UNIVERSIDAD NACIONAL DE SANTIAGO DEL ESTERO
FACULTAD DE CIENCIAS EXACTAS Y TECNOLOGÍAS

LICENCIATURA EN SISTEMAS DE INFORMACIÓN

TRABAJO FINAL DE GRADUACIÓN

PROTOTIPO PARA EL DISEÑO UNIVERSAL DE INTERFACES DE USUARIO BASADO EN ONTOLOGÍAS

Autores

LUIS GUSTAVO CHANFERONI

MERCEDES DIAZ

Profesor Guía

MARGARITA ALVAREZ

Asesor

GRACIELA E. BARCHINI

Febrero de 2011

UNIVERSIDAD NACIONAL DE SANTIAGO DEL ESTERO
FACULTAD DE CIENCIAS EXACTAS Y TECNOLOGÍAS

TRABAJO FINAL DE GRADUACIÓN DE LA LICENCIATURA EN SISTEMAS DE INFORMACIÓN

"PROTOTIPO PARA EL DISEÑO UNIVERSAL DE INTERFACES DE USUARIO BASADO EN ONTOLOGÍAS"

Autores:

....

Luis G. Chanferoni **Mercedes Diaz**

Profesor Guía:

.....

Ing. Margarita Alvarez

Asesor: **Msc. Ing. Graciela E. Barchini**

* ——————————————————— * ——————————————————— *

Aprobado el día …….. del mes de …….. del año 20…….. por el Tribunal integrado por

....

(firma) (firma) (firma)

....

(aclaración) (aclaración) (aclaración)

Santiago del Estero – Argentina

A nuestras familias

Luis G. Chanferoni – Mercedes Diaz

AGRADECIMIENTOS

A nuestras **familias** que nos apoyaron en este proceso.

A nuestra **profesora guía** que nos ayudó a avanzar y estuvo siempre cuando la necesitábamos.

A nuestra **profesora asesora** que nos animó y ayudó a avanzar.

A nuestros **compañeros** que colaboraron en algún momento en el proceso de desarrollo, alentando o prestando su ayuda.

A nuestros **amigos** que siempre estuvieron alentándonos.

Gracias a **Dios** por permitirnos compartir con todas las personas que estuvieron involucradas en nuestros estudios y en el desarrollo del trabajo final.

Luis G. Chanferoni – Mercedes Diaz

Santiago del Estero, Argentina

Febrero de 2011

CONTENIDO

RESUMEN

El diseño universal de interfaces de usuario (IU) se presenta como una guía para el desarrollo de IU en la que se consideran diferencias en los requisitos, roles, características, etc., de cada uno de los usuarios. Su objetivo es lograr múltiples IU, en lugar de una IU promedio para todos.

Como lo aseguran muchos autores, la variedad de usuarios crece, los requisitos son más sofisticados (independencia de la plataforma, acceso remoto, etc.) y se amplía la cantidad de parámetros, relaciones y atributos involucrados en el desarrollo de una interfaz; como consecuencia, el esfuerzo dedicado a diseñar la IU aumenta considerablemente.

En este trabajo, se presenta una solución para el diseño universal de IU mediante el desarrollo de un prototipo, tomando como base los denominados "Entornos de Desarrollo de Interfaces de Usuario basados en Modelos" (MB-UIDE) usando ontologías. Además, dentro del mismo se ofrecen procesos y guías metodológicas para diseñar e implementar IU de manera profesional.

Con el desarrollo de este Trabajo Final se espera obtener una herramienta que contemple las características de completitud, correctitud y usabilidad; y además que las IU generadas satisfagan a la más amplia variedad de usuarios.

Palabras Claves: Diseño Universal, Interfaces de Usuario, Ontologías, Modelos, Entornos de desarrollo de Interfaz de usuario basados en modelos.

INTRODUCCIÓN

El diseño de IU de alta calidad, accesibles y usables por una población amplia y diversa se vuelve más complejo, debido a que la cantidad de parámetros y sus relaciones a considerar son numerosos.

Hasta el momento, la disciplina interfaz hombre máquina (IHM) cuenta con los MB-UIDE [PUE97], cuya finalidad es especificar, a través de modelos, los aspectos relacionados con la IU y que influyen en la misma. Abarcan tres dimensiones: editores, métodos y modelos. Además, ofrecen herramientas de automatización de tareas y demás capacidades para desarrollar IU de manera sistemática y profesional.

Para que un MB-UIDE sea completo debe contemplar las siguientes cuestiones:

1. Un editor gráfico para la instanciación de los modelos. [VAN08]
2. Disponer de guías metodológicas que orienten a los diseñadores y stakeholders involucrados en el desarrollo de la interfaz.
3. Que el modelo de IU esté basado en una trilogía semántica, sintáctica y estilística[1]. Por lo tanto, se debe utilizar un Lenguaje de Descripción de IU (UIDL) [INT06] que contemple esta trilogía.

Aun así, los desarrolladores que utilizan estos entornos para crear IU frecuentemente se enfrentan a diversos problemas como:

- Dificultades para considerar múltiples parámetros.
- Falta de consenso y entendimiento compartido sobre las metodologías y modelos a utilizar.
- Falta de fidelidad con respecto a los modelos de tareas, dominio y usuario.
- Complejidad del lenguaje para describir los modelos.

En este trabajo se presenta el desarrollo de un prototipo para el diseño universal [SAV01] de IU basado en ontologías. Se intenta brindar una solución que facilite la manipulación de una cantidad variante y creciente de parámetros. El mismo estará compuesto por:

1. Un UIDL usiXML que brinde el lenguaje para describir los modelos de contexto de uso, de tareas, de dominio, de mapeo, de IU abstracta y de IU concreta.

[1] Ver capítulo II pág. 52

2. Una ontología que represente el UIDL, los usuarios y sus características.
3. Un editor de ontologías para instanciar las ontologías.
4. Un conjunto de reglas de inferencia que permita automatizar tareas como la creación de los modelos de IU abstracta y concreta.
5. Un conjunto de guías metodológicas que guíen al desarrollador en el proceso de diseño de la IU.

El prototipo a desarrollar producirá múltiples modelos de IU concreta para la satisfacción de las preferencias y características de cada uno de los usuarios, en lugar de producir una IU promedio para todos los usuarios.

Para ello, es necesario que la herramienta facilite este proceso de diseño y que brinde las características de completitud, correctitud y usabilidad.

Capítulo I
Problemas y Objetivos

CAPÍTULO I

PROBLEMAS Y OBJETIVOS

I.1. INTRODUCCIÓN

A continuación en este capítulo se presentarán los problemas que llevaron a la realización del presente trabajo. Además se plantean los objetivos que se intentarán alcanzar al finalizar el mismo.

Luego se enuncia el alcance del proyecto y se hace referencia a los trabajos más destacados en el área de desarrollo de este trabajo.

I.2. PLANTEAMIENTO Y FORMULACIÓN DEL PROBLEMA

En la actualidad, debido principalmente a los avances tecnológicos en la disciplina, se necesita desarrollar aplicaciones de software que sean accesibles por *cualquiera*, en *cualquier momento* y en *cualquier lugar,* lo cual ha introducido nuevas dimensiones a la disciplina IHM. Así, es importante desarrollar IU de alta calidad, accesibles y usables por una población diversa, que permita considerar distintas habilidades, requisitos y preferencias, en diferentes contextos de uso, y a través de distintas tecnologías [SAV01].

Se ha comprobado que crear una interfaz de usuario puede ser muy difícil y costoso porque es un trabajo largo, complejo, y también desafiante al momento de implementar, probar y modificar. El esfuerzo requerido para el desarrollo de la misma, en muchos casos, excede el 50% del esfuerzo de programación necesario para el desarrollo del sistema entero [MYE92].

En base a la investigación exploratoria realizada [DEL07, FUR07, SAV01, VAN08], se puede afirmar que los desarrolladores o diseñadores de interfaces de usuario que utilizan MB-UIDE, se enfrentan a los siguientes problemas:

1) La mayoría de las herramientas para desarrollar IU tienen dificultades para considerar múltiples parámetros.
2) Falta de consenso y entendimiento compartido sobre las metodologías y herramientas a utilizar. Frecuentemente, los desarrolladores se enfrentan con el obstáculo de tener que aprender y usar nuevas herramientas y metodologías para aplicaciones concretas.

3) Falta de fidelidad, esto produce incorrección, es decir:
    a. Las interfaces finales no se corresponden con la información en los modelos diseñados.
4) Complejidad del lenguaje para describir los modelos, que muchas veces son difíciles de aprender y usar.

En el presente trabajo se abordarán los tres primeros problemas, tratando de dar solución a estos se plantea la siguiente pregunta que guía la investigación:

¿Qué funcionalidades debe brindar el prototipo para ser completo, correcto, consistente y usable?

I.3. OBJETIVOS

Objetivos Generales

- Permitir el desarrollo de IU para la más amplia cantidad de usuarios con múltiples características.
- Fomentar la creación, el uso y reuso, y la compartición de las ontologías que representan los contextos de uso definidos para diferentes diseños de IU.

Objetivos Específicos

- Desarrollar una herramienta con las siguientes características:

    a) Completitud: con respecto a los múltiples parámetros de los modelos de usuarios, de tareas y de dominio.

    b) Correctitud: que se abstraigan correctamente los modelos de usuario, de tareas y de dominio.

    c) Usabilidad: que posea herramientas que faciliten el aprendizaje y modificación de las funcionalidades del prototipo.

I.4. ALCANCE

El prototipo permitirá la creación de los modelos necesarios (de contexto de uso, de tareas, de dominio, de mapeo, de interfaz de usuario abstracta y concreta) para el diseño una IU. Además poseerá una ontología que facilite la instanciación de las características de los usuarios.

Se desarrollarán un conjunto de reglas de transformación que permitirán automatizar las tareas de generación de modelos de IU abstracta y concreta.

Ofrecerá también, guías metodológicas para ayudar al desarrollador en el momento de instanciar los modelos y ejecutar las reglas de transformación.
Las pruebas se realizarán en el dominio de aplicación de e-learning, con usuarios con diferentes características, para observar cómo se adaptan las interfaces a cada uno de ellos. Esto permitirá validar la utilidad de la herramienta y saber si cumple con los objetivos planteados en este trabajo.

I.5. ANTECEDENTES

Al igual que la Arquitectura Basada en Modelos es la tendencia en la Ingeniería del Software, con la que se está aumentando el nivel de abstracción, en el desarrollo de IU se lleva utilizando desde hace más de una década una idea similar para generar automáticamente o semi-automáticamente una IU. La diferencia fundamental entre el estado de ambas tendencias está en que en IS, tras una larga etapa de unificación y estandarización, se dispone de un lenguaje unificado, y en IHM esto no ocurre. La IHM, en este sentido, está más atrasada y se encuentra en una fase de fragmentación y búsqueda de una notación unificada.
Además, ninguna de las propuestas apuesta por un entorno de desarrollo que las ponga en práctica, aunque desde hace unos años y en el campo del desarrollo de IU los MB-UIDE están logrando buenos resultados, pero sólo a nivel académico.
En concreto, los aportes que se relacionan en forma directa con este trabajo son:
OntoWeaver [LEI04]: Este enfoque de modelado recae en un conjunto de ontologías de sitio comprensivas, para modelar todos los aspectos de sitios web de gran volumen de datos. En particular, las ontologías de sitio de OntoWeaver poseen dos componentes: una ontología de vista de sitio y una ontología de presentación. La ontología de vista del sitio provee meta modelos para permitir la composición de vistas de sitio sofisticadas, lo que permite a los usuarios finales navegar y manipular las bases de datos del área de aplicación en cuestión. La ontología de presentación abstrae el look and feel para las vistas del sitio y hace posible que la apariencia visual y el diseño sean especificados a un alto nivel de abstracción.
The Unified User Interface Design Method [SAV01]: Este está caracterizado por dos propiedades que distinguen tanto la conducta del método y su respectivo resultado. Primero, el método adopta una perspectiva de diseño analítica, en el sentido de que requiere una visión sobre cómo los usuarios realizan las tareas en modelos de tareas

existentes. En segundo lugar, el método soporta un enfoque jerárquico disciplinado para poblar y articular espacios de diseño racionales.

En general, el método introduce la noción de descomposición de tareas polimórficas, a través del cual cualquier tarea (o subtarea) puede ser descompuesta en un numero arbitrario de sub-jerarquías alternativas.

Ontology-Based Method for Universal Design of User Interfaces [FUR07]: Se trata de un método estructurado para identificar parámetros requeridos para el diseño universal basado en ontologías. El método es soportado por un conjunto de herramientas, todas basadas en una ontología del área de aplicación. Posee modelos que capturan la instanciación de conceptos identificados en esta ontología para producir múltiples interfaces de usuario para una sola situación de diseño. Utiliza un editor de ontologías que deriva en archivos ASCII los modelos producidos.

