación. Entonces, podemos agrupar las diferentes *category* en los *topic* que elijamos y así poder dirigir con más facilidad una conversación hacia un punto u otro.

Existen también tags más avanzados en AIML que permiten ejecutar otros archivos. Un ejemplo es el tag <system>, que nos permite ejecutar comandos en consola de comandos desde AIML. Este tag se consideró como una posibilidad para permitir integrar ficheros AIML con el prototipo de *v-mWater* directamente, aunque la idea fue descartada porque por consola de comandos no se podía intercambiar información en tiempo de ejecución (entre el fichero AIML y VIXEE).

# 8 Análisis

## 8.1 Diagrama de casos de uso

En la Figura 8 podemos ver los casos de uso del sistema, necesarios para que el usuario sea capaz de llegar a la actividad en que se beneficiará de la mejora implementada, es decir podrá comunicarse con un bot conversacional. El usuario, una vez en la actividad, se puede encontrar en dos situaciones a la hora de hablar con el bot. Entonces se pueden dar dos situaciones. En la primera situación el usuario es libre de hablar usando lenguaje natural con el agente y preguntarle aquello que desee, en la segunda debe introducir algún dato en un formato concreto como por ejemplo cuando debe especificar el precio al que quiere vender su derecho el agua (en este caso el formato es numérico).

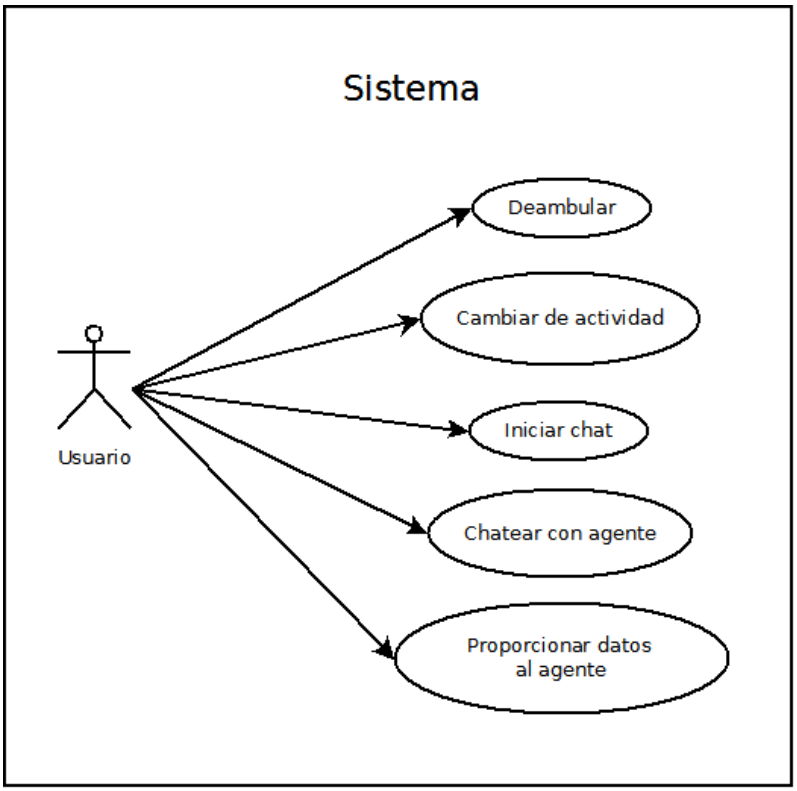


*Figura 8: Diagrama de casos de uso*

## 8.2 Casos de uso

### UC1-Deambular

En este caso de uso el usuario, ya logeado en el mundo virtual, se desplaza por éste. Lo puede hacer de varias maneras: andando, volando o teleportandose. Tanto para andar como para volar deberá usar las teclas correctas, según la configuración de teclado que tenga en su visor. Para teleportarse puede o bien hacer clic en el minimapa que incorpora el visor o bien usar uno de los carteles que hay en *v-mWater* y que disponen de un mapa. En ellos puede hacer clic a cada una de las tres salas/actividades para teleportarse frente a su puerta.

| |
|---|
| **Nombre:** UC1-Deambular |
| **Actores:** Usuario |
| **Descripción:** El usuario se mueve por el mundo virtual. |
| **Precondiciones:**<br>El usuario se encuentra en el mundo virtual. |
| **Flujo Normal:**<br>1. El usuario camina por el mundo usando el cursor. |
| **Flujo Alternativo:**<br>1a El usuario hace clic en el minimapa.<br>1b El usuario hace clic en uno de los carteles con mapa.<br>1c El usuario vuela por el mundo usando el cursor. |
| **Poscondiciones:**<br>El usuario ha llegado a su destino. |

## UC2-Cambiar de actividad

En este caso de uso el usuario tiene la intención de cambiar de actividad. Para ello, en nuestro entorno debe clicar la puerta de la sala en la que se encuentre o a la que quiera acceder. En caso de que pueda acceder ya que su rol tiene permisos, la puerta se abrirá para indicarle que se le ha permitido el paso. En pocos segundos el usuario será teleportado dentro de la actividad. En caso contrario, la puerta no se abrirá y el usuario será informado por la Institución Electrónica de que se le ha denegado el acceso a causa de su rol.

| |
|---|
| **Nombre:** UC2-Cambiar de actividad |
| **Actores:** Usuario |
| **Descripción:** El usuario cambia de actividad |
| **Precondiciones:**<br>El usuario tiene a su alcance una puerta que clicar. |
| **Flujo Normal:**<br>1. El usuario hace clic sobre una puerta.<br>2. La puerta se abre.<br>3. El usuario es teleportado dentro de la nueva actividad. |
| **Flujo Alternativo:**<br>2a. La puerta no se abre.<br>3a. El usuario es informado por la Institución Electrónica de que no tiene permiso para acceder a la actividad. |
| **Poscondiciones:**<br>El usuario está dentro de la actividad. |

## UC3-Iniciar chat

En este caso de uso el usuario inicia un chat con un agente. Para ello, hace clic derecho sobre él y selecciona la opción de chatear.

| |
|---|
| **Nombre:** UC4-Iniciar chat |
| **Actores:** Usuario, Agente |
| **Descripción:** El usuario abre una ventana de chat para un agente. |
| **Precondiciones:**<br>El agente se encuentra alcance del usuario. |
| **Flujo Normal:**<br>1. El usuario hace clic derecho sobre el agente.<br>2. El sistema muestra el menú de interacción.<br>3. El usuario hace clic sobre la opción de chatear. Ir a UC4 o UC5. |
| **Flujo Alternativo:** |
| **Poscondiciones:**<br>La ventana de chat se ha abierto. |

## UC4-Chatear con agente

El usuario chatea con el agente sin haberle dicho nada previamente o sin haberle solicitado ningún servicio en el cual el agente necesite ciertos datos del usuario (por lo tanto el agente se encuentra en un estado inicial). Si el usuario intenta chatear con un agente pero no se encuentra en la misma actividad que el agente (por ejemplo le habla a través de una ventana), a causa de las normas que rigen la Institución Electrónica el agente ignorará la petición del usuario y éste sera informado de la situación. Si el usuario le ha pedido al agente que realice un servicio, empezará por pedirle el primer dato que necesite (UC5). Si no ha ocurrido ninguna de las anteriores, el agente contestará (por ejemplo saludando al usuario de vuelta si éste le ha saludado) y el caso de uso volverá a empezar.

| |
|---|
| **Nombre:** UC4-Chatear con agente |
| **Actores:** Usuario, Agente |
| **Descripción:** El usuario escribe por chat al agente. |
| **Precondiciones:**<br>El agente se encuentra en un estado inicial de conversación con el usuario, es decir no requiere que le proporcione ningún dato. |
| **Flujo Normal:**<br>1. El usuario teclea el texto que desee en la caja de texto del chat.<br>2. El usuario presiona enter para enviar el texto al agente.<br>3. El agente contesta al usuario mediante el chat.<br>4. Volver al paso 1. |
| **Flujo Alternativo:**<br>3a. El agente ignora al usuario.<br>4a. El usuario es informado por la Institución Electrónica de que no se encontraba en la misma actividad que el agente.<br>4b. El usuario ha solicitado un servicio y el agente le pide el primer dato. (Ir a UC5). |
| **Poscondiciones:**<br>El agente ha procesado el mensaje del usuario y ha contestado al usuario de manera acorde con éste.<br>El agente ha actualizado su estado si se le pide realizar algún servicio. |

## UC5- Proporcionar datos al agente

En este caso de uso el agente está realizando un servicio solicitado por el usuario, y necesita que se le suministre un dato para ello. Por ejemplo, si el usuario ha solicitado al agente que registre su derecho del agua, el agente le pedirá que le especifique cuál es el derecho del agua a registrar. En el caso de que el dato no tenga el formato y/o el valor correctos (p.ej número entero o texto, mayor que 0, etc), el agente le recordará al usuario que debería introducir el dato y se volverá al inicio de este caso de uso.

Si necesitara otro dato mas el agente lo pedirá, volviendo también a empezar este caso de uso. En el caso de que ya tenga todos los datos necesarios, el agente ejecutará el servicio con los datos proporcionados y volverá a un estado inicial de la conversación (por lo tanto el usuario volverá a UC4).

| |
|---|
| **Nombre:** UC5- Proporcionar datos al agente |
| **Actores:** Usuario, Agente |
| **Descripción:** El agente requiere que el usuario entre un dato en un formato concreto. |

| |
|---|
| **Precondiciones:**<br>El usuario ha solicitado un servicio al agente para el cual el agente debe pedir datos. |
| **Flujo Normal:**<br>1. El usuario teclea el dato que se le pide en la caja de texto.<br>2. El usuario presiona enter para enviar el dato al agente.<br>3. El agente informa al usuario de que se ha realizado con éxito el proceso. Ir al caso UC4. |
| **Flujo Alternativo:**<br>3a. El usuario ha introducido un dato incorrecto. El agente vuelve a pedir el dato al usuario, indicando el formato correcto. Ir a UC5.<br>3b. El agente pide otro dato más al usuario. Ir a UC5. |
| **Poscondiciones:**<br>El agente ha guardado el dato correctamente.<br>El agente ha actualizado su estado interno de la conversación.<br>El agente ha contestado al usuario correctamente. |

## 8.3 Definición del modelo de dominio

A continuación, en la Figura 9 se presenta el modelo de dominio que engloba los casos de uso en los que el usuario se va a encontrar así como los componentes internos implicados en la generación de las respuestas usando AIML por parte de los agentes del sistema. Cabe destacar que se están considerando conversaciones privadas, es decir que implican un solo usuario hablando con un solo agente. De haber sido de otra manera, la estructuración del código y sobretodo su implementación se habrían complicado.

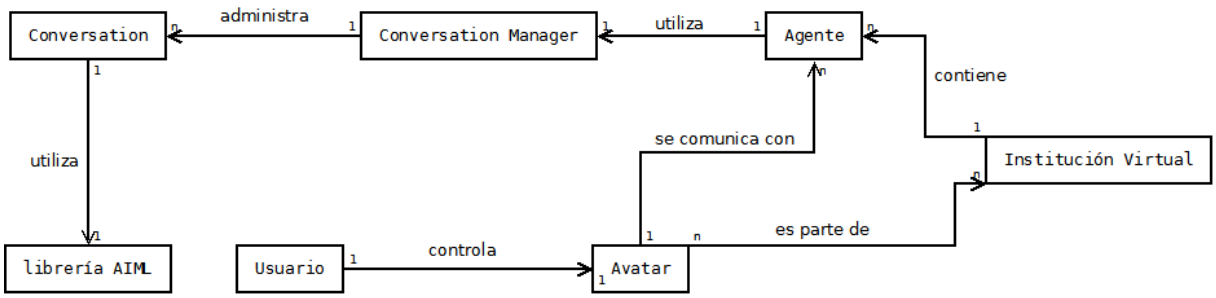


*Figura 9: Modelo de dominio*

## 8.4 Requerimientos generales: software y hardware

En cuanto a software, por parte del usuario se requiere tener instalado un cliente (i.e. visor de mundos virtuales), disponer de conexión a Internet. Los pasos a realizar por el usuario por tal de instalar el programa se detallan en Apéndice B: Manual de usuario.

En cuanto a hardware, los requisitos por parte del usuario serán los que especifique el visor que vaya a utilizar. Por parte del administrador, toda aquella infraestructura que necesite para poder arrancar tanto su Institución Electrónica como el mundo virtual, es decir su Institución Virtual.

# 9 Diseño

En este apartado se describe la solución conceptual que satisface los requisitos detallados en la sección anterior, se detallan diagramas UML de interacciones y de clase.