KnowiXML: A Knowledge-Based System Generating Multiple Abstract User Interfaces in USIXML[FUR04]: Esta investigación presenta un enfoque multidisciplinario con la intención de generar múltiples Interfaces de Usuario Abstractas (AUIs), las cuales son adaptables para diferentes tipos de usuario, realizando diferentes tareas, usando dispositivos específicos en diferentes ambientes físicos. El framework para generar IU, llamado IKnowU, está basado en un proceso unificado para el diseño de sistemas interactivos, el cual integra las mejores prácticas de la Ingeniería de Software, y la disciplina IHM. Este framework esta soportado por KnowiXML, y por un sistema basado en conocimiento que facilita la aplicación de modelos y la ubicación de elementos visuales apropiados durante la generación de AUIs. Estas AUIs son generadas usando métodos de resolución de problemas estudiados en la Inteligencia Artificial. El conocimiento de diseño codificado en KnowiXML manipula uniformemente las especificaciones de los modelos y la IU a través de un UIDL.

En los dos últimos antecedentes, se indica la importancia de tener múltiples derivaciones de un modelo de tarea para cubrir las diferencias individuales de cada usuario y además utilizan ontologías para representar los modelos. Pero, lamentablemente, no existen herramientas, no están disponibles o son apuestas meramente académicas, tampoco se observa el potencial del uso de ontologías como el razonamiento automático, el uso de agentes, etc.

La propuesta aquí presentada intenta tomar el esfuerzo de las propuestas anteriores, integrándolas en el desarrollo del prototipo.

Capítulo II
Marcos Referenciales

CAPÍTULO II

MARCOS REFERENCIALES

II.1. MARCO TEÓRICO

II.1.1. INTRODUCCIÓN

Para el diseño y construcción del prototipo que producirá múltiples modelos de IU concreta para la satisfacción de las preferencias y características de los usuarios se deben conocer principalmente conceptos de ontologías y conceptos relacionados con la disciplina Interacción Hombre-Máquina, tales como: interfaz de usuario y diseño universal.

La ontología como "el estudio metafísico de la naturaleza de ser y la existencia" es tan antigua como la disciplina de la filosofía. Recientemente, la ontología se ha definido como "la ciencia de lo que es, de los tipos y estructuras de objetos, propiedades, eventos, procesos, y relaciones en cada área de la realidad". Mientras sigue siendo un área fecunda de investigación en el campo de la filosofía, la ontología es actualmente materia de investigación, desarrollo, y aplicación en disciplinas relacionadas con la computación, la información y el conocimiento.

Las ontologías generalmente se usan para especificar y comunicar el conocimiento de un dominio de manera genérica y son muy útiles para estructurar y definir el significado de los términos.

Por otra parte, el área de la interacción hombre-máquina es la disciplina que estudia, en términos generales, el intercambio de información entre las personas y los computadores. Una de sus ramas más modernas es el diseño universal. Este paradigma de diseño, tiene como objetivo principal que el diseño de IU sean usables por el rango más amplio de personas, funcionando en el rango más variado de situaciones y que sean comercialmente practicables.

Hoy en día, se entiende cada vez más, que las IU deben acomodarse a las diferencias que presenta cada persona de tal modo que puedan ser usadas sin problemas. Por lo tanto, las IU deben ser accesibles y usables por una amplia y diversa población de usuarios con características diferentes, en cuanto a los requisitos y preferencias.

Ante esta situación, en este trabajo, se propone el diseño y construcción de un prototipo que permita la creación de IU con esas características.

II.1.2. DISCIPLINA INTERACCIÓN HOMBRE MÁQUINA

II.1.2.1. Definiciones y conceptos

La definición más utilizada de Interacción Hombre-Máquina (IHM), también llamada Interacción Persona-Ordenador, es la de ACM, Association for Computer Machinery [ACM09]. Esta asociación tiene un grupo especial de trabajo en temas de IHM denominado SIGCHI, Special Interest Group in Computer Human Interaction. Poseen un documento online con los conceptos básicos y que mantienen actualizado en función de los avances en la disciplina y la comprensión de la misma.

Sostienen que no existe un acuerdo sobre los tópicos que forman el área de IHM. Pero ofrecen una definición práctica:

"*La IHM es aquella disciplina relacionada con el diseño, evaluación e implementación de sistemas de computación interactivos para uso humano y el estudio de los principales fenómenos asociados con ello.*"

La IHM se ocupa del rendimiento conjunto de tareas realizadas por humanos y máquinas; la estructura de comunicación entre humano y máquina; capacidades humanas para usar máquinas (incluyendo el aprendizaje de interfaces); algoritmos y programación de la interfaz en sí misma; cuestiones de ingeniería que surgen en el diseño y construcción de interfaces; el proceso de especificación, diseño, e implementación de interfaces; y el diseño de compromisos. Es decir, se centra en la usabilidad y en la necesaria aceptación de un producto en función de las características que como vehículo ofrece y de cómo las ofrece.

Su objetivo es que el intercambio de información entre hombre y máquina sea más eficiente: minimizando los errores, incrementando la satisfacción, disminuyendo la frustración y en definitiva, haciendo más productivas las tareas que envuelven a las personas y las computadoras.

II.1.2.2. Áreas de la IHM

Se trata en general de un área interdisciplinaria, figura II.1. Está emergiendo como especialidad junto con otras disciplinas, cada una con diferentes énfasis: la ciencia de la computación (diseño de aplicación e ingeniería de interfaces humanas), psicología (la aplicación de teorías de procesos cognitivos y el análisis empírico del comportamiento humano), ergonomía y factores humanos (definir y diseñar herramientas y artefactos para diferentes tipos de ambiente: trabajo, descanso y doméstico), ingeniería (estudia técnicas de diseño y desarrollo del software), diseño (actividad encaminada a conseguir la producción en serie de objetos útiles y atractivos), sociología, etnografía, antropología

(interacciones entre tecnología, trabajo, y organización. Estudia las costumbres y las tradiciones de los pueblos), inteligencia artificial (utilizada en el diseño de tutores y sistemas expertos en interfaces inteligentes y en el diseño de interfaces de lenguaje natural utilizando la voz).

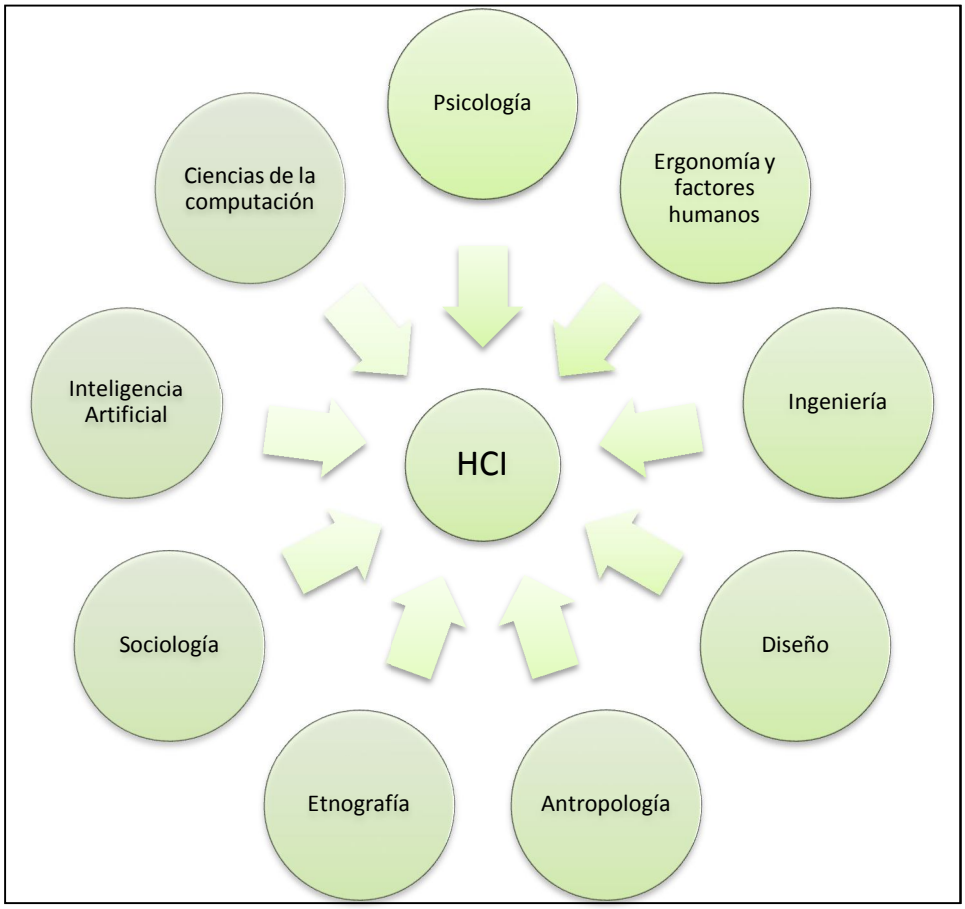


Figura II.1. Áreas de la IHM basado en **[LOR02]**.

II.1.2.3. Elementos

Con la finalidad de catalogar los elementos de la disciplina IHM se especifican a continuación los aspectos interrelacionados, ver figura II.2.

1) *Contexto y Uso de las computadoras (U1, U2 y U3)*: Los usos que se les dan a las computadoras, en el mundo de la computación son llamados "aplicaciones". Estos usos y la medida en la cual la interfaz (y la lógica de aplicación en el resto del sistema) encaja en ellos pueden tener un impacto profundo en cada parte de la interfaz y su éxito. Más aun, el contexto general, social, laboral y de negocio puede ser importante. Además de los requisitos técnicos, una interfaz debería satisfacer objetivos de calidad de trabajo de una unión laboral o considerar restricciones legales de "look and feel" o posicionar la imagen de la compañía en cierto mercado.

El contexto incluye características fundamentales de las personas y de los sistemas. Esas características son tanto las limitaciones físicas (la capacidad de memoria, los límites de transferencia de información, la habilidad lógica de la máquina) como las conceptuales (el conocimiento de las tareas y de los modelos mentales de los objetos y de los procesos).

2) *Características Humanas (H1, H2 y H3)*: Características del procesamiento de la información por parte de los humanos, como la acción de los humanos está estructurada, la naturaleza de la comunicación humana, y los requisitos psicológicos y físicos de los humanos.

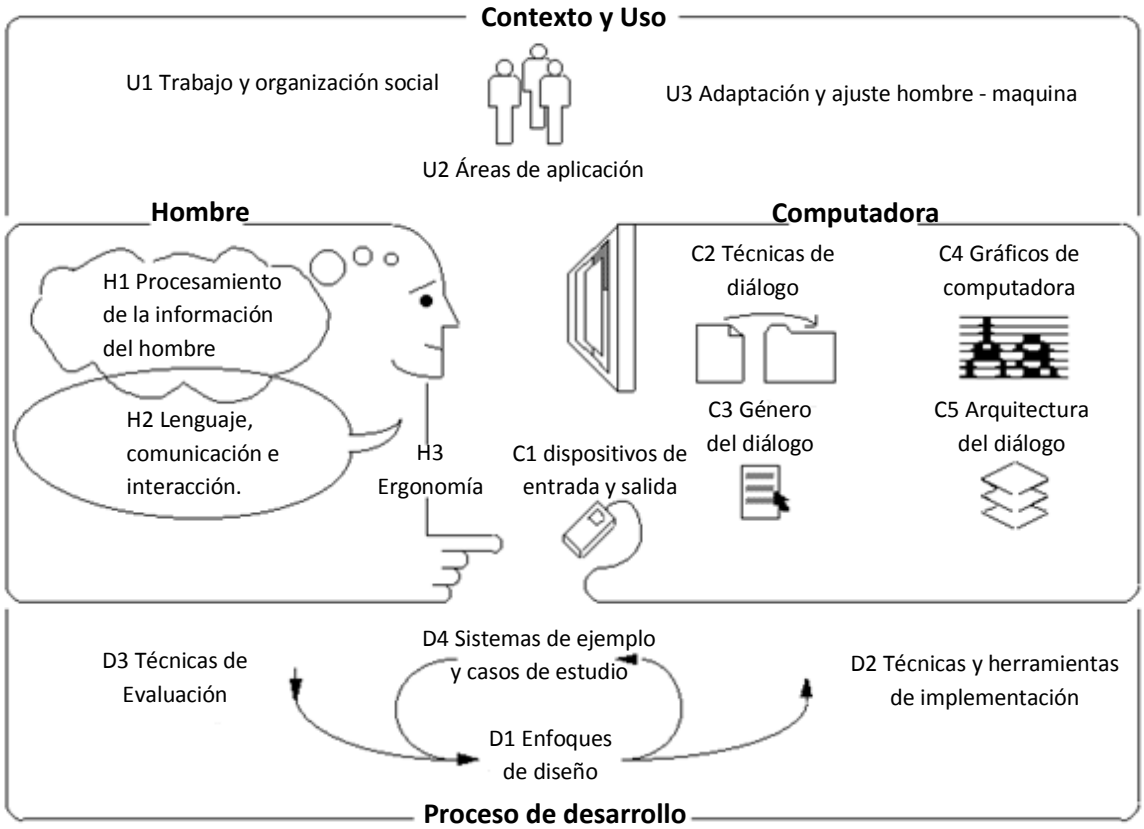


Figura II.2. Aspectos de la IHM, basado en **[ACM09]**

3) *Sistema computacional y Arquitectura de Interfaz (C1, C2, C3, C4 y C5)*: Las máquinas poseen componentes especializados para interactuar con humanos. Algunos de estos componentes son básicamente transductores para mover la información físicamente entre el humano y la máquina. Otros componentes deben encargarse de la estructura de control y representación de aspectos de interacción.