## 9.1 Diagramas de secuencia

El código a integrar en el sistema implementa una única secuencia posible, en la cual el usuario interacciona vía chat con un agente de la Institución Electrónica, y éste procesa una respuesta según determine el AIML. En caso de haberse solicitado un servicio por parte del usuario (p.ej. registrar) el programa necesitará construir un mensaje de AMELI a partir de los datos que le proporcione el usuario. InputEventAgentX representa aquella clase donde está el método que se activa cada vez que el agente

recibe un mensaje de un usuario. En concreto, en *v-mWater* InputEventAgentX corresponderá al método “Play” de uno de los *Moviescript* (ver el paso 4 de Apéndice A: Manual técnico (código fuente))*.* Cada agente tiene el suyo. El método *Invoke(method, parametres)* del diagrama hace una llamada al método *method*, con parámetros *parametres.* Este método corresponde a hacer el mapeo de la conversación mantenida entre el usuario y el bot a un mensaje de AMELI, la infraestructura de ejecución de las Instituciones Electrónicas. AMELI se encargará de comprobar si el mensaje es correcto y devolver una respuesta al mundo virtual vía el Casual Connection Server (ver sección 5.3). A continuación, vemos en la Figura 10 el diagrama de secuencia.

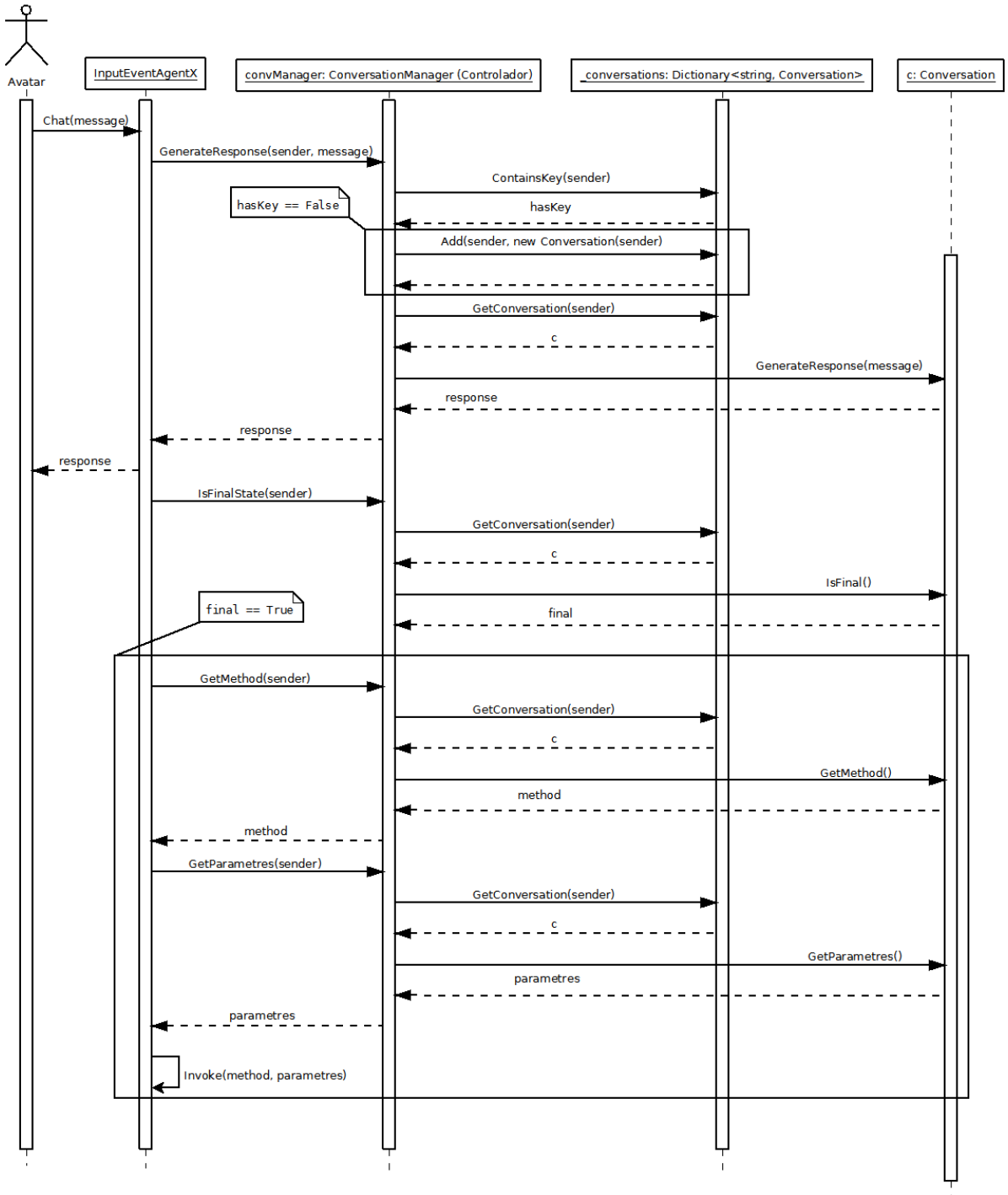


*Figura 10: Diagrama de secuencia donde el humano chatea con el bot conversacional*

## 9.2 Diagrama de clases

La mejora introducida en *v-mWater* consta de dos clases. *Conversation* representa una conversación mantenida por un usuario en concreto con un agente concreto de la Institución Electrónica. Ésta recuerda los intercambios de mensajes que se hayan producido entre el usuario y el agente y será la clase a la que se acuda para procesar las frases que dicho usuario vaya entrando. *ConversationManager,* por otra parte, mantiene todas las conversaciones pertenecientes a un agente en concreto. Por lo tanto, cada agente tendrá un *ConversationManager* propio y dentro de éste habrán una serie de *Conversation*. Es la herramienta que se sitúa entre el código de la Institución Electrónica y el procesado de AIML, haciendo de Controlador. De esta manera, conseguiremos que se puedan manipular las conversaciones de manera controlada. El diagrama de clases se muestra a continuación en la Figura 11:

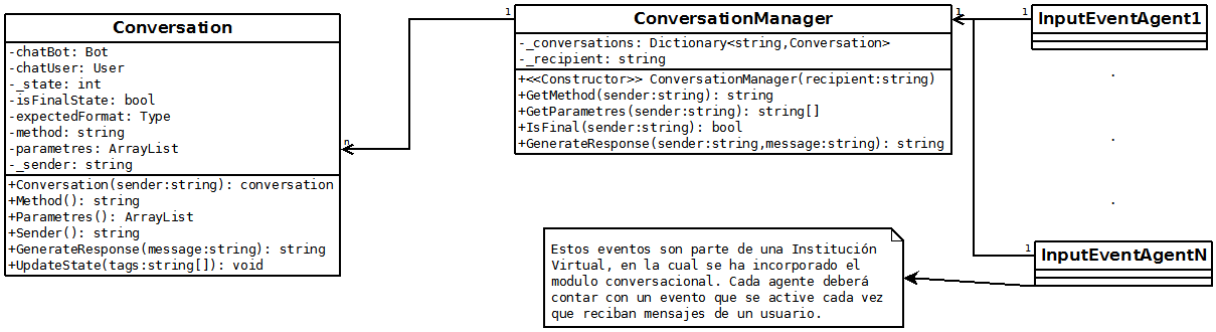


*Figura 11: Diagrama de clases*

# 10 Implementación

A continuación, en la Figura 12 se expone un diagrama de componentes de la aplicación resultante. Como podemos ver en el diagrama, los ficheros AIML deben estar en la misma ruta de VIXEE.exe ya que la librería AIMLbot.dll sólo permite usar AIML que estén en la ruta de la clase que se esté ejecutando.

La clase *Conversation* utiliza la librería *AIMLbot* para procesar los mensajes del usuario, y ésta accede a los ficheros AIML alojados en el directorio de ejecución. Vemos también que la parte de procesamiento en lenguaje natural (Conversational.dll) se conecta con los eventos de cada agente a la hora de recibir mensajes. Concretamente, estos eventos tienen acceso a la clase *ConversationManager* del agente del que se trate, y a través del *ConversationManager* el agente sera capaz de administrar sus *Conversation.* Los eventos forman parte de un *Moviescript* del CCS (Casual Connection Server, introducido en la sección 5.3), en particular de los *Action Dispatchers* que son llamados cuando suceden determinadas acciones en el mundo virtual.

Así pues, gracias a la librería *Conversational.dll* se extienden capacidades de los bots para que puedan tener conversaciones con humanos, añadiendo a los Action Dispatchers de VIXEE la capacidad de mapear una conversación entera (que es una serie de acciones en el mundo virtual) a un único mensaje de AMELI.

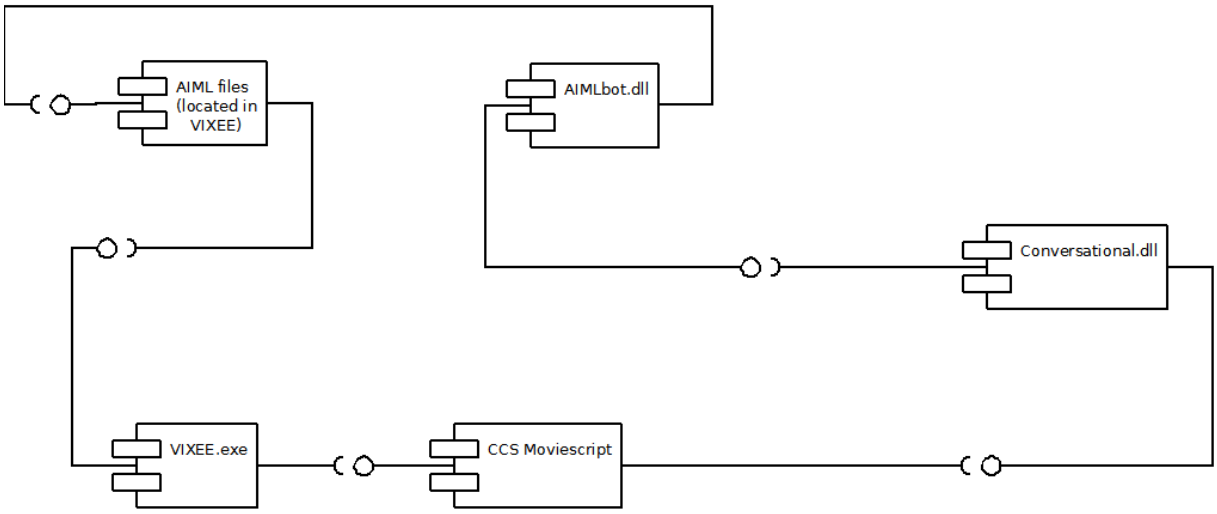


*Figura 12: Componentes del proyecto*

## 10.1 Integración de AIML

### El punto de partida: comunicación por comandos

En *v-mWater*, los bots interactuan con los usuarios mediante el chat. Para ello, utilizan un sistema comparable al de una consola de comandos. Por lo tanto, aceptan una serie de palabras puestas en un orden determinado. En caso de no recibir un mensaje entendible por el bot, no se le da respuesta ninguna al usuario.

Como se ha dicho en la sección 6, ésto ha dado problemas a la mayoría de usuarios que han realizado el test de usabilidad. De hecho, los únicos usuarios que no han tenido problemas con ello han sido los que usaban el ordenador a nivel profesional, es decir aquellos acostumbrados a usar sistemas de comandos (aún así algunos han cometido errores).

### El objetivo: comunicación por Lenguaje Natural (LN)

Se pretende proporcionar a los bots la capacidad de entender el contenido de los mensajes del usuario, y poder interpretar una misma intención por parte del usuario aunque este escrita de diferentes maneras. Por ejemplo, para entender que un usuario quiere hacer un registro el bot debería entender frases como “Vengo a registrar.”, “Quisiera registrar.” o “¿Como puedo registrar un derecho del agua?”.

Entendiendo la última frase incluso se consigue algo más allá de la mejora planteada, y es que el bot no sólo sea capaz de entender que quiere decirle el usuario sino que sepa interpretar sus intenciones finales. Si ante el mensaje “¿Como puedo registrar un derecho del agua?” le permitimos al usuario registrar directamente, no sólo le estaremos ahorrando tiempo sino que le facilitaremos las cosas entendiendo lo que pretende y guiándole así por el mejor camino hacia sus objetivos.

A nivel de AIML esto se ha podido conseguir simplificando los mensajes que los bots esperan obtener. Por ejemplo, en el bot de registro de *v-mWater* la manera de interpretar que el usuario quiere registrar es buscando la palabra “register” o “registration” esté donde esté. Por ejemplo, en la Figura 13 el usuario entra una frase que contiene “register”. Siendo así, independientemente del resto del mensaje, el bot conversacional le pide el ID de su derecho del agua.

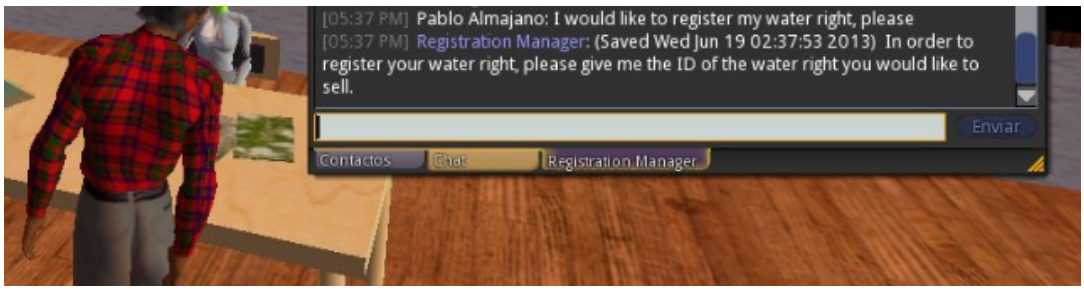


*Figura 13: El usuario le comunica (en lenguaje natural) al bot conversacional que quiere registrar. El bot inicia el proceso de registro pidiendo el ID del derecho del agua a registrar.*

```
<category>
<pattern>REGISTER</pattern>
<template>In order to register your water right, please give me
the ID of the water right you would like to sell.</template>
</category>

<category>
<pattern>* REGISTER *</pattern>
<template><srai> REGISTER </srai></template>
</category>

<category>
<pattern>* REGISTRATION *</pattern>
<template><srai> REGISTER </srai></template>
</category>
```

Es un procesamiento muy básico (p.ej. el mensaje “asdfghk register” serviría) pero que a la vez hace a los bots muy prácticos ya que si el usuario incluye la palabra “register” en su mensaje seguramente quiera registrar aunque no pregunte directamente por ello.

### Limitaciones de AIML

Hasta aquí hemos hablado de cosas para las cuales AIML es sobradamente flexible y capaz, ya que para poder conversar de cualquier tema sólo es necesario crear más y más AIML para cada tema. Gracias a tags como *that* y *topic,* cuando tengamos una base de conocimiento en AIML muy extensa podremos agrupar las diferentes *category* por temática y así evitaremos que un mensaje del usuario pueda coincidir con más de un *pattern.* Aún así hay una diferencia muy importante que distingue el caso en el que nos encontramos en *v-mWater* y el caso para el cual AIML está pensado y donde es una herramienta mas que suficiente.

En *v-mWater* encontramos casos en los que el usuario quiere pedir algún servicio al bot. Por ejemplo si el usuario se dirige al bot de registro, lo más

posible es que le quiera pedir registrar un derecho del agua. Es más, en los casos en que tengamos mundos virtuales destinados a e-Government (p.ej.), encontraremos bots a los cuales deberemos pedir servicios para realizar determinados trámites y de la misma forma no hablaremos con ellos de temas como los deportes o la música.

Por otra parte, AIML está pensado para generar bots conversacionales, con los cuales la gente pueda charlar de una amplia variedad de temas. Están pensados para ofrecer nada más que la conversación puramente dicha, y por lo tanto tienen una necesidad limitada de recordar aquello que les decimos. Este no es el caso de *v-mWater*, donde el usuario quiere registrar un derecho del agua, y necesitaremos pedirle los datos necesarios para ello de uno en uno y recordándolos, para que una vez los tengamos todos se pueda mapear la conversación (los diferentes mensajes con datos) a un mensaje que posteriormente sera enviado a AMELI.

Por desgracia, AIML es muy limitado en lo que a guardar variables se refiere. AIML tiene una serie de tags especiales en los cuales se pueden guardar valores. Estos tags van asociados a información sobre el usuario (p.ej. edad, sexo, nombre, ...). De esta manera AIML puede recordar la información sobre el usuario con quien esta hablando cuando el usuario se la proporcione, y a partir de ese momento utilizar esa información en las conversaciones. Por ejemplo, AIML después de preguntar "¿Cómo te llamas?" puede guardar el nombre del usuario una vez ha respondido para así poder decir "Encantado Jaime". Aun así esta funcionalidad no es suficiente para nuestras necesidades, ya que necesitamos tener un número no limitado de variables con nombres concretos para poderlas distinguir.

A modo de resumen, podemos decir que AIML está limitado en cuanto a: orden de las conversaciones, memoria de cara a guardar variables y formato a la hora de identificar el formato de los mensajes que se le pasan.

### Requerimientos de una solución genérica

Como hemos visto en el apartado anterior el bot no sólo debe dar una respuesta en lenguaje natural que sea entendible por el usuario sino que también debe guardar otras informaciones (i.e. si se ha solicitado un servicio como por ejemplo *registrar*, si se requieren datos, etc).

Para que las clases de *Conversational.dll* sean de carácter general y sirvan para cualquier bot el planteamiento es que tanto el responder en lenguaje natural como el tener consciencia del estado de la conversación con el usuario (i.e. si el usuario solicita algún servicio y cual, si se deben pedir datos, cuántos y cómo deben ser, etc) debe hacerse en AIML. Entonces, debe haber un AIML específico para cada bot que contendrá su vocabulario específico, ya que cada bot debería contestar de manera diferente al usuario según en que actividad se encuentre, o al menos

debería dar servicios al usuario que otros bots no pueden dar.

Por lo tanto, y partiendo de la premisa de que las clases de la librería Conversational.dll deben mantener su uso genérico para cualquier bot, hay que realizar ajustes en los ficheros AIML para que incluyan contenido adicional, esto es datos *de control*. Esto comporta que las respuestas que proporcione AIML, esto es los tag *pattern*, deberán contener información extra aparte de la respuesta en lenguaje natural.

Esto es, además de trabajar con *patterns* y *templates* que contienen la clásica pregunta del usuario y respuesta del bot:

```
<category>
<pattern>¿Hola bot, quien eres?</pattern>
<template>Hola, soy el Market Facilitator.</template>
</category>
```

Necesitaremos trabajar con *patterns* y *templates* de cierto modo "especiales" ya que incluyen en el *template*, además de la respuesta que se dará al usuario*, datos de control* para el módulo de procesamiento de *Conversational*. En el siguiente ejemplo "n|string" es la información de control y "In order to register your water right, please give me the ID of the water right you would like to sell." es la respuesta dada por el bot al usuario.

```
<category>
<pattern>REGISTER</pattern>
<template>n|string In order to register your water right,
please give me the ID of the water right you would like to
sell.</template>
</category>
```

Con esta información añadida, podremos realizar una verificación de formato así como un seguimiento del orden de las conversaciones sabiendo cuando acaban.

Como consecuencia surge un nuevo problema, y es que el usuario debería obtener sólo las respuestas al mensaje que haya enviado, y no esos *datos de control* que hay al inicio del *template.* La solución es procesar el *template* para eliminar esos *datos de control*, ver en Figura 30 "respuesta sin tags".

### La solución: procesamiento intermedio con C#

La solución a estos problemas ha sido combinar AIML con una librería C#, de forma que las respuestas de AIML no lleguen directamente al usuario, sino que sean "filtradas" antes por código C# (Conversational.dll en la Figura 14) para quitar aquellas partes de la especificación AIML que son *datos de control* y así devolver únicamente la *respuesta* al usuario.

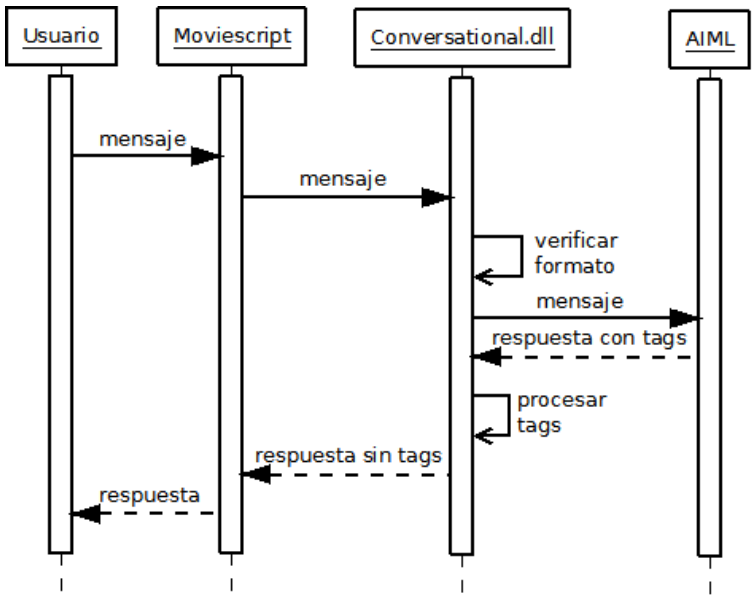


*Figura 14: Desarrollo normal (con formato correcto) de la interacción Usuario-AIML*

En la Figura 14 se clarifica como actúan tanto AIML como la librería Conversational.dll y cooperan por tal de ofrecer una funcionalidad más compleja que la básica de AIML.

Cuando llega un mensaje, primero *Conversational* verifica que el mensaje sea del formato que se espera (en cuanto a tipo de datos y/o valor se refiere). Si el mensaje no es correcto, será *Conversational* mismo quien envíe una respuesta a *Moviescript* que llegará al usuario, informando de que el formato no es correcto para que lo vuelva a introducir. Si es correcto deja que AIML lo interprete, esto es establece cual es la respuesta del bot (template) que se corresponde con el pattern (mensaje del usuario).

Una vez interpretado el mensaje, y antes de devolver la respuesta de AIML al usuario, *Conversational* extrae los tags que contiene la respuesta para procesar sus datos de control. Gracias a estos datos de control *Conversational* sabrá qué formato deberá tener el siguiente mensaje en caso de que el usuario deba introducir mas datos, y en caso de que el usuario haya terminado de introducir los datos necesarios para ejecutar el servicio (p.ej. registrar).

Vemos entonces que *Conversational* actúa como un elemento intermedio entre el bot conversacional y el usuario que chatea, y que realiza funciones que AIML no puede realizar (i.e. verificar formatos y valores, almacenar datos, hacer llamadas) e interviene en lugar de éste cuando es necesario. Por ejemplo, cuando un usuario entra un dato con un formato incorrecto, *Conversational* contesta al usuario en lugar de AIML y de hecho este último no llega a recibir el mensaje del usuario.

Por lo tanto *Conversational* no solo es imprescindible para que el bot

sepa cuando termina una conversación y para almacenar datos (cosa que AIML no puede hacer), sino que a su vez “protege” a AIML asegurándose de que sólo lleguen los mensajes correctos. De no ser así, por ejemplo, si el usuario entrara un dato con el formato incorrecto, AIML lo seguiría considerando y pasaría a preguntar por el siguiente dato o dar por terminado el servicio.

Hemos visto como hemos podido solucionar las tres limitaciones que presentaba AIML: orden, memoria y formato. Las tres han sido solucionadas mediante la inclusión de tags que proceden del AIML y que *Conversational* procesa por tal de cumplir con esas funciones en lugar de AIML.

### Ejemplo y estructura

Los diferentes datos de control que se han incluido en los *template* de AIML han sido:

- ¿Estado final? (y/n): Este indicador servirá para saber si se requiere llamar a una función del sistema (donde se creara un mensaje en AMELI) antes de responder al usuario. Si es “y” el servicio ya se ha completado, en cambio si es “n” aún quedan datos por pedir. La Figura 15 continuación expone este dato de control como una máquina de estados.

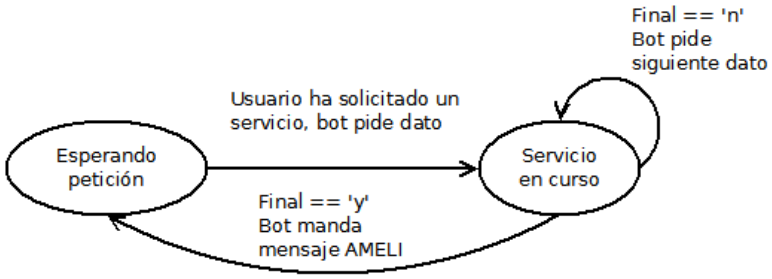


*Figura 15: Máquina de estados que ejemplifica la función de "¿Estado final?"*

- Función (nombre de la función): En este campo se incluye el nombre de la función a llamar, en el caso de los estados finales (campo de estado final con valor 'y').
- Formato del dato (int, double, string, ...): En caso de que no se trate de un estado final (campo de estado final con valor 'n'), el campo para la función tendrá en su lugar el formato que debe tener el dato que se pide.