4) *Proceso de desarrollo (D1, D2, D3 y D4)*: La construcción de interfaces humanas es tanto una cuestión de diseño como de ingeniería. Estas cuestiones tratan sobre la metodología y práctica del diseño de interfaz. Otros aspectos del proceso de

desarrollo incluyen la relación del desarrollo de interfaz con la ingeniería (ambas software y hardware) del resto del sistema [ACM09].

II.1.2.4. Disciplina IHM y la ingeniería de software

La Ingeniería de Software (IS) y la IHM son dos disciplinas que pretenden aportar métodos y herramientas para potenciar la calidad de los productos software (figura II.3). Los puntos de vista desde los que se quiere alcanzar estos objetivos son diferentes, y ambas disciplinas se solapan en parte de sus ámbitos.

La IS aporta procesos, métodos, notaciones y herramientas que afectan tanto al proceso como al producto, contempla factores internos y externos de un producto software.

Por otra parte, la IHM ofrece otros métodos, en algunos casos complementarios, y que se centran en el proceso, en el producto y en la identificación de dónde, cómo y por quién es utilizado el producto software. La IS está preocupada por ofrecer una visión arquitectónica del sistema, para ello ha hecho varias propuestas, entre ellas, el Proceso Unificado (PU) y el Lenguaje Unificado de Modelado (UML).

La IHM busca la complicidad del usuario y la determinación de su contexto. Para ello propone un Diseño Centrado en el Usuario (DCU) [THE09].

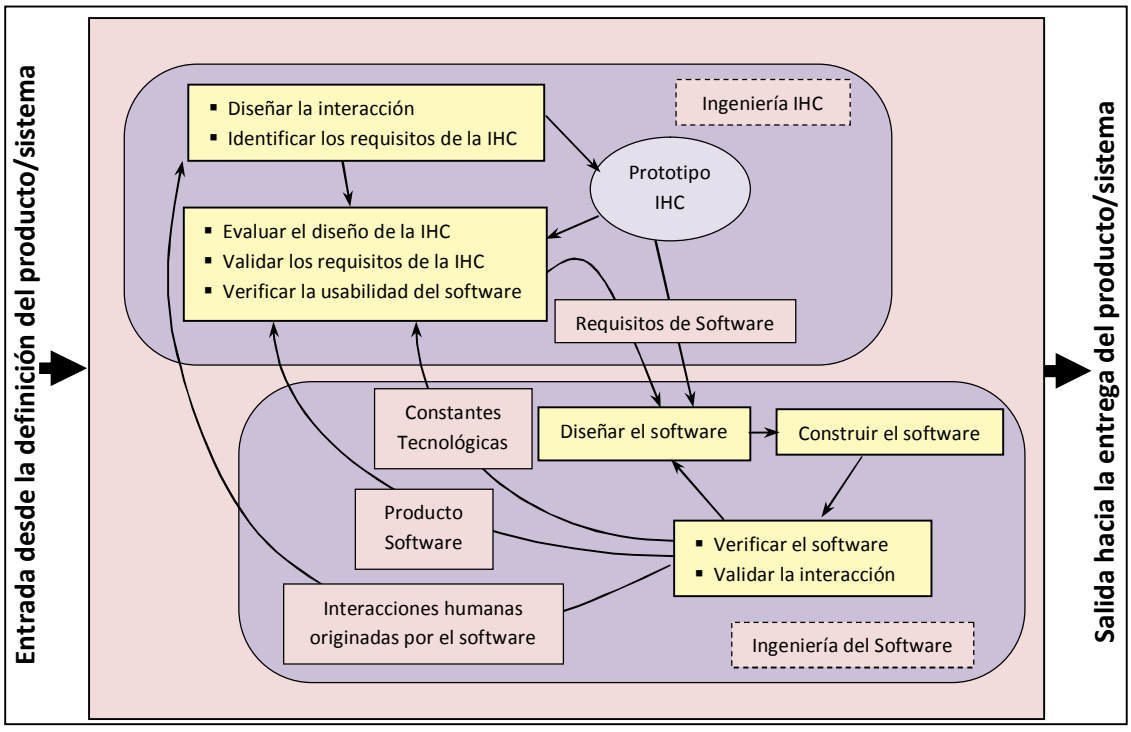


Figura II.3. Ingeniería de software e IHM basado en [BUI97]

Algunos problemas relacionados con el desarrollo de IU, a los que la IHM ha querido dar respuesta, son:

- el modelado de tareas,
- la consideración del usuario y
- el tratamiento de los problemas relacionados con la usabilidad.

II.1.2.5. Interfaz de Usuario

II.1.2.5.1. Definición

La interfaz de usuario es el principal punto de contacto entre el usuario y el ordenador; es la parte del sistema que el usuario ve, oye, toca y con la que se comunica. El usuario interacciona con el ordenador para poder realizar una tarea. Dependiendo de la experiencia del usuario con la interfaz, el sistema puede tener éxito o fallar en ayudar al usuario a realizar la tarea. El tipo de problemas que origina una interfaz de usuario pobre incluye la reducción de la productividad, un tiempo de aprendizaje inaceptable y niveles de errores inaceptables que produce frustración y probablemente el desechar el sistema [LOR02].
La interfaz de usuario de un sistema consiste de aquellos aspectos del sistema con los que el usuario entra en contacto, físicamente, perceptivamente o conceptualmente. Los aspectos del sistema que están escondidos para el usuario se denominan la implementación [MOR81].

II.1.2.5.2. Participantes

Según Delgado [DEL07] se pueden encontrar cuatro tipos de participantes involucrados en los distintos procesos existentes en el desarrollo y uso de los IU. A continuación se aclara el significado de los diferentes términos utilizados en la bibliografía.

- **End-user** (usuario final): La persona que usa el programa resultante. También se les suele llamar simplemente user (usuario).
- **User interface designer** (diseñador de IU): Las personas que crean la IU de la aplicación. Aparecen en algunos textos denominados como designer (diseñador).
- **Application programers** (programadores de la aplicación): Trabajan junto al diseñador escribiendo el código para el resto de la aplicación.
- **Tool creators** (creadores de herramientas): Los diseñadores pueden usar herramientas especiales que les ayudan en la creación de los IU creadas por los tool creators.

Hay que tener en cuenta que los diseñadores de la IU son usuarios del software construido por los creadores de herramientas, por ello, en muchos de los documentos referenciados se trata de evitar el término usuario. También en muchos textos se puede encontrar el término programador usado para cualquier persona que escribe código, ya sea un diseñador, un programador de aplicaciones o un creador de herramientas.

II.1.2.6.Diseño universal

II.1.2.6.1. Definición

El diseño universal o Interfaces de usuario para todos [SAV01] se define como un enfoque sistemático para el diseño, implementación, y evaluación de interfaces de usuario que considere los requisitos de la más amplia población de usuarios.

II.1.2.6.2. Tendencias en el Diseño Universal

En la figura II.4 se pueden observar tres tendencias que dan lugar al término de diseño universal de interfaces. Estas son:

I. Proliferación tecnológica

La proliferación tecnológica contribuye con un mayor rango de sistemas o dispositivos para facilitar el acceso a la comunidad a todo el conjunto de recursos de información. Estos dispositivos incluyen computadoras, teléfonos estándar, teléfonos celulares, televisores, quioscos de información, aparatos especiales de información, y otros dispositivos con conexión. Dependiendo del contexto de uso, los usuarios pueden emplear cualquiera de los antes mencionados para revisar o explorar, manipular y configurar herramientas de información, en cualquier momento.

II. Interacción Social

Los sistemas basados en computadoras tienden cada vez más a ser herramientas de comunicación, colaboración e interacción social entre grupos de personas. Desde el punto de vista del especialista, la computadora se está transformando en una herramienta de información para el ciudadano en la Sociedad de la Información.

De ello se deduce que los diseñadores cada vez más tienen que proporcionar herramientas de información que puedan ser utilizadas por diversos grupos de usuarios, incluidas las personas con diferentes culturas, educación, capacitación y experiencia laboral, usuarios de computadoras novatos y experimentados, los niños y los ancianos, y personas con diferentes tipos de discapacidad.

III. Lugares de uso

El uso "tradicional" de las computadoras se complementa con el uso residencial y nómada, por lo tanto, es posible ingresar en una gama más amplia de actividades humanas y en una gran variedad de entornos, tales como la escuela, el hogar, el mercado, y de otros lugares civiles y sociales. Como resultado, las herramientas de información deben tener la capacidad de interactuar con el usuario en todos los lugares, independientemente de la ubicación, máquina destino, o entorno de ejecución. Es probable que la usabilidad en esos lugares de uso "no tradicionales" resulte un objetivo difícil de cumplir a diferencia del caso del lugar de trabajo.

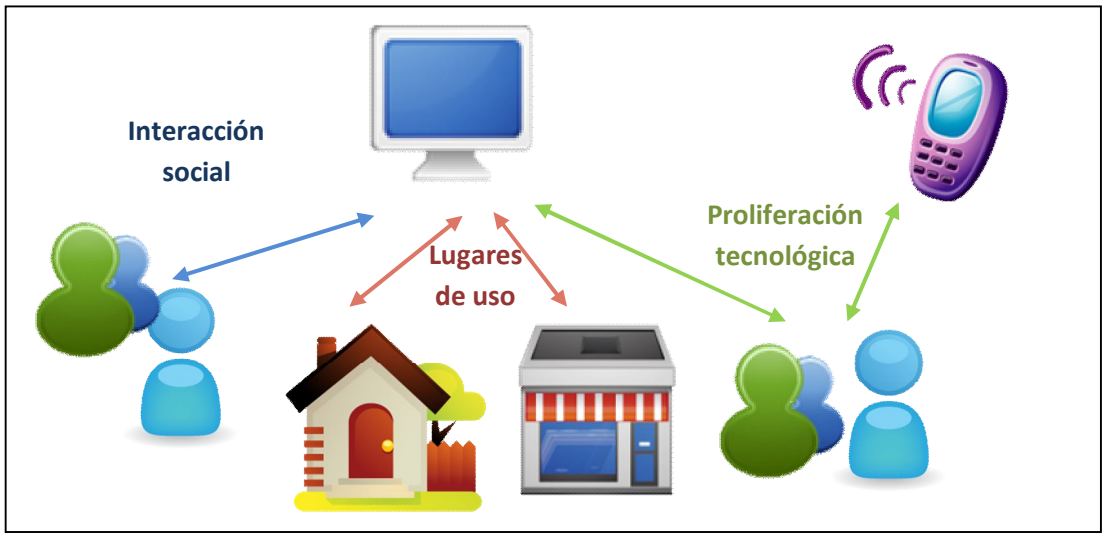


Figura II.4. Tendencias en el diseño universal

De las descripciones de las tendencias se deduce que existe una relación entre todas ellas y que son las características y preferencias de los usuarios.

En el presente trabajo se quieren solucionar algunas dificultades a las que se enfrenta el desarrollador al momento de crear las IU y se pretende crear un prototipo que considere todas las preferencias y características deseadas.

II.1.2.6.3. Principios del Diseño Universal

Los siguientes Principios de Diseño Universal [CON97], se establecieron para servir como guía a un amplio espectro de disciplinas del diseño; entre los cuales se incluyen entornos, productos y comunicaciones. Los siete principios pueden ser usados para evaluar diseños existentes, como guía en el proceso de diseño y para educar tanto a diseñadores como consumidores sobre las características de entornos y productos de uso más fácil.

PRINCIPIO UNO: Uso equitativo

El diseño es útil y vendible a personas con diversas capacidades.

Guías:

1a. Proporciona las mismas formas de uso para todos: idénticas cuando sea posible, equivalentes cuando no.