A continuación se muestra un ejemplo de *category* en AIML utilizadas en la mejora que se ha implementado y que corresponden al servicio de registro.

```
<category>
<pattern>REGISTER</pattern>
<template>n|string In order to register your water right,
please give me the ID of the water right you would like to
sell.</template>
</category>

<category>
<pattern>*</pattern>
<that>N STRING IN ORDER TO REGISTER YOUR WATER RIGHT PLEASE
GIVE ME THE ID OF THE WATER RIGHT YOU WOULD LIKE TO SELL</that>
<template>n|float Last, what is the price you want to sell the
water for?</template>
</category>

<category>
<pattern>*</pattern>
<that>N FLOAT LAST WHAT IS THE PRICE YOU WANT TO SELL THE WATER
FOR</that>
<template>y|register Thank you, you water right has been
registered.</template>
</category>
```

Como podemos ver en el ejemplo, tenemos un *pattern* con '*' y también un *that.* Eso quiere decir que se activará esta *category* cuando se cumpla el *that* e independientemente del mensaje. Por ejemplo, en las *category* anteriores, si el usuario introduce “register” entra ya en una secuencia, y introduzca lo que introduzca a continuación se le responderá con “n|float Last, what is the price you want to sell the water for?”. Se puede también ver que los *that* contienen los campos de información que tenía el mensaje anterior ya que AIML no hace distinción entre esta información y el resto del mensaje. Cabe destacar que los caracteres “|” no se incluyen en los *that* de AIML y en su lugar se pone un espacio, por motivos del funcionamiento interno de AIML. Así mismo, en los tags *that* se escribe en mayúscula por convención, ya que así es mas fácil distinguir el texto del tag y AIML es independiente de las mayúsculas/minúsculas.

La respuesta que se le da al usuario en este *template* contiene un primer campo con toda la información que *Conversational* deberá procesar y eliminar. En el campo (“y|register”) tenemos los tres datos que AIML proporciona a *Conversational*, separados por '|'. En concreto aquí se indica que *Conversational* deberá pasar al estado inicial de la conversación (UC4-Chatear con agente), que se ha finalizado la recolecta necesaria de datos para ofrecer un servicio, y que el servicio a proporcionar al usuario se realiza mediante una función llamada “register” en el *Moviescript* del que proviene el mensaje.

## 10.2 Resultados

A continuación se presentan capturas de pantalla pertenecientes a *v-mWater* accedido desde el visor Imprudence, y en el que un avatar conversa con un bot conversacional por tal de registrar su derecho del agua. Las imágenes reflejan la secuencia de la conversación que mantienen el bot y el usuario.

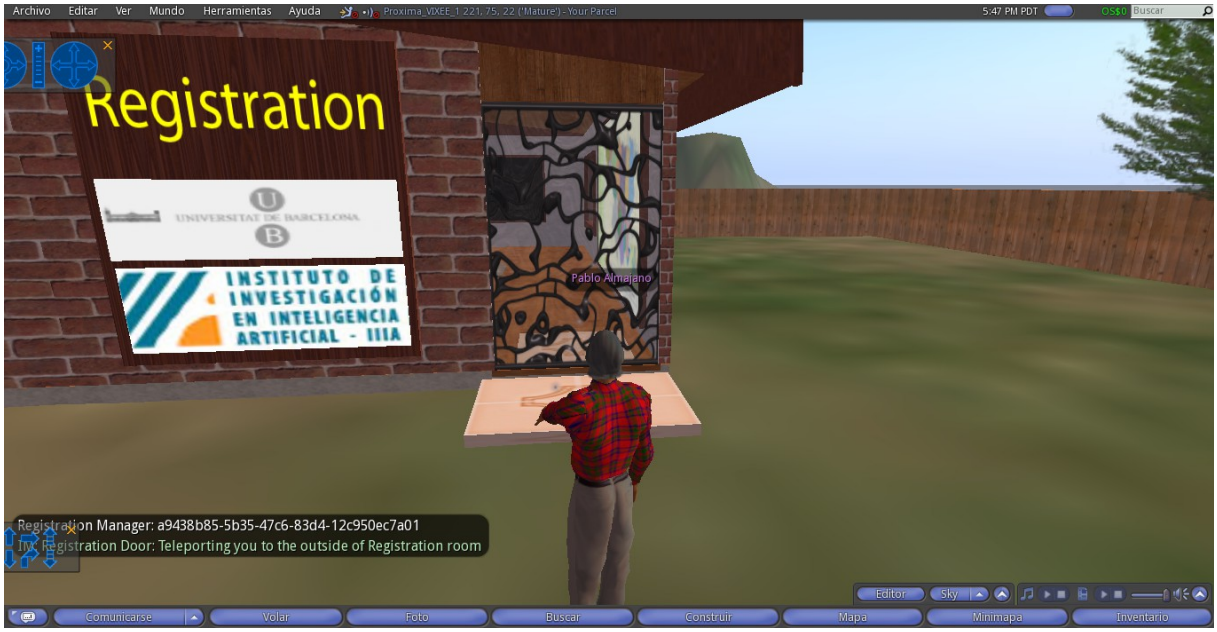


*Figura 16: Entrando a la sala de registro*

En la Figura 16 vemos que nuestro avatar, representando a Pablo Almajano, se dirige a la sala de registro para registrar su derecho del agua, clica en la puerta para entrar en la actividad de registro.


*Figura 17: Saludando al Registration Manager*

A continuación, en la Figura 17 Pablo saluda (“Hello”) al Registration Manager.

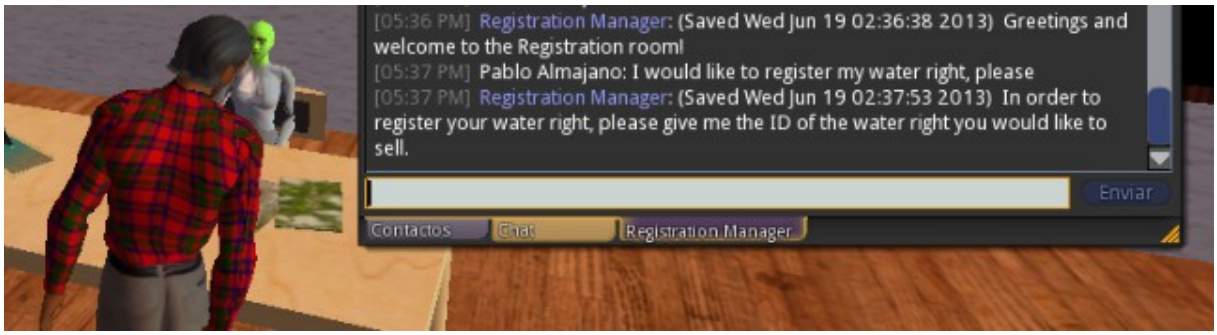


*Figura 18: Solicitando un servicio*

En la Figura 18 Pablo expresa al bot conversacional su intención de registrar.

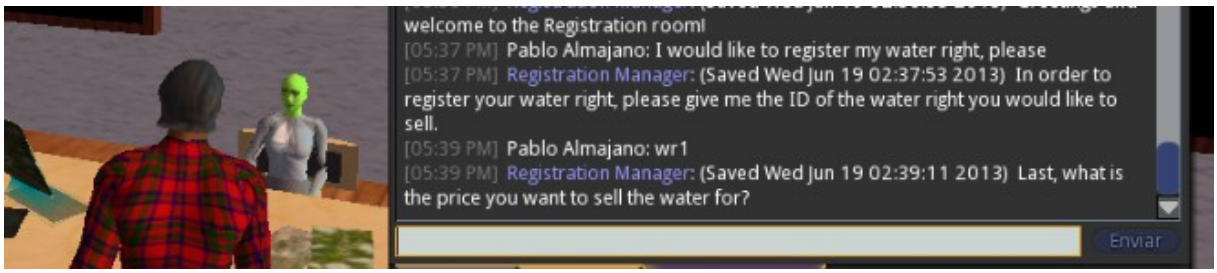


*Figura 19: Pidiendo datos*

La Figura 19 muestra que el Registration Manager le pide a Pablo los datos necesarios para que registre, empezando por el *water right*. Pablo se lo proporciona, (*wr1*) y el Registration Manager le pide el precio.

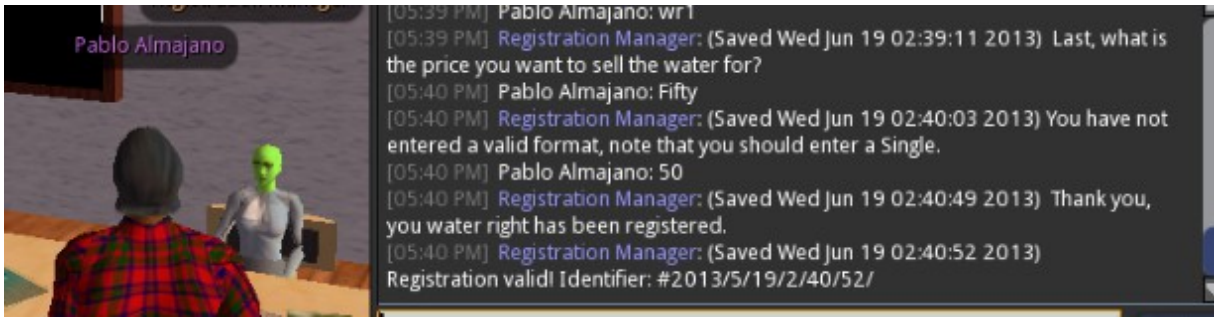


*Figura 20: Verificando el formato*

En la Figura 20 Pablo introduce como precio “cincuenta”. Como se debe proporcionar un número entero, el Registration Manager se lo dice a Pablo. Pablo introduce el precio correctamente y el Registration Manager, que ya tiene todos los datos que necesita, registra el derecho del agua y se lo confirma a Pablo (*Thank you, your water right has been registered)*.

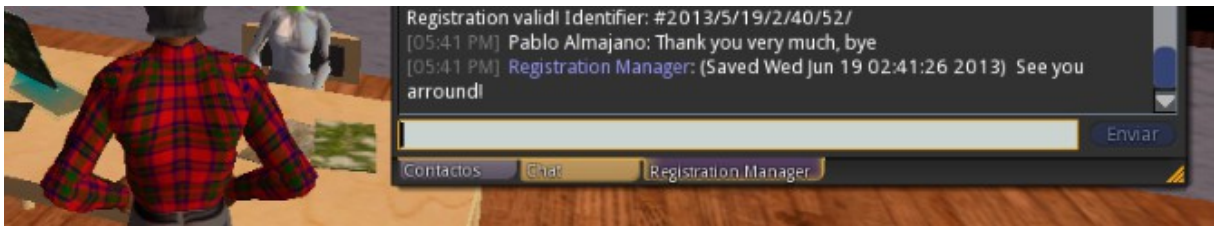


*Figura 21: Despidiéndose*

La Figura 21 muestra que Pablo ya ha realizado la tarea que quería, por lo tanto se despide del Registration Manager que le contesta con otra despedida.

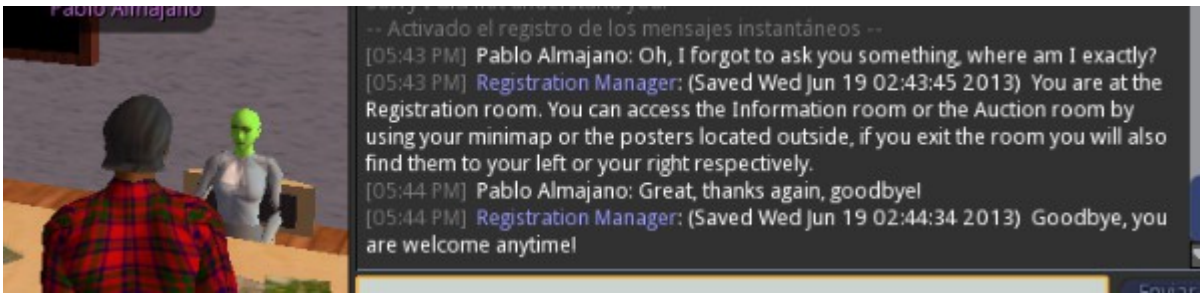

*Figura 22: Situando al usuario*

En la Figura 22 Pablo anda un poco perdido en *v-mWater,* así que decide aprovechar y preguntar *(Oh, I forgot to ask you something, where am I exactly?) al* Registration Manager, por si supiera situarle. El Registration Manager le sitúa explicándole las demás salas y diciéndole cómo acceder a ellas. Pablo se despide de nuevo, a lo que el bot conversacional contesta con otra despedida diferente a la anterior.

## 10.3 Pseudocódigo para futura mejora: generación automática de ficheros AIML

En este proyecto, los AIML que se han usado han sido generados de forma manual conociendo las conversaciones que podía mantener el bot con los usuarios. Ya que estos AIML se basan en buscar palabras clave en lugar de frases que coincidan enteras (véase sección 9.1) y estas palabras clave suelen ser los nombres de los servicios que realizan los bots (p. ej. cuando el bot a petición del usuario inicia cualquier trámite) se entrevé la posibilidad de automatizar el proceso de generación de los ficheros AIML. Con ésto se conseguiría una mayor eficiencia y comodidad ya que los ficheros se generarían antes y con menos esfuerzo.

En nuestro caso los bots a los que nos referimos son *staff agents,* es decir agentes software que pertenecen a una Institución Electrónica. En *v-mWater,* esos agentes son *MarketFacilitator* y *BasinAuthority.*

Entonces, el pseudocódigo que se presenta mas adelante en esta sección está ideado para obtener los ficheros AIML necesarios para hacer funcionar el procesamiento de lenguaje de manera automática, requiriendo por parte del usuario proporcionando la especificación de la Institución Electrónica.

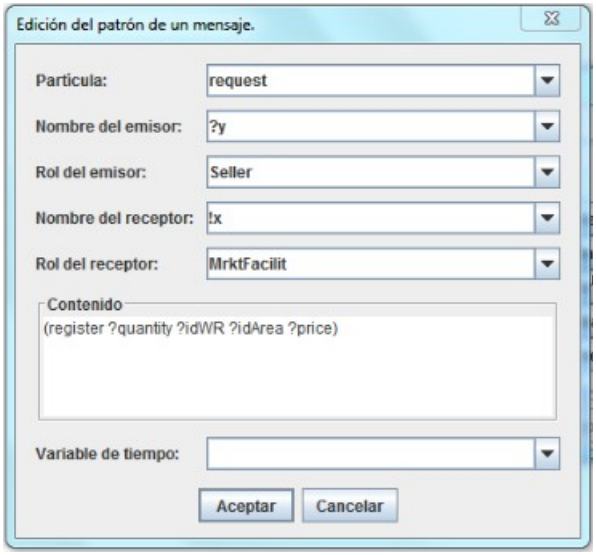


*Figura 23: Composición de un request en la especificación de una Institución Electrónica.*

En la Figura 23 podemos recordar la composición del mensaje *request* de la especificación de una Institución Electrónica. El campo “Rol del receptor” indica a que bot le vamos a construir la base de conocimiento AIML.

También utilizaremos la información disponible en “Contenido”, aunque a partir de el *campo de descripción* de que disponen los arcos como el que estamos viendo, y donde se relacionaremos cada servicio y los datos que se necesitan con los nombres que se les quiera dar a la hora de identificarlos en AIML. Aparte, será necesario incluir el resto de campos de información que necesita el AIML en un formato determinado para permitir la generación automática. A continuación se incluye un ejemplo con estos campos.

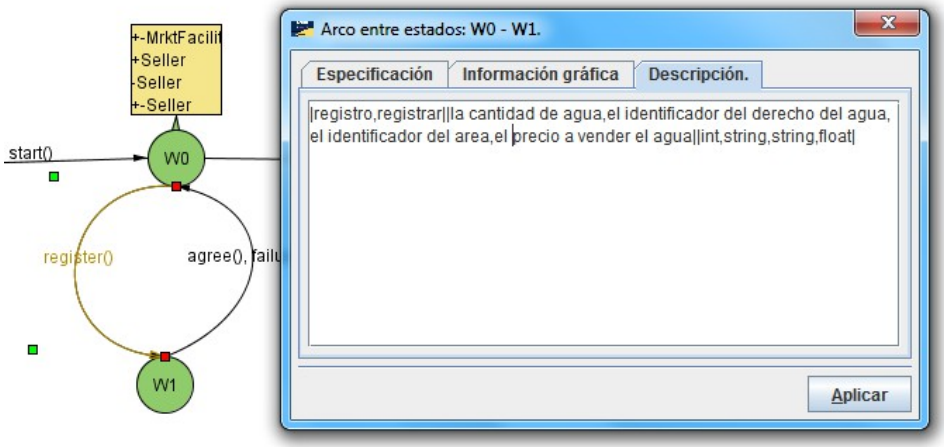


*Figura 24: Campo de descripción preparado para generación automática*

En la Figura 24 podemos ver el formato concreto que el algoritmo utilizará para generar los ficheros AIML de manera automática. Primero, vemos que cada campo se encuentra entre caracteres '|'. De esta manera podemos separar cada uno de los diferentes campos que queremos que se tengan en cuenta, y también permitimos que se pueda seguir usando el campo para otros propósitos, ya que una vez no se encuentren mas campos (buscaremos '|' para saberlo) se dará por concluida la generación de ese servicio sin importar qué mas contenga el campo de descripción. En cada campo entre '|' hay varias palabras, separadas todas ellas por coma:

- Primer campo: indica el nombre que se le quiere dar al servicio, y se permite poner tantos nombres diferentes como se quieran separados por coma. Así, en este caso se detectará que el usuario quiere registrar tanto si pone “Quisiera registrar” como si pone “¿Es aquí el mostrador de registro?”.
- Segundo campo: contiene los datos que se necesitan para realizar el servicio. Por cada palabra que haya, se pedirá un dato más y se le dará el nombre que se indique. Por ejemplo, para pedir el primer dato el bot nos dirá “Por favor, introduce la cantidad de agua” y para pedirnos el segundo nos dirá “Por favor, introduce el identificador del derecho del agua”.
- Tercer campo: nos indica para cada uno de los datos que formato le corresponde, por orden. Entonces, vemos que la cantidad de agua se deberá introducir como un número entero, mientras que el identificador del derecho será en forma de texto, permitiendo tanto números como letras.

```
<category>
<pattern>REGISTRAR</pattern>
<template>n|int Por favor, introduce la cantidad de
agua</template>
</category>

<category>
<pattern>* REGISTRAR *</pattern>
<template><srai> REGISTRAR </srai></template>
</category>

<category>
<pattern>*</pattern>
<that>N INT POR FAVOR, INTRODUCE LA CANTIDAD DE AGUA</that>
<template>n|string Por favor, introduce el identificador del
derecho del agua</template>
</category>
```

En el pseudocódigo elaboraremos sólo el template de “nombreDato”, y para el resto usaremos el tag <srai>. *Srai* permite hacer que un *template* se redirija a otro *template*, que tendrá el *pattern* que se especifique. Como podemos ver en el ejemplo, cuando se active el *pattern* “* REGISTRAR*”, la salida será aquella del *pattern* “REGISTRAR”.

Para hacer más precisa la detección, consideraremos los *pattern* “servicio”, “* servicio”, “ servicio *” y “* servicio *” donde “servicio“ corresponde a el nombre que se le haya dado al primer el campo de descripción. Así, buscaremos el nombre del servicio en concreto por sí solo, con cualquier texto detrás, con cualquier texto delante y por último con cualquier texto detrás y delante, respectivamente. Así pues, en el ejemplo anterior “registrar” coincidiría con el primer *pattern* y “asdfg registrar hjklñ” con el segundo.

Como podemos observar en el ejemplo de AIML anterior, en el caso de los datos el mensaje que nos dice el bot para pedir los datos es siempre el mismo añadiendo al final el nombre que se le haya dado al dato (“Por favor, introduce ...”). Esta frase tan genérica nos permite poder utilizarla para cualquier tipo de dato sin que pueda quedar mal al juntar la frase con el nombre del dato. Esta frase aparecerá en cada *template* de las *category* en donde se pida un dato. Recordemos que una *category* es un tag en AIML constituido como mínimo por un *pattern* o patrón que debe cumplirse para activar la *category* y el *template* o respuesta a dar cuando se cumpla el *pattern*. A continuación vemos un ejemplo del AIML que se generaría automáticamente.

También vemos en el código AIML anterior que en el caso en que se estén pidiendo datos (UC5- Proporcionar datos al agente) el *pattern* puede ser cualquiera y lo que realmente se busca es qué dijo antes del bot usando el tag *that*. Así, conseguiremos encadenar los mensajes de manera que se pidan los datos de manera ordenada y una vez se introduzca el último podamos dar el servicio por terminado. Las primeras palabras del

*that* corresponden a datos que luego el módulo de conversación lo aprovechará (borrándolo luego), y que por tanto el usuario solo recibirá la parte de la respuesta en lenguaje natural. “INT” por lo tanto corresponde al formato del dato que se pide.

“N” corresponde a si se trata del último dato o no. Puede ser “y” (yes) o “n” (no). Si es “y”, sabremos que hemos acabado el servicio y por lo tanto pasaremos a un estado en el cual no se ha solicitado aún ningún servicio (UC4-Chatear con agente) y tomará importancia el campo *pattern* ya que en ese momento buscaremos palabras clave en los mensajes del usuario para iniciar un servicio. Al contrario, cuando es “n” y pedimos datos nos fijamos en el *that* porque nos interesa saber que dato toca pedir mirando cual se ha dado antes, independientemente de que valor se le haya dado. Por ejemplo, después de recibir la cantidad de agua a vender pediremos el identificador del derecho del agua, independientemente de si la cantidad hubiera sido 50 o 500.

El pseudocódigo que se ha diseñado es el siguiente:

```
 entrada: ruta al documento .xml con la especificación de la Institución
Electrónica
 salida: conjunto de ficheros AIML con los nombres de cada bot.


 # Se recorre la especificación
 Para cada actividad en entrada:
     Para cada request en actividad:
           # Se guardan los diferentes campos, sabiendo que los
           # campos están entre '|' y dentro de ellos cada palabra
           # separada por ','
           descripcion = campo de descripción del request
           metodos = strings en primer campo de descripción
           (separador = ',')
           parametros = strings en segundo campo de descripción
           (separador = ',')
           formatos = strings en tercer campo de descripción
           (separador = ',')
           emisor = emisor del request
           # Creamos el AIML del bot que ofrece el servicio si no existe
           si AIML con nombre == emisor no existe:
                 crear carpeta con nombre = emisor
                 crear nuevo AIML en carpeta con nombre = emisor
           # Almacenamos en una variable el fichero AIML que vamos a
           # modificar (p.ej. RegistrationManager.aiml)
```

```
            aiml = AIML con nombre emisor en carpeta con nombre emisor
            # Se crean las category donde el usuario pide el servicio
            # Cogemos la primera palabra de cada uno de los campos
            primerMetodo = metodos.first
            parametro = parametros.first
            formato = formatos.first
            añadir category con pattern = primerMetodo y template = "n|"
            + formato + " Por favor, introduce " + parametro a aiml
            # Añadimos las combinaciones (asd register, register fgh, etc)
            añadirCombinacionesPattern(primerMetodo, primerMetodo, aiml)
            # Hacemos lo mismo para todos los demás nombres
            Para los demás metodo en metodos:
                  añadirCombinacionesPattern(metodo, primerMetodo, aiml)
            # Creamos las category
            anteriorParametro = parametro
            anteriorFormato = formato
            cont = 0
            Para los demás parametro en parametros:
                  cont ++
                  # Si estamos con el último parámetro, hay que terminar
                  # el servicio
                  si cont == parametros.length:
                        añadir category con pattern = "*", template = "y|"
                        + primerMetodo + "  La operación se ha realizado
                        con éxito" y that =                "N|" +
                        anteriorFormato.enMayúsculas() + " POR FAVOR,
                        INTRODUCE " + parametro.enMayúsculas() a aiml
                  formato = formatos.next()
                  añadir category con pattern = "*", template = "n|" +
                  formato + " Por favor, introduce " + parametro y that =
                  "N|" + anteriorFormato.enMayúsculas() + " POR FAVOR,
                  INTRODUCE " + parametro.enMayúsculas() a aiml

# Crea todas las combinaciones de un método con srai a otro método
rutina añadirCombinacionesPattern(metodo, metodoSrai, aiml):
    # Miramos si coinciden los dos métodos, entonces se trata del
    # primer método de todos y su primera combinación ya esta
    # creada (la que contiene el template, y no un srai)
    Si métodoSrai != método:
          añadir category con pattern = metodo y srai = metodoSrai a
          aiml
    añadir category con pattern = "* " + metodo y srai = metodoSrai a
    aiml
```

```
añadir category con pattern = metodo + " *" y srai = metodoSrai a
aiml

añadir category con pattern = "* " + metodo + " *" y srai =
metodoSrai a aiml
```

Una vez implementado este algoritmo se debería poder prescindir de generar manualmente los AIML. De todas formas, el programador siempre será libre de retocar los AIML manualmente para dejarlos a su gusto. Gracias al AIML pues el programador ahorrará tiempo, tanto si quiere unos mensajes sencillos y concisos como si busca un bot conversacional con mensajes más complejos o más variados, ya que entonces ya los tendrá generados y sólo necesitará cambiar los campos que quiera en los ficheros AIML ya generados.

# 11 Conclusiones y trabajo futuro

En este proyecto hemos realizado un estudio de usabilidad sobre un prototipo de un entorno virtual que implementa un mercado del agua, posteriormente se ha realizado una mejora en dicho prototipo para abordar el mayor problema de usabilidad del mismo, la comunicación agente-humano por comandos textuales con una sintaxis estricta.

Gracias a este proyecto se ha podido mejorar de manera considerable el prototipo del que se partía en cuanto a usabilidad se refiere. Así mismo, se ha creado una herramienta aplicable a cualquier otra institución Virtual y que da solución a un problema de usabilidad importante. También se ha podido obtener una medida de la disposición que tienen los usuarios de utilizar mundos virtuales para tareas relacionadas con el e-Government y la utilidad que ven en estos entornos.

De manera más general, se ha planteado un nuevo modo de explotar el potencial del lenguaje AIML, ampliando su utilidad mediante código intermedio que le proporciona más flexibilidad y capacidades. Esto es, la capacidad del bot (ahora conversacional) para pedir datos al usuario y controlar que éstos sean correctos.