1b. Evita segregar o estigmatizar a cualquier usuario.

1c. Todos los usuarios deben contar con las mismas garantías de privacidad y seguridad.

1d. Que el diseño sea agradable para todos.

PRINCIPIO DOS: Uso Flexible

El diseño se acomoda a un amplio rango de preferencias y habilidades individuales.

Guías:

2a. Ofrece opciones en la forma de uso.

2b. Sirve tanto para los diestros como para los zurdos.

2c. Facilita al usuario la precisión y exactitud.

2d. Se adapta al ritmo de uso del usuario.

PRINCIPIO TRES: Uso Simple e Intuitivo

El uso del diseño es fácil de entender, sin importar la experiencia, conocimientos, habilidades del lenguaje o nivel de concentración del usuario.

Guías:

3a. Elimina la complejidad innecesaria.

3b. Es consistente con la intuición y expectativas del usuario.

3c. Se acomoda a un rango amplio de grados de alfabetización y conocimientos del lenguaje.

3d. Ordena la información de acuerdo a su importancia.

3e. Proporciona información y retroalimentación eficaces durante y después de la tarea.

PRINCIPIO CUATRO: Información Perceptible

El diseño transmite la información necesaria de forma efectiva al usuario, sin importar las condiciones del ambiente o las capacidades sensoriales del usuario.

Guías:

4a. Utiliza diferentes medios (pictóricos, verbales, táctiles) para la presentación de manera redundante de la información esencial.

4b. Maximiza la legibilidad de la información esencial.

4c. Diferencia elementos de manera que puedan ser descritos por sí solos (por ejemplo que las instrucciones dadas sean fáciles de entender).

4d. Proporciona compatibilidad con varias técnicas o dispositivos usados por personas con limitaciones sensoriales.

PRINCIPIO CINCO: Tolerancia al Error

El diseño minimiza riesgos y consecuencias adversas de acciones involuntarias o accidentales.

Guías:

5a. Ordena los elementos para minimizar el peligro y errores: los elementos más usados están más accesibles; los elementos peligrosos son eliminados, aislados o cubiertos.

5b. Advierte de los peligros y errores.

5c. Proporciona características para controlar las fallas.

5d. Descarta acciones inconscientes en tareas que requieren concentración.

PRINCIPIO SEIS: Mínimo Esfuerzo Físico

El diseño puede ser usado cómoda y eficientemente minimizando la fatiga.

Guías:

6a. Permite al usuario mantener una posición neutral de su cuerpo.

6b. Usa fuerzas de operación razonables.

6c. Minimiza las acciones repetitivas.

6d. Minimiza el esfuerzo físico constante.

PRINCIPIO SIETE: Adecuado Tamaño de Aproximación y Uso

Proporciona un tamaño y espacio adecuado para el acercamiento, alcance, manipulación y uso, independientemente del tamaño corporal, postura o movilidad del usuario.

Guías:

7a. Proporciona una línea clara de visibilidad hacia los elementos importantes, para todos los usuarios de pie o sentados.

7b. Proporciona una forma cómoda de alcanzar todos los componentes, tanto para los usuarios de pie como sentados.

7c. Acomoda variantes en el tamaño de la mano y asimiento.

7d. Proporciona un espacio adecuado para el uso de aparatos de asistencia o personal de ayuda.

Estos Principios de Diseño Universal abarcan sólo diseños de uso universal, mientras que la práctica del diseño involucra no sólo la consideración de facilidad de uso. Los diseñadores deben incorporar otras consideraciones como economía, ingeniería, cultura, género y aspectos ambientales en sus procesos de diseño. Estos principios ofrecen al diseñador una guía para integrar aspectos que satisfagan las necesidades de la mayor cantidad de usuarios posibles.

II.1.3. ONTOLOGÍAS

II.1.3.1. Definición

La definición de ontología propuesta por Gruber [GRU95] en 1993 que es una de las más conocidas es la siguiente: "Una ontología es la especificación explícita de una conceptualización."

Continuando con lo que explica el autor en diferentes trabajos se entiende que este término proviene de la filosofía, en donde una ontología es tomada como una teoría sobre la naturaleza de lo existente.

Este término luego fue adoptado en Inteligencia Artificial donde es usado para referirse tanto a una teoría de un mundo modelado como a un componente de sistemas de conocimiento. En general en la ciencia de la computación e información, ontología es un término técnico que denota un artefacto que es diseñado para un propósito, el cual es el de permitir el modelado de conocimiento sobre un dominio, real o imaginario.

Se define aquí un conjunto de primitivas representativas con las cuales modelar un dominio de conocimiento del discurso. Las primitivas representativas son comúnmente clases (o conjuntos), atributos (o propiedades), y relaciones (o relaciones entre miembros de clases). Las definiciones de las primitivas representativas incluyen información sobre su significado y restricciones sobre su aplicación coherente y lógica.

Gruber indica los siguientes componentes de una ontología:

- *Conceptos:* son las ideas básicas que se intentan formalizar. Los conceptos pueden ser clases de objetos, métodos, planes, estrategias, procesos de razonamiento, etc.
- *Relaciones:* representan la interacción y enlace entre los conceptos del dominio. Suelen formar la taxonomía del dominio. Por ejemplo: subclase-de, parte-de, parte-exhaustiva-de, conectado-a, etc.
- *Funciones:* son un tipo concreto de relación donde se identifica un elemento mediante el cálculo de una función que considera varios elementos de la ontología. Por ejemplo, pueden parecer funciones como categorizar-clase, asignar-fecha, etc.

- *Instancias:* se utilizan para representar objetos determinados de un concepto.
- *Axiomas:* son teoremas que se declaran sobre relaciones que deben cumplir los elementos de la ontología. Por ejemplo: "Si A y B son de la clase C, entonces A no es subclase de B", "Para todo A que cumpla la condición C1, A es B", etc.

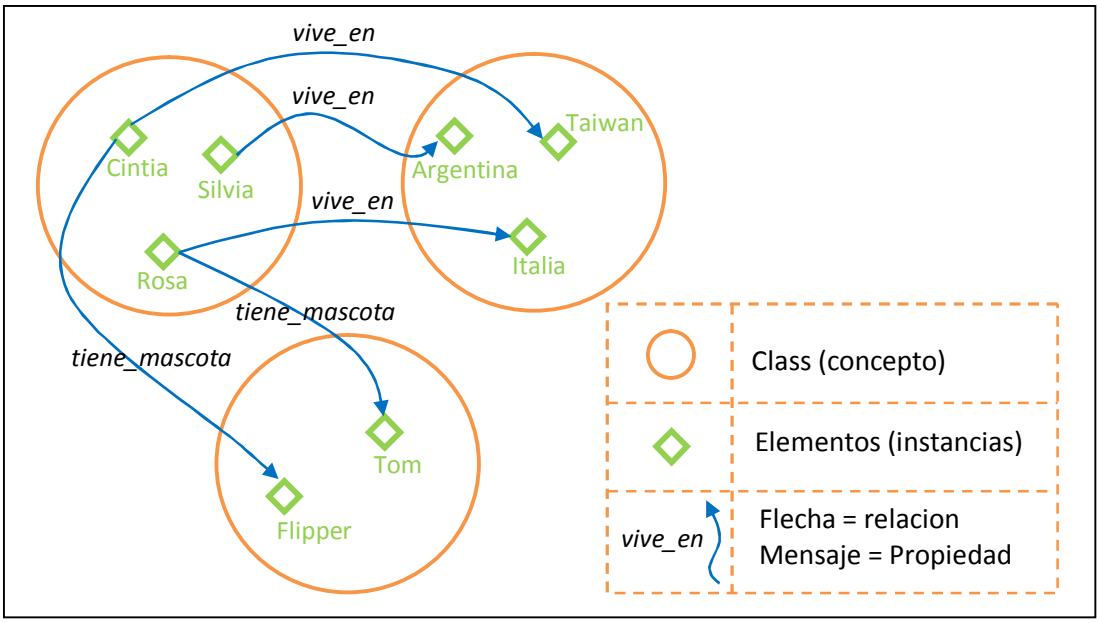


Figura II.5. Componentes de una ontología

En la figura II.5 se puede observar un ejemplo gráfico de una ontología. Los conjuntos representan clases o conceptos, en este caso *Personas*, *Países* y *Mascotas.* Cada una de estas clases tiene instancias o elementos que son particularizaciones de esas clases, a su vez las instancias están relacionadas y esa relación se representa con una propiedad, como en el gráfico, *vive_en* y *tiene_mascota.*

Borst [BOR97] modificó ligeramente la definición de Gruber: "Las ontologías se definen como la especificación formal de una conceptualización compartida".

Studer, Benjamins, y Fensel [STU98], agregaron expresividad a las definiciones de Gruber y Borst explicitando:

- *Conceptualización* se refiere a un modelo abstracto de algún fenómeno en el mundo, proveniente de haber identificado los conceptos relevantes de dicho fenómeno.
- *Explícita* se refiere a que los conceptos usados y las restricciones para su uso se definen explícitamente.
- *Formal* se refiere al hecho de que la ontología debería ser legible o interpretable por una computadora.

- *Compartida* refleja la noción de que una ontología captura conocimiento consensuado, es decir, no es conocimiento privado de un individuo, sino aceptado por un grupo o comunidad.

II.1.3.2. Clasificación de las Ontologías

Algunos autores como Gómez Pérez, Mizoguchi y Sowa [GOM99, MIZ95, SOW00], han clasificado las ontologías existentes según diferentes criterios. A continuación se exponen algunos de ellos.

Clasificación por Grado de Axiomatización

Según Sowa [SOW00], las ontologías se pueden clasificar en Terminológicas o Formales de acuerdo al grado de axiomatización que tenga la definición de sus categorías.

- Ontologías Terminológicas: Una ontología terminológica define términos y sus relaciones en taxonomías que involucran tanto relaciones de subtipo y supertipo, como las que relacionan partes con un todo (part-whole), pero no incluyen axiomas y definiciones expresadas en lógica o algún tipo de lenguaje formal procesable por una computadora. Esto hace que se disponga de menos información del universo modelado, pero permite a su vez, por la relativa simplicidad de su especificación, construir ontologías de gran tamaño. La mayoría de los campos de la ciencia, ingeniería, negocios y jurídico, han desarrollado sistemas de terminología o nomenclatura para designar, clasificar y estandarizar sus conceptos en enormes ontologías terminológicas.
- Ontologías Formales: Una ontología formal tiene sus categorías restringidas por axiomas y definiciones expresadas en lógica formal o en algún tipo de lenguaje procesable por la computadora. Las ontologías formales suelen tener menos cantidad de conceptos que las terminológicas, pero sus axiomas y definiciones pueden soportar cómputos e inferencias más complejas.

La diferencia entre una ontología terminológica y una formal, según Sowa [SOW00], es en el grado de especificación. Teóricamente, en la medida que se adicionen axiomas a una ontología terminológica, podrá evolucionar a una formal, pero la definición de axiomas no es una tarea trivial, y es por eso que en la práctica es difícil tal evolución.

Clasificación por Dependencia del Contexto

Según Mizoguchi, Vanwelkenhuysen e Ikeda [MIZ95] las ontologías pueden clasificarse conforme al grado de dependencia del contexto que presenten, en el sentido de que aquellas menos dependientes del contexto serán las candidatas a ser más reusadas.

La clasificación que surge es:

- *Ontologías de dominio*: Expresan conceptualizaciones que son específicas a un dominio particular, colocando restricciones en la estructura y contenido de un dominio de conocimiento mediante axiomas que se cumplen siempre entre los elementos de dicho dominio. Su principal objetivo es permitir el reuso de la ontología para diferentes aplicaciones que involucren al mismo dominio.
- *Ontologías generales o de sentido común:* Definen un vocabulario relacionado a cosas, eventos, tiempo, espacio, causalidad, comportamiento, función, etc.
- *Meta ontologías, Ontologías genéricas (CoreOntologies):* Son similares a las de dominio, pero los conceptos que definen se consideran genéricos a través de diferentes áreas de conocimiento y por ello reusables en diferentes dominios. Típicamente, las ontologías genéricas definen conceptos como estado, evento, proceso, acción, etc. Los conceptos de las ontologías de dominio son a menudo definidos como especializaciones de conceptos existentes en ontologías genéricas. Cabe destacar que el límite para considerar una ontología genérica no está bien definido, pero la distinción es intuitivamente significativa y útil cuando es necesario organizar ontologías en bibliotecas.