Como trabajo futuro se propone implementar el algoritmo de generación automática de los AIML a partir de la especificación de la Institución Electrónica. Esto facilitará el proceso de creación de bots conversacionales en una Institución Virtual. Otra mejora futura estrechamente relacionada con la comunicación bot-humano puede ser la integración de un módulo sintetizador de voz y un módulo de reconocimiento de voz para que el usuario no solo pueda usar el chat textual sino también el chat de audio.

# 12 Bibliografía


A. Bogdanovych. Virtual Institutions. PhD thesis, University of Technology, Sydney, Australia, 2007.

AIML (consultado el 06/2013) *http://www.alicebot.org/AIML.html* A.L.I.C.E.

AIMLbot (consultado el 06/2013) *http://aimlbot.sourceforge.net/* Program#

Almajano P., Mayas E., Rodriguez I., Lopez-Sanchez M. and Puig A. (2013). Structuring Interactions in a Hybrid Virtual Environment - Infrastructure & Usability. In Proceedings of the International Conference on Computer Graphics Theory and Applications and International Conference on Information Visualization Theory and Applications, pages 288-297. DOI: 10.5220/0004215802880297

Almajano, P.; Trescak, T.; Esteva, M.; Rodríguez, I.; López-Sanchez, M. (2012). V-mWater: an e-government application for water rights agreements

Behrens, T., Hindriks, K., and Dix, J. (2011). Towards an environment interface standard for agent platforms. Annals of Mathematics and Artificial Intelligence, 61:261–295.

Blair, J. and Lin, F. (2011). An Approach for Integrating 3D Virtual Worlds with Multiagent Systems. In Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on, pages 580 –585.

Book, B. (2004). Moving beyond the game: social virtual worlds. State of Play, 2:6–8.

Bowman, D., Gabbard, J., and Hix, D. (2002a). A survey of usability evaluation in virtual environments: classification and comparison of methods. Presence: Tele- operators & Virtual Environments, 11(4):404–424.

Bowman, D., Johnson, D., and Hodges, L. (1999). Testbed evaluation of virtual environment interaction techniques. In Proceedings of the ACM symposium on Virtual reality software and technology, pages 26–33. ACM.

Bowman, D. A., Gabbard, J. L., and Hix, D. (2002b). A survey of usability evaluation in virtual environments: classification and comparison of methods. Presence: Teleoper. Virtual Environ., 11:404–424.

Cranefield, S. and Li, G. (2009). Monitoring social expectations in Second Life. In AAMAS'09, pages 1303–1304, Richland, SC.

Dignum, V., Meyer, J., Weigand, H., and Dignum, F. (2002). An organization-oriented model for agent societies. In AAMAS'02.

Esteva, M., de la Cruz, D., and Sierra, C. (2002). IS- LANDER: an electronic institutions editor. In AAMAS '02, pages 1045–1052. ACM.

Esteva, M., Rosell, B., Rodríguez-Aguilar, J. A., and Arcos, J. L. (2004). AMELI: An agent-based middleware for electronic institutions. In AAMAS'04, pages 236–243.

Ferraris, C. and Martel, C. (2000). Regulation in groupware: the example of a collaborative drawing tool for young children. In Groupware, 2000. CRIWG 2000. Proceedings. Sixth International Workshop on, pages 119–127. IEEE.

Gajananan, K., Nakasone, A., Hildebrandt, A., and Prendinger, H. (2010). A Novel Three-Dimensional Collaborative Online Platform for Bio-molecular Modeling. In Taylor, R., Boulanger, P., Krger, A., and Olivier, P., editors, Smart Graphics, LNCS, pages 44– 55. Springer Berlin / Heidelberg.

Giret, A., Garrido, A., Gimeno, J. A., Botti, V., and Noriega, P. (2011). A MAS decision support tool for water-right markets. In AAMAS '11, pages 1305–1306.

Guerra, A., Paredes, H., Fonseca, B., and Martins, F. (2008). Towards a Virtual

Environment for Regulated Interaction Using the Social Theatres Model. Groupware: Design, Implementation, and Use, pages 164–170.

Imprudence (consultado el 06/2013). Imprudence team. *http://wiki.kokuaviewer.org/*.

Imprudence download (consultado el 06/2013)

*http://wiki.kokuaviewer.org/wiki/Imprudence:Downloads*

Mezura-Godoy, C. and Talbot, S. (2001). Towards social regulation in computer-supported collaborative work. In Groupware, 2001. Proceedings. Seventh International Workshop on, pages 84–89. IEEE.

OpenMetaverse (consultado el 07/2012). OpenMetaverse Foundation. http://openmetaverse.org/.

OpenSimulator (consultado el 06/2013). Overte Foundation. *http://opensimulator.org/*.

Pandorabots (consultado el 06/2013) *http://www.pandorabots.com/botmaster/en/home*

Pandorabots in SecondLife (consultado el 06/2013) *http://pandorabot.blogspot.com.es/2010/11/what-is-pandorabot.html*

Paredes, H. and Martins, F. (2007). Social theatres: a web-based regulated social interaction environment. In Proceedings of the 13th international conference on Groupware: design implementation, and use, pages 87–94. Springer-Verlag.

Paredes, H. and Martins, F. (2010). Social interaction regulation in virtual web environments using the Social Theatres model. In Computer Supported Cooperative Work in Design (CSCWD), 2010 14th International Conference on, pages 772–777. IEEE.

Pronost, N., Sandholm, A., and Thalmann, D. (2011). A visualization framework for the analysis of neuromuscular simulations. The Visual Computer, 27(2):109–119.

Rubin, J. and Chisnell, D. (2008). Handbook of usability testing : how to plan, design, and conduct effective tests. Wiley Publ., Indianapolis, Ind.

Trescak, T.; Rodriguez, I.; Lopez Sanchez, M.; Almajano, P. Execution infrastructure for normative virtual environments Engineering Applications of Artificial Intelligence Volumen 26 Número: 1 Páginas, inicial: 51 final: 62 Año: 2013 ISSN: 0952-1976

Trescak, T., Esteva, M., and Rodriguez, I. (2011). Vixee an innovative communication infrastructure for virtual institutions. In Proceedings of The 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'11), volume 3 of AAMAS '11, pages 1131–1132, Rich- land, SC. IFAAMAS

Tromp, J., Steed, A., and Wilson, J. (2003). Systematic us- ability evaluation and design issues for collaborative virtual environments. Presence: Teleoperators & Virtual Environments, 12(3):241–267.

Tsiatsos, T., Andreas, K., and Pomportsis, A. (2010). Evaluation framework for collaborative educational virtual environments. Educational Technology & Society, 13(2):65–77.

van Oijen, J., Vanh´ee, L., and Dignum, F. (2011). CIGA: A Middleware for Intelligent Agents in Virtual Environments. In Proceedings of the 3rd International Workshop on the uses of Agents for Education, Games and Simulations.

# Apéndice A: Manual técnico (código fuente)

En este manual se especifican las instrucciones a seguir para incorporar el sistema de procesamiento en lenguaje natural en tu instalación del código fuente VIXEE. A lo largo de este manual haremos referencia al fichero comprimido (adjuntado junto con esta memoria) pfc-EnricMayas.zip que contiene los ficheros AIML, las librerías y el código necesarios.

Como prerequisitos, debes tener instalado MonoDevelop para modificar el proyecto, MySQL community edition y MySQL WorkBench Administrator para arrancar la base de datos con los assets de *v-mWater,* OpenSim y por último un visor para poder acceder a *v-mWater* con un avatar (p.ej. Imprudence).

Hay que seguir los siguientes pasos:

1. Contactar con el autor de VIXEE (Tomas Trescak, tomi.trescak@gmail.com) para **bajar el código fuente.** A continuación en la Figura 25 podemos ver el árbol de directorios resultante.

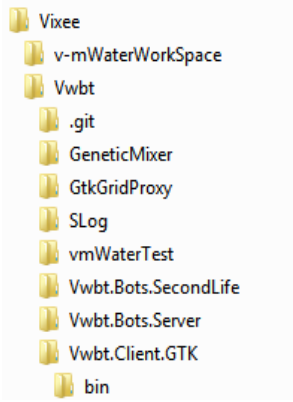


*Figura 25: Árbol de ficheros de VIXEE*

2. **Copiar la carpeta "pfc-EnricMayas/codigo/aiml"** en el directorio *Debug* de VIXEE, esto es "Vixee\Vwbt\Vwbt.Client.GTK\bin\Debug". Seguidamente prepararemos el código para que use estos ficheros AIML. **Copia las librerías Conversational.dll y AIMLbot.dll que están en pfc-EnricMayas/codigo/libreriasAiml**

en "Vixee\Vwbt\Vwbt.Client.GTK\bin\Debug". Asegúrate de que el proyecto referencie estas librerías que acabas de copiar. Tu árbol de directorios quedará tal y como se muestra en la Figura 26:

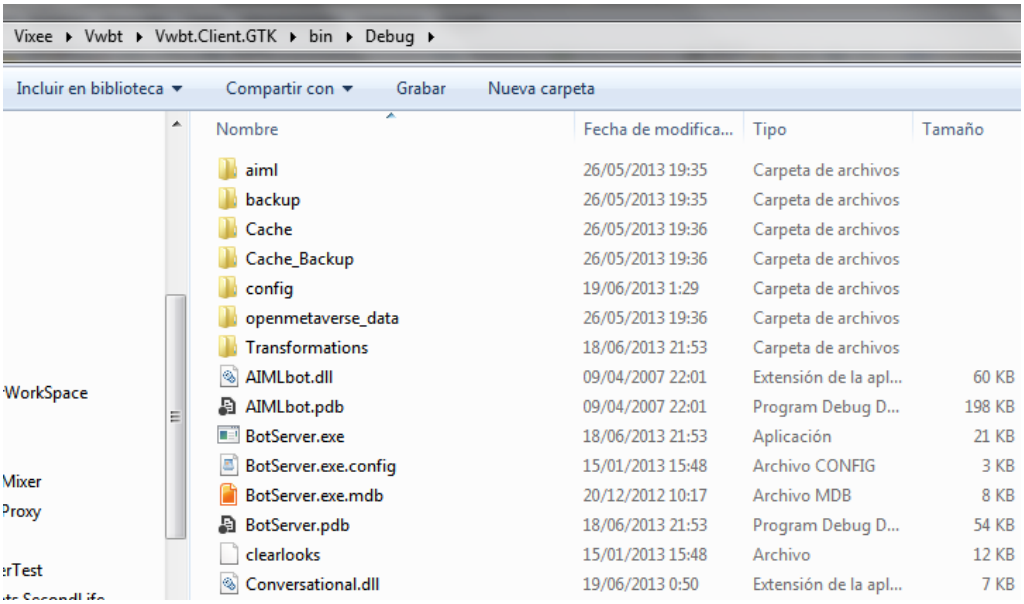


*Figura 26: Carpeta Debug de VIXEE*

3. **Copiar la carpeta "pfc-EnricMayas/codigo/v-mWaterWorkSpace"** en el directorio "Vixee".
4. Dentro de "v-mWaterWorkSpace" se pueden encontrar los *Moviescript* de recepción de mensaje de los diferentes agentes. Estos *Moviescript* (ver sección 5.3) tienen como nombre "ToNombreAgenteInstantMessage.cs". En la Figura 27 podemos ver la estructura del proyecto una vez abierto, en concreto con Xamarin Studio. Vemos que la carpeta "VwActions" está abierta, esta carpeta contiene todos los *Moviescript* de *v-mWater.*

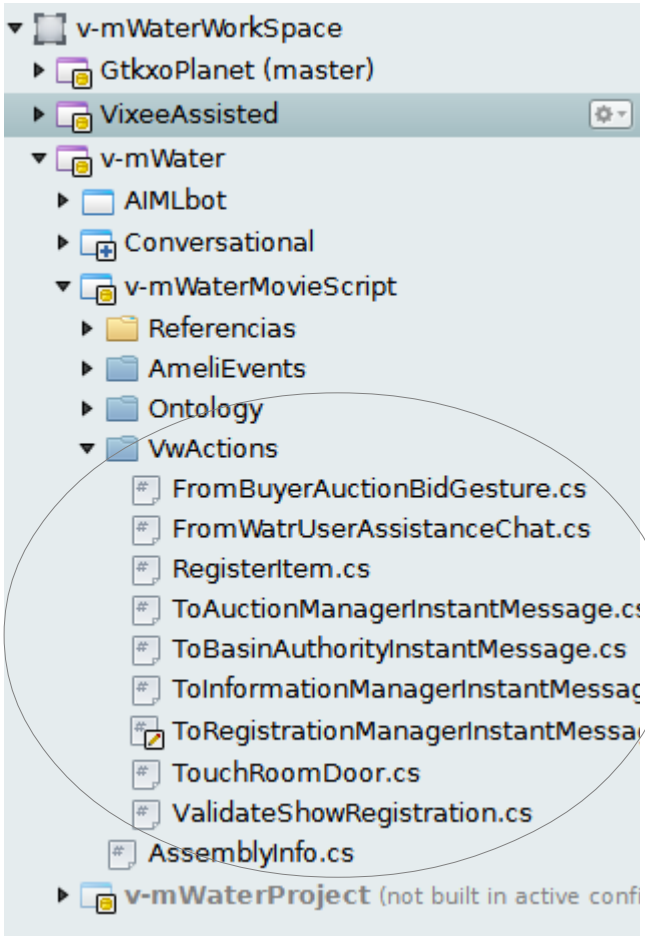


*Figura 27: Proyecto VIXEE/v-mWater en Xamarin Studio*

Para cada uno de los *Moviescripts* que modifiquemos, empezaremos por instanciar un "Conversation Manager" como static. Esta clase se encargará de gestionar todas las conversaciones que el bot mantenga. Introduce esta línea en las declaraciones iniciales del *Moviescript*:

```
1   private static Conversational.ConversationManager convManager = new
Conversational.ConversationManager("nombre del bot");
```

Sustituye el campo "*nombre del bot*" por el nombre del bot en concreto para el cual se estén procesando las entradas. Aunque no es necesario hacerlo así, de esta forma te será mas fácil asociar cada *ConversationManager* con el bot al que pertenece.

A continuación, nos dirigiremos al método llamado *Play* en el *Moviescript.* Veremos un *try* en este método, esa es la acción que se ejecuta cada vez que le llega un mensaje al bot. Nosotros introduciremos nuestro código de procesamiento de lenguaje natural en el *catch.* Así, el bot intentará primero encontrar un comando (tal y como y se hacia antes), y en caso contrario acudirá a nuestro procesamiento de lenguaje natural. El código que debe haber en el *catch* es:

```
1  #region IMovieScriptActionVw implementation
2  public ActionResult Play (ref MessageBase message, ScriptLine line,
EiAgentState state)
3  {
4        var instantMessage = message.Content ["Message"].ToString ();
5        var commands = instantMessage.Split(' ');
6        string functionName = "";
7        functionName = state.SceneState.Scene.Name + "_" + commands[0];
8        try { # El bot comprueba si lo que ha recibido es un comando
9              MethodInfo m = this.GetType ().GetMethod (functionName,
BindingFlags.NonPublic | BindingFlags.Instance);
10             return (ActionResult)m.Invoke (this, new object[]{state,
commands});
11       } catch (Exception e) { # El bot usa AIML
12             string response =
convManager.GenerateResponse(state.Agent._foreignID, instantMessage);
13             SlController ctrBot =
ServerManager.DefaultServer.GetAvatarControllerByAvatarName("Registration
Manager") as SlController;
14             ctrBot.SendIM(state.Agent._foreignID, response);
15             if (convManager.IsFinal(state.Agent._foreignID)){
16                   string method = state.SceneState.Scene.Name + "_" +
                     convManager.GetMethod(state.Agent._foreignID);
17                   string[] parametres =
convManager.GetParametres(state.Agent._foreignID);
18                   MethodInfo m = this.GetType ().GetMethod (method ,
BindingFlags.NonPublic | BindingFlags.Instance);
19                   return (ActionResult)m.Invoke(this, new object[]
{state, parametres});
20             }
21             return ActionResult.OK;
22       }
23  }
```

Deberás sustituir "Registration Manager" de la línea 2 por el nombre del bot en cuestión. Para ello puedes revisar el nombre del *Moviescript,* “ToNombreAgenteInstantMessage.cs” donde “NombreAgente” es el nombre del bot.

5. Una vez modificado el código fuente, recompilamos Vixee. Para ello, debemos arrancar la base de datos de los *assets* (contenido 3D) de

*v-mWater,* el servidor de mundos virtuales (*OpenSim.exe*) y BotServer.exe (en “Vixee\Vwbt\Vwbt.Bots.Server\bin\Debug”) que gestiona los bots que se conectan al mundo virtual desde Vixee.

En este punto Vixee ya estaría preparado para que un usuario se conectara mediante un visor (p.ej. Imprudence).

# Apéndice B: Manual de usuario

En este manual se especifica cómo el usuario utilizará el sistema de lenguaje natural en una Institución Virtual.

Lo primero que va a ser necesario será disponer de un **viewer. E**sta herramienta nos permitirá conectarnos al mundo virtual con nuestro avatar y interactuar a través de él y del mundo con la Institución Electrónica. Los pasos a seguir para instalar un viewer son:

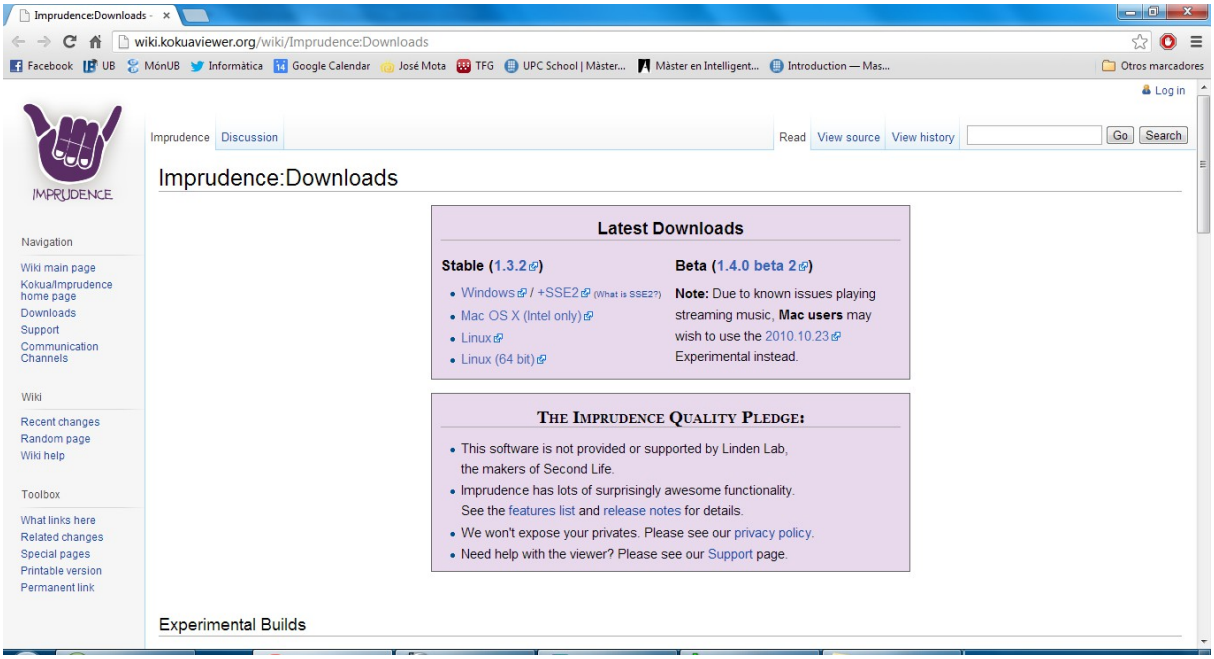

*Figura 28: Página web de descarga de Imprudence*

1. Descargar el viewer. Existen varios pero aquí tomaremos como ejemplo el viewer **Imprudence.** Este viewer esta disponible para diferentes sistemas operativos (ver Figura 28). Puedes ver cuáles son y descargarlo a través de este link: *http://wiki.kokuaviewer.org/wiki/Imprudence:Downloads*


*Figura 29: Primera pantalla del instalador de Imprudence*

2. Instala el viewer. En el caso de Windows ésto se realiza abriendo el ejecutable que nos hemos descargado del link en el paso 1. El instalador te guiará a través del proceso de instalación (ver Figura 29).

Una vez tengamos instalado un viewer, será necesario crear una **cuenta** asociada al mundo virtual al que queramos acceder. Para ello, deberás contactar con el administrador del mundo virtual del que se trate. Necesitarás un nombre de usuario que se asociará a tu avatar, así como una contraseña para poder autenticar-te en el mundo virtual.

Ahora debemos **conectarnos** al mundo virtual con nuestro avatar. Para ello:

1. Obtén los datos del mundo virtual. Contacta con el administrador por tal de obtener la información necesaria. Deberás obtener, como mínimo, una dirección (URI) a través de la cual conectarte al mundo virtual.

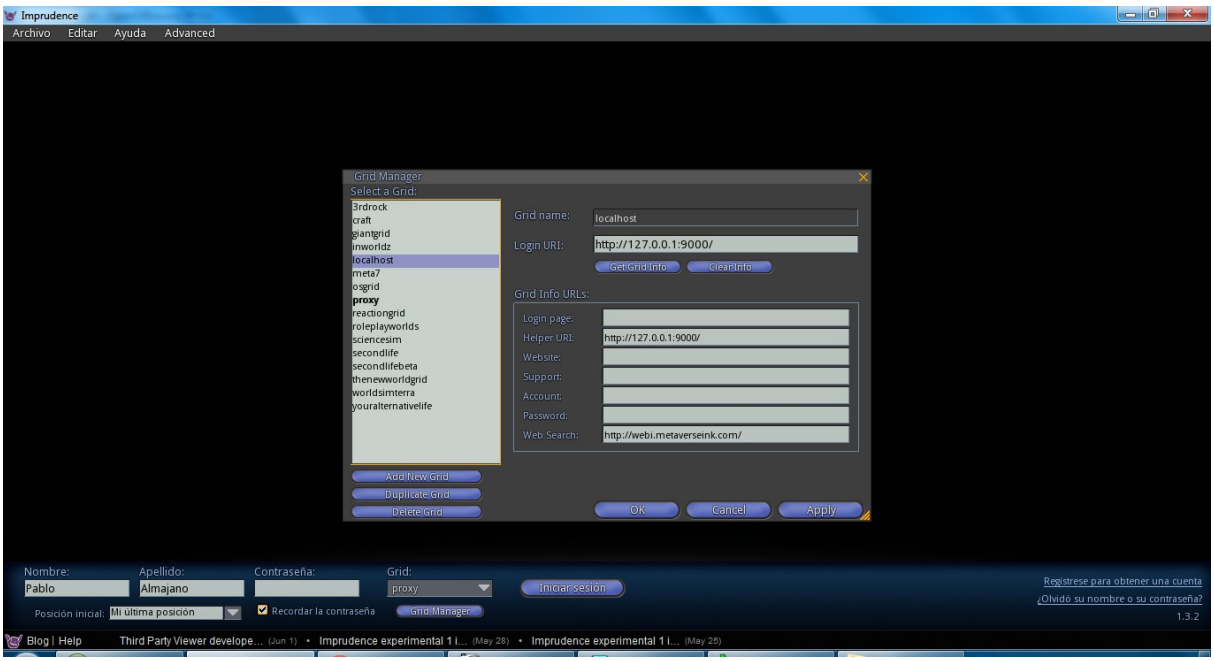


*Figura 30: Grid Manager de Imprudence*

2. Añade el mundo virtual a tu colección de mundos. En **Imprudence,** ésto se hace mediante el grid manager. Haz clic en el botón con el nombre "Grid Manager" (ver Figura 30). Ahora, haz clic en el botón "Add New Grid", por tal de añadir nuestro mundo a la colección. Rellena el campo "Grid Name" con el nombre que desees darle (ésto sólo te sirve a ti como modo de identificarlo), e introduce la URI que te proporcionó el administrador en "Login URI". Verás que hay más campos por rellenar, pero sólo te serán necesarios los mencionados.

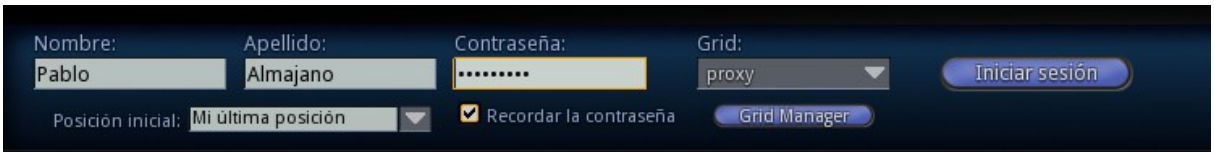


*Figura 31: Campos a rellenar para hacer Login*

3. Ingresa en el mundo virtual. Para ello selecciona el mundo que has creado del desplegable “Grid:”, y rellena los campos “Nombre:”, “Apellido:” y “Contraseña:” según los datos que te haya proporcionado el administrador (ver Figura 31).


*Figura 32: El avatar entra en la sala de registro*

Una vez dentro del mundo, podremos **interactuar** con los bots. Para ello dirígete a cualquier de los bots de la Institución Electrónica (recuerda que deberás entrar a la misma sala en la que estén ellos). Para entrar, clica con el botón izquierdo del ratón a la puerta de la sala, sin importar la distancia (ver Figura 32) y en breve serás teleportado dentro. Una vez dentro localiza a un bot conversacional y habla con él. Para ello, haz clic derecho sobre el bot, y selecciona la opción “Enviar un MI…” (ver Figura 33). Puedes probar saludándoles, pidiéndoles ayuda, o si sabes qué servicios te ofrece el bot pregúntale por éstos directamente. Una vez te hayan atendido, despídete de ellos y habla con los demás, seguro estarán encantados de ayudarte.