Clasificación por el Sujeto de Conceptualización

Algunos autores como Gomez-Perez, entre otros [GOM99], también clasifican las ontologías por el tipo de dominio que modelan, o por el sujeto de conceptualización, algunas de las categorías de ontologías identificadas son:

- *Ontologías de tareas:* Proveen un sistemático vocabulario de los términos usados para resolver problemas asociados con tareas particulares, ya sean dependientes o no del dominio. Por ejemplo, relacionados a la tarea de evaluación, se tendrán términos que involucren a los conceptos de “medición”, “cálculo” y “objetivo”, así como el término que refiera a la acción “generación de informe”.
- *Ontologías de aplicación:* Contienen todas las definiciones que son necesarias para modelar el conocimiento requerido para una aplicación particular en un dominio dado. Típicamente son una mezcla de conceptos provenientes de ontologías de dominio y de ontologías genéricas. Las ontologías de aplicación no se construyen con el propósito de lograr reusabilidad en diferentes dominios.
- *Ontologías de representación de conocimiento:* Describen las primitivas de representación usadas para formalizar conocimiento en paradigmas de desarrollo de

sistemas basados en el conocimiento, es decir, explican la conceptualización que subyace en un formalismo de representación de conocimiento. Se pretende que sean neutrales con respecto a las entidades del mundo, es decir, que provean un marco representacional sin hacer aseveraciones acerca del mundo. Las ontologías genéricas y de dominio son descriptas usando ontologías de representación.

- *Ontologías de más Alto Nivel (Top-Level):* Pretenden establecer una estructura básica, bajo la cual todos los términos de cualquier ontología existente, deberían poder relacionarse. Hasta ahora el principal problema es que no existe una ontología única de este tipo.

II.1.3.3. Usos y Aplicaciones de las Ontologías

II.1.3.3.1. El rol de las ontologías en los sistemas de información.

Según Barchini [BAR07] los sistemas de información (SI) son esencialmente artefactos de conocimiento que capturan y representan el conocimiento sobre ciertos dominios. Los profesionales e investigadores de los SI han tratado tradicionalmente con los problemas de identificar, capturar, y representar el conocimiento del dominio dentro de los SI.

Las ontologías generalmente se usan para especificar y comunicar el conocimiento del dominio de una manera genérica y son muy útiles para estructurar y definir el significado de los términos.

La forma de resolver dicho problema consiste en crear un entendimiento compartido, como son las ontologías, que unifican los diferentes puntos de vista y sirven para:

- Entender cómo diferentes sistemas comparten informaciones.
- Descubrir ciertas distorsiones presentes en los procesos cognitivos de aprendizaje en un mismo contexto.
- Formar patrones para el desarrollo de SI.

Es así como, el uso de ontologías en el desarrollo de los SI permite establecer correspondencia y relaciones entre los diferentes dominios de entidades de información.

Frank asegura que el uso de ontologías en el desarrollo de los SI contribuye a mejorar la calidad del producto final.

Es así como, las ontologías pueden proveer los mecanismos para organizar y almacenar ítems que incluyen esquemas de las bases de datos (BD), objetos de interfaz de usuario, y programas de la aplicación. Es decir, las ontologías están llegando a ser una herramienta fructífera en la investigación y desarrollo de la disciplina de los SI.

Esto ha llevado a la noción de SI basados en ontología (SIBO), un concepto que, aunque en una fase preliminar de desarrollo, abre nuevas maneras de pensar sobre las ontologías y los SI en conjunción una con otra, y cubre las dimensiones estructurales, las dimensiones temporales de los SI e involucra tanto a los desarrolladores como a los usuarios de los SI.
Las ontologías y los SIBO están desarrollándose y aplicándose en una variedad de áreas de aplicación emergentes tales como modelización de empresas, diagnósticos, toma de decisión, planeamiento y adaptación, modelado de procesos y sistemas.

II.1.3.3.2. Uso de ontologías y web semántica para apoyar la gestión del conocimiento

Según Sandoval Cantor [SAN07] es indispensable hacer uso de las ontologías para dar un carácter especializado a los contenidos temáticos, categorizando y catalogando la información a través de la generalización de áreas globales y especializadas.
Las ontologías incluyen definiciones de conceptos básicos relacionados con un dominio, así como las relaciones entre ellos, de tal forma que las computadoras pueden codificar el conocimiento y también el conocimiento extendido, haciendo reutilizable el conocimiento.
Integrando todos los aspectos anteriores, las organizaciones y entidades, cuentan con herramientas tecnológicas eficientes de comunicación y organización de información, de tal forma que el conocimiento organizacional podrá ser gestionado, categorizado, retroalimentado y ampliamente divulgado.
Para realizar la implementación de ontologías en los servicios web, se requiere hacer uso del lenguaje OWL (Ontology Web Language), el cual proporciona terminologías interpretables por la Web para la creación de estructuras ontológicas, brindando integración e interoperabilidad de datos descriptivos para el trabajo entre diversas comunidades.
OWL proporciona las siguientes capacidades a las ontologías:

- Capacidad de ser distribuidas a través de varios sistemas.
- Es escalable a las necesidades de la Web.
- Es compatible con los estándares Web de accesibilidad e internacionalización.
- Es abierto y extensible.
- Da utilidad de las Ontologías para la Web.

El Grupo de Trabajo de Ontologías para la Web del Profesor Nicola Guarino, identificó los principales casos de uso de ontologías en la Web y los describió en el documento de "Casos de Uso y Requisitos"; se realizó un estudio sobre veinticinco servicios

implementados con lenguajes de ontologías para Web poco avanzados, de donde se obtuvo la siguiente clasificación:

- Portales Web.
- Reglas de categorización utilizadas para mejorar la búsqueda.
- Colecciones multimedia.
- Búsquedas basadas en contenido para medios no textuales.
- Administración de Sitios Web Corporativos.
- Organización taxonómica automatizada de datos y documentos.
- Asignación entre Sectores Corporativos.
- Documentación de Diseño.
- Explicación de partes "derivadas" (Ej.: el tornillo de una pieza mecánica).
- Administración explícita de restricciones.
- Agentes Inteligentes.
- Expresión de las preferencias y/o intereses de los usuarios.
- Mapeo de contenidos entre sitios Web.
- Composición y descubrimiento de Servicios Web.
- Administración de derechos y control de acceso.

II.2. MARCO METODOLÓGICO

II.2.1. INTRODUCCIÓN

En este marco se incluyen las metodologías, técnicas y herramientas necesarias que darán soporte al diseño y construcción del prototipo, por lo que se incluyen temas relacionados con la ingeniería ontológica y entornos de desarrollo de interfaces de usuarios basados en modelos.

La ingeniería ontológica es un nuevo campo de la Informática que estudia los métodos y metodologías para construir ontologías. De la bibliografía consultada se han seleccionado tres metodologías que se describen en este marco, que servirán de guía para la construcción y reuso de las ontologías necesarias que empleará el prototipo.

Además, de las metodologías descriptas, es necesario contar con herramientas o entornos que permitan la creación, edición y reuso de ontología, y un motor de reglas. Para tal fin, se incluyen en este marco el entorno de desarrollo Protégé y el motor de reglas JessTab.

También, se describen dos ontologías que se reusarán en este trabajo, GUMO y UsiXML, que aportan conceptos, propiedades y relaciones que representan al modelo de usuario y al modelo de Interfaz de Usuario, respectivamente.

Por otra parte, en este marco se presentan los entornos de desarrollo de interfaces de usuarios basados en modelos que permiten generar automáticamente o semi-automáticamente IU, para lo cual es necesario comprender los modelos en que se basan y el proceso de diseño de la IU basada en modelos.

Por último, se describe el diseño centrado en el usuario como un modelo para el diseño de IU, mediante el cual los diseñadores pueden captar como se lleva a cabo la interactividad entre usuario y sistema técnico. La idea principal es que el usuario es el centro del diseño, en cualquier sistema computacional. Los usuarios, los diseñadores y el equipo técnico trabajan unidos con el objetivo de articular aquello que se desea, que se necesita y conocer las limitaciones del usuario para crear un sistema adecuado.[WIK09]

II.2.2. INGENIERÍA ONTOLÓGICA

La Ingeniería Ontológica se refiere al conjunto de actividades relacionadas con el proceso de desarrollo, el ciclo de vida, los métodos y metodologías para la construcción de ontologías, y las herramientas y lenguajes que dan soporte a las mismas.

Desde los años 90, se ha incrementado la atención puesta a la Ingeniería Ontológica. Ahora las ontologías se utilizan ampliamente en las áreas de Ingeniería de Conocimientos, Inteligencia Artificial e Informática en general, en aplicaciones relacionadas con la gestión de conocimientos, el procesamiento del lenguaje natural, el comercio electrónico, la integración inteligente de información, bio-informática, educación, etc., así como el elemento fundamental en el desarrollo de la Web Semántica. [GOM03]

II.2.2.1. Metodologías de desarrollo

II.2.2.1.1. Guía para crear ontologías

Mizoguchi [MIZ95] presenta una metodología simple para la ingeniería del conocimiento aplicada a la creación de ontologías. Esta metodología está compuesta por siete pasos:

1. **Determinar el dominio y el alcance o ámbito de la ontología**. Definir el dominio y el alcance de la ontología.
2. **Considerar la reutilización de ontologías existentes**. Comprobar si es posible usar y extender fuentes de conocimientos ya existentes, y que puedan ser de utilidad para el dominio del problema.

3. **Enumerar los términos importantes en la ontología**. Es útil escribir una lista con todos los términos con los que se harían afirmaciones acerca del dominio o se explicaría éste a un usuario. El contenido de la lista debe ser preciso y carente de ambigüedades.
4. **Definir las clases y la jerarquía de clases**. De la lista creada en el paso 3, se seleccionan aquellos términos independientes para constituir las clases. A partir de éstas se organiza la jerarquía.
5. **Definir las propiedades de las clases – (slots)**. Las clases por sí solas no ofrecen suficiente información para responder a las preguntas de competencia. Por tanto, se deben describir los conceptos propios de la estructura interna de las clases. Por lo general los términos que no fueron seleccionados en el paso 4 pasan a considerarse propiedades de las clases. Adicionalmente, todas las propiedades de la superclase son heredadas por sus subclases.
6. **Definir las características (facetas) de las propiedades**. Las ranuras tienen diferentes propiedades que describen el tipo de valor, los valores permitidos, el número de valores (cardinalidad), así como otras características de los valores que la propiedad puede tener.
7. **Crear instancias**. Es el último paso de este proceso. Definir una instancia individual de una clase requiere a) elegir una clase; b) crear una instancia individual de esa clase, y c) rellenar las propiedades con valores.

II.2.2.1.2. Enterprise

Es un esquema metodológico que constituye la base de muchos de los métodos propuestos y usados en la actualidad. El esquema está constituido por cuatro pasos y considera además un conjunto de guías o recomendaciones de diseño que se deben tener presentes en cada paso del método [GRU96, GRU94a, GRU94]:

1. **Identificar el propósito y el alcance de la ontología**. Se señala claramente el propósito para el que se desea construir la ontología así como el alcance de la misma.
2. **Construir la ontología.** Este paso considera tres aspectos necesarios para la construcción de la ontología, a saber:
    a. **Capturar el conocimiento:** Se identifican los conceptos claves y sus relaciones en el dominio.
    b. **Codificar el conocimiento:** Se representa en un lenguaje formal la conceptualización capturada en el paso anterior.

c. **Integrar el conocimiento:** Se examinan las ontologías existentes y se verifica si pueden ser integradas a la que se está construyendo.

3. **Evaluar.** Se hace un juicio técnico a la ontología considerando la conceptualización, el ambiente, el software y la documentación, con respecto a una referencia. Esta referencia puede ser: requisitos de especificación, preguntas de competencias y/o el mundo real.
4. **Documentar.** Se documenta adecuadamente el conocimiento expresado en la ontología, para así garantizar que sea apropiadamente compartido y reutilizado.

II.2.2.1.3. Desarrollo de Ontologías-101

A continuación se presenta una metodología creada por Noy y McGuinness [NOY05] para el desarrollo de ontologías.

Paso 1. Determinar el domino y alcance de la ontología

Se sugiere comenzar el desarrollo de una ontología definiendo su dominio y alcance. Es decir, responder a algunas preguntas básicas:

- ¿Cuál es el dominio que la ontología cubrirá?
- ¿Para qué se usará la ontología?
- ¿Para qué tipos de preguntas la información en la ontología deberá proveer respuestas?
- ¿Quién usará y mantendrá la ontología?

Las respuestas a esas preguntas pueden cambiar durante el proceso del diseño de la ontología, pero en cualquier momento dado ellas ayudarán a limitar el alcance del modelo.