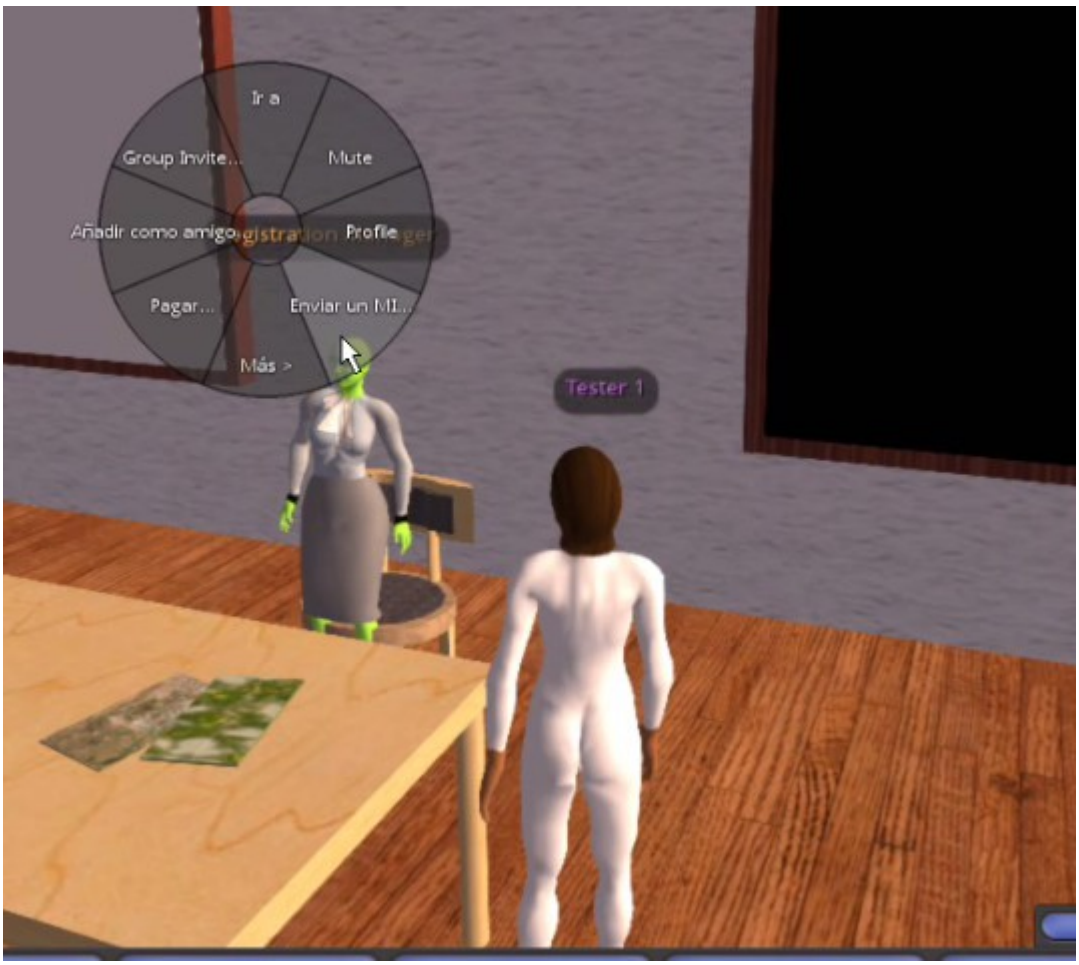


*Figura 33: El avatar inicia un chat con el bot conversacional*

# Apéndice C: Cuestionario post-test

## *Encuesta de satisfacción*

*Nombre:.....*

**1; ¿Cómo de fácil te ha sido situarte y moverte en el espacio 3D?**

| | | | | |
|---|---|---|---|---|
| 1 Muy difícil | 2 Difícil | 3 Normal | 4 Fácil | 5Muy fácil |

**2; ¿Cómo de cómodo te ha resultado caminar o teleportarte por el espacio 3D?**

*Caminar*

| | | | | |
|---|---|---|---|---|
| 1 Muy incómodo | 2 Incómodo | 3 Normal | 4 Cómodo | 5 Muy cómodo |

*Teleportar*

| | | | | |
|---|---|---|---|---|
| 1 Muy incómodo | 2 Incómodo | 3 Normal | 4 Cómodo | 5 Muy cómodo |

**3; ¿Te ha sido fácil entender el panel de información o la información proporcionada por el *bot*?**

| | | | | |
|---|---|---|---|---|
| 1 Muy difícil | 2 Difícil | 3 Normal | 4 Fácil | 5Muy fácil |

**4; ¿Qué te ha parecido la interacción con los *bots*?**

| | | | | |
|---|---|---|---|---|
| 1 Muy difícil | 2 Difícil | 3 Normal | 4 Fácil | 5Muy fácil |

**5; ¿Te parece importante saber qué personajes 3D son *bots (controlados por software)* y cuales avatares (controlados por humanos)?**

| | | | | |
|---|---|---|---|---|
| 1 Nada importante | 2 Poco importante | 3 Indiferente | 4 Importante | 5 Muy importante |

**6; ¿Como de cómodo te ha parecido la comunicación por chat con el *bot*?**

| | | | | |
|---|---|---|---|---|
| 1 Muy incómoda | 2 Incómoda | 3 Normal | 4 Cómoda | 5 Muy cómoda |

**7; ¿Te ha sido fácil realizar la tarea en el mundo virtual?**

| | | | | |
|---|---|---|---|---|
| 1 Muy difícil | 2 Difícil | 3 Normal | 4 Fácil | 5Muy fácil |

**8; Durante el test, ¿hasta qué punto te has sentido inmerso en el espacio 3D?**

| | | | | |
|---|---|---|---|---|
| 1 Nada inmerso | 2 Poco inmerso | 3 Normal | 4 Inmerso | 5 Muy inmerso |

**9; Una vez realizado el test, ¿ha mejorado tu perspectiva de los mundos virtuales y su utilidad?**

| | | | | |
|---|---|---|---|---|
| 1 Nada | 2 Poco | 3 La misma | 4 Mejor | 5 Mucho mejor |

**10; ¿Usarías un sistema similar a este para tareas parecidas a la que has realizado?**

| | | | | |
|---|---|---|---|---|
| 1 Nunca | 2 A veces | 3 A menudo | 4 Frecuentemente | 5 Muy frecuentemente |

**11; ¿Crees que la interfaz 3D (el mundo virtual) te ha ayudado a realizar tu tarea?**

| | | | | |
|---|---|---|---|---|
| 1 Nada | 2 Poco | 3 Algo | 4 Bastante | 5 Mucho |

**12; ¿Cuál es tu impresión global del sistema?**

| | | | | |
|---|---|---|---|---|
| 1 Muy mala | 2 Mala | 3 Normal | 4 Buena | 5 Muy buena |

Comentarios adicionales (qué te ha gustado, qué has echado en falta, ideas o sugerencias ...):

# Apéndice D: Hoja de consentimiento del usuario

Por favor, lea esta hoja y firme-la.

En este test de usabilidad, le pedimos (en el siguiente orden):

- Mantener una pequeña entrevista con usted.
- Realizar una tarea en un entorno virtual.
- Grabar el test (por voz).
- Rellenar un pequeño cuestionario de satisfacción.

La participación en este estudio de usabilidad es voluntaria. Toda la información sera estrictamente confidencial. Las descripciones y los resultados se pueden utilizar para ayudar a mejorar nuestro entorno virtual. No obstante, en ningún momento su nombre o cualquier otro tipo de identificación sera utilizado.

Usted puede retirar su consentimiento para el experimento y dejar de participar en cualquier momento.

He leído y entiendo la información en este formulario y he resuelto todas mis dudas.

Firma del usuario Fecha

Firma del diseñador Fecha

# Apéndice E: Guión del moderador

Antes de empezar pondré en marcha el software de grabación del escritorio, abriré el cliente, entrare con el usuario Tester 1(lo situare en el punto de inicio y minimizaré) y tomaré asiento con el usuario.

1. Presentación: "Hola, mi nombre es Enric y actuare como moderador durante esta sesión de test. Mi compañero/a Pablo/Inma/... actuara como observador/a. Primer de todo me gustaría agradecerte tu participación, tenemos planeado que la sesión dure unos 20 minutos."
2. Firmar documento: "Hemos preparado una hoja de aceptación que deberías firmar antes de empezar el test."
3. Introducir la sesión de test: "Gracias. Ahora, por favor lee este documento, que te servirá para informarte de los objetivos del test y de cual será tu tarea. Quiero recalcar que este test se basa en tus impresiones, así que te animo a que seas expresivo/a y que eres libre de expresar en voz alta cualquier pensamiento que se te pase por la cabeza. ¿Te ha quedado alguna duda sobre el documento?"
4. Preguntas pre-test: "Antes de empezar con la sesión te haré unas preguntas" (observador apunta):
    1. "¿Qué experiencia tienes en el campo del eGovernment?" Si hace falta aclarar → "¿Has hecho alguna vez un trámite utilizando el ordenador?"
    2. "¿Qué te parece la idea de involucrar mundos virtuales al eGovernment?" Si hace falta aclarar → "Por ejemplo, ¿te gustaría renovar-te el DNI mediante un mundo virtual? (en lugar de tener que ir presencialmente, podrás seguir preguntando dudas por ejemplo a un personaje del mundo virtual que podría ser la misma persona que te daría la información si fueras presencialmente)."
5. Training: Maximizo el mundo virtual, que se abrirá ya en la posición de test. En caso contrario, hago teleport hasta el destino. "Ahora te hemos preparado un pequeño escenario de introducción para que conozcas como es el mundo en el que harás la tarea y para que aprendas a moverte e interaccionar con el entorno. Éste es tu avatar; lo puedes mover con el cursor (pongo los dedos sobre el cursor, sin pulsarlo). Dirígete hacia la casa de Training (aprovecho para recalcar lo del idioma inglés, que si hay cualquier duda que se me pregunte sin pensarlo dos veces), e intenta entrar, para ello ponte sobre esta plataforma (señalo la plataforma) y pica a la puerta haciendo clic izquierdo sobre ella (usuario llega a la puerta, se abre). Si te fijas aquí hay dos personajes más, éste es un humano y ése que está sentado ahí es un *bot* controlado por ordenador. También

hay sillas, te puedes sentar si quieres; para ello haz clic derecho sobre la silla. Haciendo clic derecho sobre los objetos, o los otros personajes, puedes descubrir qué acciones puedes realizar con ellos; si te fijas una de las opciones es sentar-se, prueba a hacer clic izquierdo aquí. (espero un poco, y si el usuario no sabe levantarse) Para levantarte clica aquí (señalo a Levantarse). Ahora intenta hablar con la chica (dejo un pequeño tiempo para ver si el usuario accede ya directamente con el botón derecho) Para hablar con ella selecciona la opción Enviar IM... Dile lo que quieras.. (la chica contesta) te dice que esta ocupada ahora mismo; intenta hablar con el *bot.* Ahora con el *bot* funciona de manera distinta, hay una serie de palabras limitadas que él entiende. Para preguntarle al *bot* qué palabras entiende pon help y envía el mensaje con Enter. Aquí te está diciendo que puedes poner hello seguido de espacio y tu nombre o bien bye seguido de tu nombre, probemos a decir hello. (se enciende el panel, si hace falta le digo que mire el panel) Mira, esa es su respuesta. Ahora prueba con bye (se enciende el panel). Bueno pues ésta ha sido la introducción, puedes curiosear un poco por la habitación y cuando quieras sal de la habitación y empezamos el test. (el usuario sale, hago teleport hasta el punto de inicio del test). Como puedes ver existe la posibilidad de teleportarse allá donde queramos."

6. Tarea: "¿Recuerdas la tarea? Sino la podemos releer en un momento" (Releemos la tarea). A partir de este punto intentare no intervenir si no es necesario, me dedicare a observar al usuario pero en una postura más alejada que hasta ahora, para que el usuario note que ya no cuenta con alguien para guiarle, aunque a la vez lo suficientemente cerca como para que sepa que estoy allí y que puede pedirme ayuda si así lo desea. Cuando se encuentre con algún error, esperaré a que el usuario haya agotado sus recursos, es decir no sepa qué hacer, y entonces intervendré. Una vez acabe la tarea → "¡Felicidades! Has completado con éxito la tarea :)"

Cuestionario: "Por último me gustaría que rellenaras un cuestionario para que puedas reflejar tu opinión sobre el test que hemos realizado, y por supuesto si tienes algún otro comentario que quieras hacer me lo puedes compartir conmigo." (el usuario acaba el test) "Pues ya está, ¡muchas gracias por tu participación!"