Paso 2. Considerar la reutilización de ontologías existentes

Casi siempre vale la pena considerar lo que otra persona ha hecho y verificar si se puede refinar y extender recursos existentes para dominio y tarea particular que se está trabajando. Reusar ontologías existentes puede ser un requerimiento si el sistema necesita interactuar con otras aplicaciones que ya se han dedicado a ontologías particulares o vocabularios controlados. Muchas ontologías ya están disponibles en forma electrónica y pueden ser importadas dentro un entorno de desarrollo de ontologías que se está usando. El formalismo en el cual está expresada una ontología a menudo no interesa, puesto que muchos sistemas de representación de conocimiento pueden importar y exportar ontologías. Aun si el sistema de representación de conocimiento no puede funcionar directamente con un formalismo particular, la tarea de traducir una ontología a partir de un formalismo a otro no es usualmente difícil.

Paso 3. Enumerar términos importantes para la ontología

Es útil escribir una lista con todos los términos con los que se quisiera hacer enunciados o dar explicación a un usuario. ¿Cuáles son los términos de los cuales se quisiera hablar? ¿Qué propiedades tienen esos términos? Inicialmente, es importante obtener una lista integral de términos sin preocuparse del recubrimiento entre los conceptos que representan, relaciones entre los términos, o cualquier propiedad que los conceptos puedan tener, o si los conceptos son clases o slots. Los siguientes dos pasos (desarrollar la jerarquía de clases y definir las propiedades de los conceptos (slots)) están estrechamente relacionadas. Es difícil hacer primero uno de ellos y luego hacer el otro. Típicamente, se crean definiciones de los conceptos en la jerarquía y luego se continúan describiendo las propiedades de esos conceptos y así sucesivamente. Esos dos pasos son también los más importantes en el proceso de diseño de la ontología. Se los describirá brevemente y se les dedicará lospasos 4 y 5en los que se discutirán asuntos más complicados que necesitan ser considerados, peligros comunes, decisiones a tomar, etc.

Paso 4. Definir las clases y la jerarquía de clases

Hay varios posibles enfoques para desarrollar una jerarquía de clases:

- Un proceso de desarrollo top-down comienza con la definición de los conceptos más generales en el dominio y la subsecuente especialización de los conceptos.
- Un proceso de desarrollo bottom-up comienza con la definición de las clases más específicas, las hojas de la jerarquía, con el subsecuente agrupamiento de esas clases en conceptos más generales.
- Un proceso de desarrollo combinado es el resultado de una combinación de los enfoques top-down y bottom-up: primero se definen los conceptos más sobresalientes y luego se los generaliza y especializa apropiadamente.

Paso 5. Definir las propiedades de las clases: slots

Las clases aisladas no proveerán suficiente información para responder las preguntas de competencia del Paso 1. Una vez que se hayan definido algunas de las clases, se debe describir la estructura interna de los conceptos.

Ya se han seleccionado clases de la lista de términos creada en el Paso 3. La mayoría de los términos restantes son muy probablemente propiedades de esas clases.

Para cada propiedad en la lista, se debe determinar qué clase es descrita por la propiedad. Esas propiedades se convierten en slots adosados a las clases.

En general, hay varios tipos de propiedades de objeto que pueden llegar a ser slots en una ontología:

- propiedades "intrínsecas";
- propiedades "extrínsecas";
- partes, si el objeto es estructurado; pueden ser "partes" físicas y abstractas;
- relaciones con otros individuos; éstas son las relaciones entre miembros individuales de una clase y otros ítems.

Paso 6. Definir las facetas de los slots

Los slots puedes tener diferentes facetas que describen el tipo de valor, valores admitidos, el número de los valores (cardinalidad), y otras características de los valores que los slots pueden tomar.

Cardinalidad del slot

La cardinalidad de un slot define cuantos valores puede tener este. Algunos sistemas solamente distinguen entre cardinalidad simple (admitiendo a lo sumo un valor) y cardinalidad múltiple (admitiendo cualquier cantidad de valores). Algunos sistemas admiten la especificación de una cardinalidad mínima y máxima para describir la cantidad de valores de un slot con más precisión. Una cardinalidad mínima N significa que un slot debe tener al menos N valores. Una cardinalidad máxima M significa que un slot puede tener a lo sumo M valores.

Tipo de valor de los slots

Una faceta tipo de valor describe qué tipos de valores pueden llenar el slot. Aquí está una lista de los tipos de valores más comunes:

- **String** es el tipo de valor más simple, por ejemplo es usado por slots como *nombre*: el valor es una simple cadena de caracteres.
- **Number** algunas veces los tipos de valores Float e Integer son usados por ser más específicos, describe slots con valores numéricos.
- Los slots del tipo **Boolean** son simples banderas si/no.
- Los slots del tipo **Enumerated** especifican una lista de valores admitidos para el slot. En Protégé-2000 los slots enumerados son del tipo Symbol.
- Los slots del tipo **Instance** admiten la definición de relaciones entre individuos. Los slots con tipo de valor Instance deben también definir una lista de clases admitidas de las cuales pueden provenirlas instancias.

Dominio y rango de un slot

Las clases admitidas para los slots de tipo Instance son a menudo llamadas rango de un slot. Algunos sistemas permiten restringir el rango de un slot cuando el slot esta adosado a una clase particular.

Las clases a las cuales un slot está adosado o las clases cuyas propiedades son descritas por un slot son llamadas dominio del slot. En los sistemas en los cuales se adosan slots a las clases, las clases a las cuales el slot es adosado usualmente constituye el dominio del slot. No hay necesidad de especificar el dominio separadamente.

Paso 7. Crear instancias

El último paso consiste en crear instancias individuales de clases en la jerarquía. La definición de una instancia individual de una clase requiere (1) elegir una clase, (2) crear una instancia individual de la clase y (3) rellenar los valores del slot.

II.2.2.2. Herramientas

II.2.2.2.1. Protégé

Protégé [PRO09] es un entorno independiente de plataforma extensible, para la creación y edición de ontologías y bases de conocimiento. Es una plataforma gratuita y de código abierto que provee un crecimiento a la comunidad de usuarios con un conjunto de herramientas para construir modelos de dominio y aplicaciones basadas en conocimiento con ontologías. En su núcleo Protégé implementa un buen conjunto de estructuras y acciones de modelización de conocimiento que soporta la creación, visualización y manipulación de ontologías en varios formatos de representación. Protégé puede ser personalizado para proveer soporte de dominio amigable para la creación de modelos de conocimiento e ingreso de datos. Además, puede ser extendido mediante plug-ins y APIs realizados con JAVA para la construcción de herramientas y aplicaciones basadas en conocimiento.

La plataforma soporta dos formas principales de modelar ontologías:

- El editor de Frames Protégé permite a los usuarios construir y poblar ontologías que son basadas en frames, de acuerdo al protocolo Open Knowledge Base Connectivity (OKBC). En este modelo, una ontología consta de un conjunto de clases organizadas en una jerarquía de clasificación para representar conceptos principales de un dominio, un conjunto de propiedades asociadas a clases para describir sus propiedades y relaciones, y un conjunto de instancias de tales clases, o sea,

ejemplares individuales de los conceptos que tienen valores específicos para sus propiedades.

- El editor de OWL Protégé permite a los usuarios construir ontologías para la Web Semántica, en particular en el Lenguaje de Ontologías Web (OWL) de W3C. Una ontología de OWL puede incluir descripciones de clases, propiedades y sus instancias. Dada tal ontología, la semántica formal OWL especifica como derivar sus consecuencias lógicas, por ejemplo hechos literalmente ausentes en la ontología, pero implicados por la semántica. Estas implicaciones pueden basarse en un solo documento o en múltiples documentos distribuidos que han sido combinados usando mecanismos OWL definidos.

Hay muchas características que distinguen Protégé de otras herramientas de edición de bases de conocimiento.

Protégé tiene las siguientes características:

- Interfaz de usuario gráfica intuitiva y fácil de usar.
- Escalabilidad: la base de datos de back-end de Protégé carga frames sólo cuando hay demanda y usa la cache para liberar memoria cuando es necesario. Prácticamente no hay ningún deterioro en el rendimiento cuando se va de cientos de frames a miles de frames. La comunidad de usuarios de Protégé ha llegado a crear bases de conocimiento de 150000 frames.
- Arquitectura de plug-ins extensible: Se puede ampliar fácilmente Protégé con plug-ins adaptados a sus dominios y tareas. Algunos plug-ins son:
  - Pequeños componentes de interfaz de usuario que son adecuados para mostrar y obtener valores de su dominio. Tales componentes podrían ser usados en los formularios de Protégé.
  - Plug-ins de back-end personalizables que usan mecanismos de almacenamiento propios.
  - Nuevas aplicaciones estrechamente relacionadas a una base de conocimientos, como una pestaña más de Protégé.

II.2.2.2.2. Jess

Es un motor de reglas y ambiente de script escrito en Java. Utiliza una versión mejorada del algoritmo Rete para procesar las reglas. Posee diferentes características únicas incluyendo encadenamiento hacia atrás y consultas en memoria de trabajo [SAN08].

JessTab

JessTab [HEN06] es un plug-in para Protégé que permite usar Jess y Protégé en forma conjunta. Brinda lo mejor de los dos. JessTab provee una consola de Jess para Windows en donde se puede interactuar con Jess mientras se ejecuta Protégé. Más aun, JessTab extiende Jess con funciones adicionales que permiten mapear las bases de conocimiento de Protégé con hechos (facts) de Jess. También, existen funciones para manipular bases de conocimiento de Protégé desde Jess.

Mediante el uso de JessTab, se pueden desarrollar programas que manejan ontologías de Protégé directamente, y se puede escribir reglas que se disparan de acuerdo a los patrones de la base de conocimientos. Es posible utilizar JessTab como una extensión a Jess orientada a objetos mediante la definición de clases e instanciación de clases en Protégé. Ya que, Protégé posee metaclases explícitas que se pueden mapear a hechos y también realizar mapeo de patrones con propiedades de las clases. En resumen, se pueden manipular las ontologías de Protégé (por ejemplo, instanciando clases y cambiando los valores de los slots).

II.2.2.3. Ontologías existentes

II.2.2.3.1. GUMO – the General UserModelOntology

GUMO [HEC05] pretende ser una ontología de alto nivel aceptada para representar los modelos de usuario. Esta ontología está representada en un lenguaje web semántico moderno como OWL, y está disponible a través de internet para cualquier sistema adaptativo. La mayor ventaja si se la utiliza de esta forma es el intercambio de datos del modelo de usuario entre los distintos sistemas adaptativos.

El problema actual de las diferencias estructurales y sintácticas entre los sistemas de modelado de usuario podría superarse con una ontología comúnmente aceptada, especializada para las tareas de modelado de usuario.

GUMO es una ontología de modelo de usuario en vez de una ontología de modelado de usuario que, además, incluye técnicas de inferencia o de conocimiento sobre el área de investigación en general.

Se recogen dimensiones del usuario que son modeladas dentro de sistemas adaptativos (adaptables al usuario) como el ritmo cardíaco del usuario, la edad, la posición actual del usuario, el lugar de nacimiento o la capacidad del usuario para nadar. Además se analizan, el modelado de los intereses del usuario y sus preferencias, como la lectura de poemas, juegos de aventura o beber ciertos vinos franceses de Burdeos.

La forma de relacionar a los usuarios con sus preferencias en esta ontología es a través de la división de las dimensiones del modelo de usuarios en tres partes: auxiliares, predicado, y rango (*auxiliary, predicate, range*). Se pueden observar algunos auxiliares y predicados en la imagen II.1.

Una declaración situacional (*situationalstatements*) forma una estructura uniforme y central para representar cualquier información de situación.

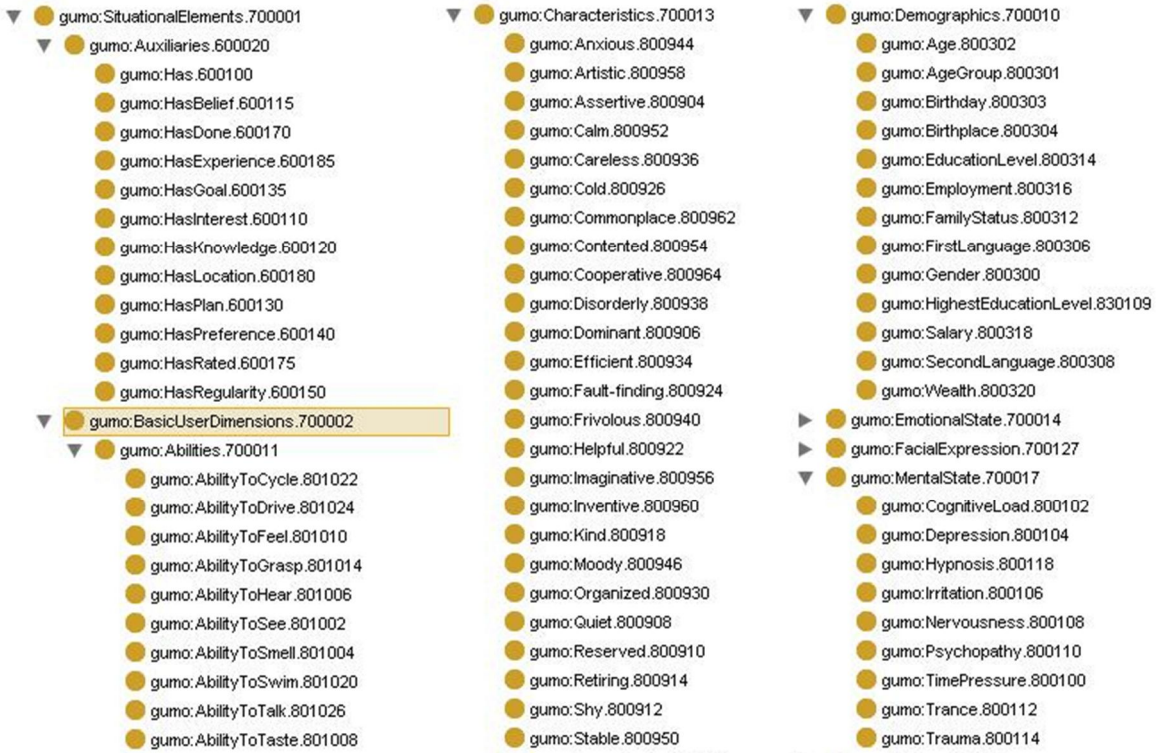


Imagen II.1. Los Auxiliares (Auxiliaries) y algunas de las Dimensiones Básicas del Usuario (BasicUserDimensions), Habilidades (Abillities), Características (Characteristics), Características Demográficas (Demographics) y Estado Mental (MentalState)

Debido al alcance de este trabajo se utilizó solamente el conjunto de atributos mainpart. En la tabla II.1, se detallan dichos atributos, y se ejemplifica para "Pedro tiene 11 años de edad".

Tabla II.1. Atributos de la capa Mainpart.

Mainpart	**Quíntupla de atributos de declaración situacional**	**Ejemplo**
Subject	La entidad principal a la cual se refiere la declaración (la mayoría de las veces el usuario).	"Pedro"
Auxiliary	Parte del predicado como "hasProperty" or "hasInterest".	"tiene"
predicate	La dimensión o categoría del usuario como "Edad" o "Natación".	"edad"
Range	El rango del objeto atributo, es posible que tome valores por defecto.	"11"
Object	El valor de la declaración de la terna subject-auxiliary-predicate.	"años"

Si se quiere decir algo sobre el interés del usuario en el fútbol, se podría dividir en *auxiliary*=**tieneInteres (hasInterest)**, *predicate*=**futbol** y *range*=**bajo-medio-alto**. Si uno quiere expresar algo así como el conocimiento de las sinfonías, se podría dividir en *auxiliary*=**tieneConocimiento**, *predicate*=**sinfonías** y *range*=**pobre-promedio-bueno-excelente**. Hasta el momento han sido identificados e insertados en la ontología aproximadamente 1000 grupos de auxiliares, predicados y rangos.

En este trabajo se utilizará GUMO como estructura conceptual para modelar las características o preferencias de los usuarios que son la base para la personalización de las interfaces de usuario.

II.2.1.5.1. UsiXML

II.2.1.5.1.1. Definición

UsiXML [QUE04] (USer Interface eXtensible Markup Language) es un UIDL compatible con XML que describe una IU para múltiples contextos de uso tales como Interfaces de Usuario basadas en Caracteres (CUI), Interfaces de Usuario Gráficas (GUI), Interfaces de Usuario de Auditoría e Interfaces de Usuario Multimodales. En otras palabras, las aplicaciones interactivas con diferentes tipos de técnicas de interacción, modalidades de uso y plataformas pueden describirse de una forma que preserve el diseño independiente de características peculiares de las plataformas físicas.

UsiXML tiene las siguientes características:

- No está destinado a desarrolladores, sino a analistas, especificadores, diseñadores, expertos en factores humanos, líderes de proyectos, programadores novatos, etc. Pero, además UsiXML puede ser utilizado por desarrolladores experimentados. Gracias a UsiXML, los que no son desarrolladores pueden formar la IU de cualquier aplicación interactiva nueva a través de la especificación o descripción con UsiXML, sin necesidad de tener habilidades para programar como en los lenguajes de marcado (por ej. HTML) y lenguajes de programación (por ej. Java o C++).
- Consta de un UIDL, que es un lenguaje declarativo que captura la esencia de que una IU es, o debería ser, independiente de las características físicas.
- Describe los elementos que componen la IU a un alto nivel de abstracción: widgets, controles, contenedores, modalidades, técnicas de interacción, etc.
- Permite el desarrollo de aplicaciones interactivas en cualquier herramienta. Una IU de cualquier aplicación compatible con UsiXML se ejecuta en todas las herramientas que lo implementen: compiladores e intérpretes.

- Soporta independencia del dispositivo: una IU puede describirse de una forma para que siga siendo autónoma con respecto a los dispositivos usados en la interacción, tales como, mouse, pantalla, teclado, sistema de reconocimiento de voz, etc. En caso de necesidad, se puede incorporar una referencia a un dispositivo en particular.
- Soporta independencia de plataforma: una IU puede describirse de forma que siga siendo autónoma con respecto a las distintas plataformas, como, teléfonos móviles, Pocket PC, Tablet PC, PC portátiles, de escritorio, etc. En caso de necesidad, se puede incorporar una referencia a una plataforma en particular.
- Soporta independencia de modalidad: una IU puede describirse de forma que siga siendo independiente de cualquier modalidad de interacción, como interacción gráfica, vocal, 3D o táctil. En caso de necesidad, se puede incorporar una referencia a una modalidad particular.
- Permite el reuso de elementos previamente descriptos en IUs anteriores para componer una IU en aplicaciones nuevas.

Como se observa en la figura II.6 el modelo de IU es la superclase más alta y contiene características comunes compartidas por todos los modelos que componen una interfaz de usuario. Un modelo de IU puede estar compuesto por una lista de modelos en cualquier orden y cualquier número. Como ser, el modelo de tareas (task model), de dominio (domain model), de interfaz de usuario abstracta (abstract user interface model), de interfaz de usuario concreta (concrete user interface model), de asignación (mapping model), de contexto (context model), de recursos (resource model), y de transformación (transformation model). El modelo de interfaz de usuario es el núcleo de todos los modelos UsiXML. Un modelo de interfaz de usuario no necesita incluir a cada uno de los modelos. Por otra parte, puede haber más de un tipo particular de modelo.

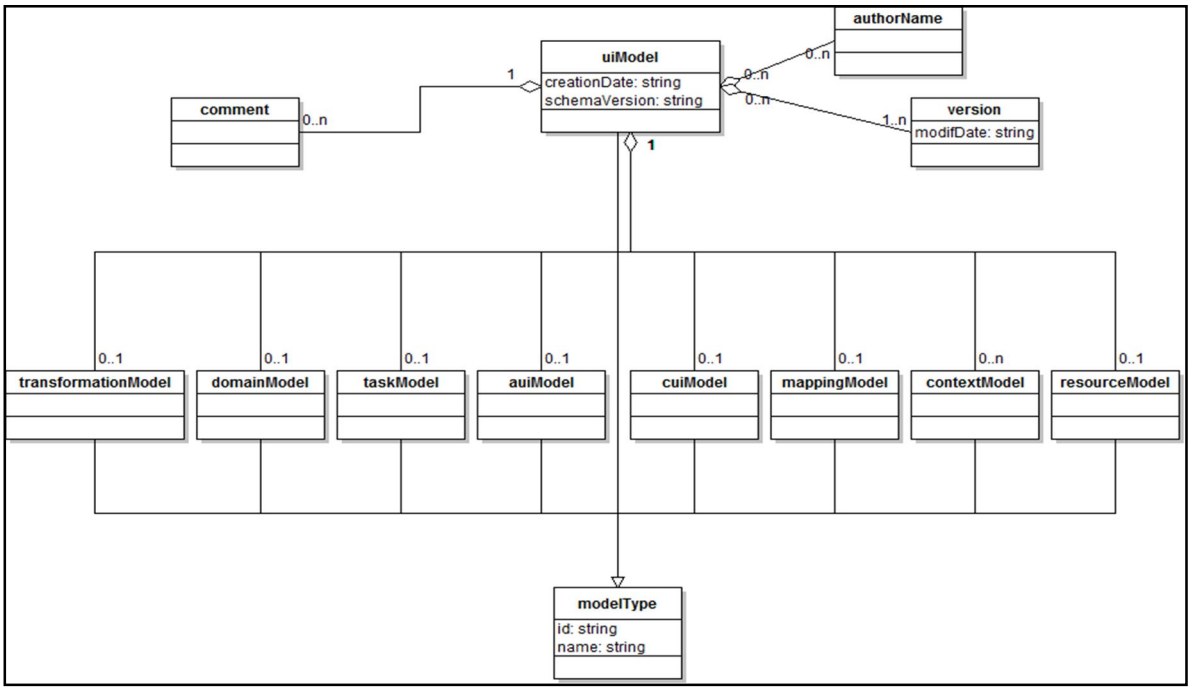


Figura II.6. Modelos de UsiXML [USI07].

Elementos Concretos

A continuación se detallan los elementos concretos y sus propiedades, estos serán usados más adelante en este trabajo para producir las interfaces concretas.

Elemento *2DgraphicalIndividualComponent*

Es un elemento de tipo *GraphicalCio* contenido en un *graphicalContainer*.

isUnderlined	(Boolean) Especifica si un *textComponent* esta subrayado o no.
textSize	(integer) Especifica el tamaño de un *textComponent*

Elemento window

Es un área rectangular cerrada en una pantalla, hereda propiedades de *2DgraphicalContainer*.

windowLeftMargin	(integer) Indica el tamaño del margen izquierdo en pixeles.
windowTopMargin	(integer) Indica el tamaño del margen superior en pixeles
isResizable	(Boolean) Especifica si se puede modificar el tamaño de una ventana o no. Por defecto: true.

Elemento button

Su objetivo es disparar cualquier tipo de acción disponible en el sistema. Se utilizan las propiedades de los elementos *CIO* y *2DgraphicalIndividualComponent*

Elemento OutputText

Es un componente individual gráfico especializado para manejar la salida de contenido textual. Se agregan a las propiedades siguientes las propiedades de los elementos *CIO* y *2DgraphicalIndividualComponent*.

defaultHyperLinkTarget	(uri) Es el hipervínculo por defecto.
hyperLinkTarget	(uri) Designa hacia donde apunta el hipervínculo.
visitedLinkColor	(string) Es el color del hipervínculo después de haber sido visitado una vez.
activeLinkColor	(string) Es el color de un *outputText* de tipo hipervínculo cuando se hace clic en el.
textMargin	(integer) Especifica el tamaño del *textmargin* en pixeles.
textVerticalAlign	(string) Indica una restricción de alineación vertical del texto contenido en un *outputText*. Los valores permitidos son: top, middle, bottom.
textHorizontalAlign	(string) Indica una restricción de alineación horizontal del texto contenido en un *outputText*. Los valores permitidos: left, middle, right.
numberOfColumns	(integer) Es el número de columnas de un outputText.
numberOfLines	(integer) Es el número de filas de un outputText.

Elemento listBox

Permite una selección sobre una lista desplegable de selección simple o múltiple. Hereda propiedades de *2DgraphicalIndividualComponent*.

maxlineVisible	(integer) Indica el número de líneas visibles.
isEditable	(Boolean) Especifica si el contenido de un listbox es editable o no.
multipleSelection	(Boolean) Si es verdadero entonces se pueden seleccionar varios ítems, si es falso, solamente un ítem puede ser seleccionado cada vez.

Elemento inputText

Es un componente individual gráfico especializado para manejar el ingreso de contenido textual. Hereda las propiedades de *2DgraphicalIndividualComponent*.

textMargin	(integer) Especifica el tamaño de *textmargin* en pixeles.
isEditable	(Boolean) Especifica si un *inputText* es editable o no.
isWordWrapped	(Boolean) Indica si el texto del*inputText*se adapta a la forma o no.
forceWordWrapped	(Boolean) Indica si el wrapping (similar a justificado) de un inputText es forzado o no. Si es verdadero las palabras pueden verse separadas. Este atributo posee un valor solamente en el caso de que *isWordWrapped*sea verdadero.
maxLength	(integer) Es la longitud máxima de un contenido de un *inputText*. Expresada en número de caracteres.
numberOfColumns	(integer) Es el número de columnas de un *inputText*.
numberOfLines	(integer) Es el número de líneas de un *textComponent*.
textVerticalAlign	(string) Indica una alineación vertical del texto contenido en un *inputText*. Valores permitidos: top, middle, bottom.
textHorizontalAlign	(string) Indica una restricción de alineación horizontal del texto contenido en un *inputText*. Valores permitidos: left, middle, right.
filter	(uri) Es una expresión regular que restringe el contenido de un *inputText*.
defaultFilter	(string) Es el filtro por defecto.
isPassword	(Boolean) Especifica si el *inputText* es de tipo password o no. Si es verdadero el input del usuario estará oculto.

Elemento vocalOutput

Hereda de: *vocalIndividualComponent*

Es un objeto utilizado para transformar en audio la información del sistema que se visualice por pantalla para el usuario. La información es especificada en el atributo *defaultContent* heredado de la clase *CIO*.

Volume	(integer) Es el volumen del sonido expresado en Db (decibeles).
Intonation	(string) Expresa el tono dominante de en el que será sintetizado el *vocalOutput*, este puede ser: positivo, negativo, interrogativo o exclamativo.
isInterruptible	(Boolean) Especifica si el *vocalPrompt* puede ser interrumpido por una expresión del usuario.

Elemento imageComponent

Hereda de: *2DgraphicalIndividualComponent*. Es un *graphicalIndividualComponent* especializado para manejar contenido de imágenes.

imageHeight	(integer) Es la altura de un *ImageComponent* expresada en pixeles.
imageWidth	(integer) Es el ancho de un *ImageComponent* expresada en pixeles.
imageHorizSpace	(integer) Expresa un offset horizontal (en pixeles) con respecto al contenedor de un *imageComponent*.
imageBorder	(integer) Expresa el ancho de un borde en pixeles
hyperLinkTarget	(uri) Especifica el archivo objetivo alcanzable desde un *imageComponent* (depende del contexto).
defaultHyperLinkTarget	(uri) Es el hipervínculo por defecto.

En la figura II.7 se observa la sintaxis del lenguaje usiXML que se define a través de conjuntos de esquemas XML. Cada esquema corresponde a cada uno de los modelos.
En este caso se trata de un ejemplo de interfaz de usuario concreta que muestra un mensaje de “HelloWorld”.

```
<?xmlversion="1.0"encoding="UTF-8"?>
<uiModelxmlns="http://www.usixml.org"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.usixml.org/
  http://www.usixml.org/spec/UsiXML-ui_model.xsd"
id="hello_world_12"name="hello world"
creationDate="2007-02-12T14:31:45.143+01:00"schemaVersion="1.8.0">
<head>
<versionmodifDate="2007-02-12T14:31:45.143+01:00">1</version>
<authorName>Benjamin Michotte</authorName>
<comment>Hello World example</comment>
</head>
<cuiModelid="hello_world-cui_12"name="hello world-cui">
<windowid="window_component_0"name="window_component_0"
content="/uiModel/resourceModel/
cioRef[@cioId='window_component_0']/resource/@content"
defaultContent="Hello world"width="470"height="255">

<gridBagBoxid="grid_bag_box_1"name="grid_bag_box_1"
gridHeight="12"gridWidth="23"/>
</window>
</cuiModel>
<contextModelid="hello_world-contextModel_12"
name="hello world-contextModel">
<contextid="hello_world-context-en_US_12"
name="hello world-context-en_US">
<userStereotypeid="hello_world-sten_US_12"language="en_US"
stereotypeName="hello world-sten_US"/>
<platformid="hello_world-platform_12"
name="hello world-platform"/>
<environmentid="hello_world-env_12"name="hello world-env"/>
</context>
<contextid="hello_world-context-fr_FR_12"
name="hello world-context-fr_FR">
<userStereotypeid="hello_world-stfr_FR_12"language="fr_FR"
stereotypeName="hello world-stfr_FR"/>
<platformid="hello_world-platform_12"
name="hello world-platform"/>
<environmentid="hello_world-env_12"name="hello world-env"/>
</context>
</contextModel>
<resourceModelid="hello_world-res_12"name="hello world-res">
<cioRefcioId="window_component_0">
<resourcecontent="Bonjour le monde"
contextId="hello_world-context-fr_FR_12"/>
<resourcecontent="Hello world"
contextId="hello_world-context-en_US_12"/>
</cioRef>
</resourceModel>
</uiModel>
```

Reseña: Modelo – Submodelo – Atributos – Agregados

Figura II.7. Ejemplo de la sintaxis UsiXML

II.2.3. ENTORNOS DE DESARROLLO DE INTERFACES DE USUARIOS BASADOS EN MODELOS

A continuación se presentan los entornos de desarrollo de interfaces de usuarios basados en modelos (MB-UIDE). Para su comprensión es importante conocer los modelos teóricos y metodologías en los que se basan. También se tratará una metodología (Diseño Centrado en el Usuario) empleada en el diseño de IU cuando se emplean este tipo de sistemas.

II.2.3.1. Definición

Son suites de software que soportan el diseño y desarrollo de IU a través de la creación de diversos modelos.

Las herramientas del entorno están comúnmente agrupadas en herramientas de tiempo de diseño, herramientas de tiempo de ejecución y sistemas de tiempo de ejecución. Esta caracterizado por la conectividad de los modelos y componentes dentro de él.

La mayoría de los problemas de desarrollo de interfaces provienen de: 1) la necesidad de ambientes de diseño centrado en el usuario (DCU) y 2) la falta de sistemas de software que soporten la mayoría de los niveles de desarrollo.

Un *modelo de interfaz* es una descripción declarativa de los aspectos que involucra la IU de un sistema, incluyendo cada componente y el diseño de esa interfaz. Un modelo, por tanto, conlleva la consideración de diferentes elementos a distintos niveles de abstracción: tareas de usuarios, elementos de dominio, presentaciones, diálogos, tipos de usuarios y relaciones de diseño. Estos están comúnmente organizados en modelos (por ejemplo de tareas, de usuarios o modelos de presentación) dentro del modelo de interfaz. Los modelos de interfaz están expresados mediante un *lenguaje de descripción de interfaz de usuario (UIDL)* [PUE97].

II.2.3.2. Ventajas

Los MB-UIDE tratan de ofrecer un entorno en el que los desarrolladores puedan diseñar e implementar IU de un modo profesional y sistemático, de forma más sencilla que cuando se usan herramientas tradicionales de desarrollo. Para lograr este objetivo, las IU se describen usando modelos declarativos. Hay tres ventajas principales derivadas del uso de Modelos Declarativos de IU (MDIU).

Pueden ofrecer descripciones más abstractas del IU que las que se pueden alcanzar con otras herramientas de desarrollo.

Facilitan la creación de métodos para diseñar e implementar la IU de una manera sistemática puesto que ofrecen capacidades para:

1. Modelar IU usando diferentes niveles de abstracción.
2. Refinar los modelos de forma incremental.
3. Reutilizar las especificaciones de la IU.

Ofrecen la infraestructura necesaria para la automatización de las tareas relacionadas con los procesos de diseño e implementación de IU [DEL07].

II.2.3.3. Modelos

En un principio los modelos no tenían su equivalente computacional. Eran usados para definir los componentes y sus funcionalidades, guiar el diseño de interfaces y los entornos de desarrollo de interfaces. En la actualidad los diferentes MB-UIDE utilizan distintos tipos de modelos y con denominaciones heterogéneas. A continuación se enumeran los más comunes.

II.2.3.3.1. Modelo de contexto de uso

Es un modelo que describe tres aspectos de un contexto de uso en el cual un usuario final realiza una tarea interactiva con una plataforma de computación específica en un ambiente dado. Consecuentemente, un modelo de contexto consta de un modelo de usuario, un modelo de plataforma y un modelo de ambiente. Cada una de estas tres facetas es un modelo en sí.

- **Modelo de usuario**: El modelo de usuario recoge las características de los usuarios finales. Un modelo de usuarios describe a los futuros usuarios del sistema según sus habilidades, su conocimiento de la aplicación o la información que deben procesar.
- **Modelo de plataforma**: Recoge toda la información respecto a la plataforma computacional en donde correrá la interfaz. Por ejemplo, información acerca del código fuente, configuraciones de archivos, instalaciones, etc.
- **Modelo de ambiente**: Se refiere al ambiente en el cual el usuario se encuentra trabajando. Por ejemplo, en su hogar, en un ambiente público, etc.

II.2.3.3.2. Modelo de dominio

Es una descripción de las clases de objetos manipulados por el usuario mientras interactúa con el sistema. Recoge información de los objetos que manipulará el usuario, con sus características y el comportamiento que ofrecerán los mismos.

II.2.3.3.3. Modelo de tareas

Describe las tareas que los usuarios pueden realizar en la aplicación así como la relación entre ellas. El modelo de tareas, describe las tareas que el usuario puede realizar incluyendo objetos de dominio, subtareas, sus objetivos, y los procedimientos usados para alcanzar los objetivos. Los objetivos especifican cuando un estado deseado es alcanzado, secuencias de acciones que definen procedimientos para alcanzar un objetivo, y los objetos de dominio que representan los elementos que deben mostrarse en la interfaz para completar cada tarea en el modelo.

II.2.3.3.4. Modelo de diálogo

El modelo de diálogo describe la conversación hombre-computadora. Especifica cuándo el usuario final puede invocar funciones a través de diferentes mecanismos de triggering (botones, comandos, etc.) y medios de interacción (entrada de voz, pantallas sensibles al tacto, etc.).Cuándo el usuario final puede seleccionar o especificar entradas, y cuándo la computadora puede presentar la información al usuario final. Muchos modelos de diálogo mostraron evidencia de éxito, sin embargo, no se ha llegado a ningún consenso final sobre una técnica de modelado de diálogo.

II.2.3.3.5. Modelo de presentación

El modelo de presentación especifica cómo los objetos de interacción (o widgets) aparecen en los diferentes estados de diálogo. Generalmente está compuesta de una descomposición jerárquica de las posibles muestras de pantallas en grupos de objetos de interacción.

Por definición, los modelos de presentación y diálogo están muy interrelacionados, es por esto que algunos ambientes de desarrollo basados en modelos los consideran juntos.

El modelo de presentación describe los aspectos visuales de la interfaz. Se suele dividir en dos sub-modelos: el modelo de presentación abstracto (MPA) y el modelo de presentación concreto (MPC).

- El MPA provee una vista abstracta de una interfaz que es independiente del modelo concreto subyacente.
- El MPC es la instancia concreta de una interfaz que puede ser presentada a un usuario. Puede haber muchas instancias concretas de un modelo de presentación abstracto [DEL07].

II.2.3.4. Niveles de abstracción

Como generalmente sucede, la solución a un problema complejo, tal como el diseño de un sistema interactivo, se puede basar en un pequeño conjunto de conceptos básicos y claros. Para poder solucionar tales problemas es importante considerar los diferentes puntos de vista que se pueden tener en un sistema interactivo. Tales puntos de vista difieren de los niveles de abstracción (hasta que nivel de detalle se considera) y el foco (ya sea que se considere la tarea o la interfaz de usuario). La comunidad de desarrollo basado en modelos viene discutiendo desde hace tiempo tales puntos de vista.

Los niveles de abstracción son:

- *Modelo de tareas y dominio,* en este nivel, se consideran las actividades lógicas que necesitan ser realizadas para alcanzar los objetivos de los usuarios. Generalmente son representadas en forma jerárquica junto con indicaciones de las relaciones temporales entre ellas y sus atributos asociados. Se identifican también los objetos del dominio que necesitan ser manipulados para poder realizar las tareas.
- *Interfaz de usuario abstracta,* en este caso el foco se traslada al rendimiento de la tarea que soporta la interfaz de usuario. Solamente se considera la estructura lógica, es decir independiente de la implementación, por lo tanto evita detalles de bajo nivel. Los objetos de interés se describen en términos de su semántica a través de un objeto interactor. Por lo tanto es posible indicar, por ejemplo, que en un punto determinado existe la necesidad de un objeto de selección, sin indicar si la selección es realizada gráfica, vocal, a través de un gesto o alguna otra modalidad.
- *Interfaz de usuario concreta,* en esta instancia de modelado cada objeto de interacción abstracto es reemplazado con un objeto de interacción concreto que depende del tipo de plataforma y medio disponible y posee un número de atributos que definen más concretamente como debería ser percibida por el usuario.
- *Interfaz de usuario final,* en este nivel la interfaz concreta es traducida a una interfaz definida por un ambiente de software específico (por ejemplo: XHTML, Java, etc.) [MOD09].

II.2.3.5. Lenguaje de descripción de interfaz de usuario

A principios de la década del 80, los Sistemas de Administración de IU (UIMS) eran la raíz del desarrollo de IU basado en modelos. Los UIMS son un concepto importante para definir abstracciones de alto nivel, superiores a los conceptos de bajo nivel. Como resultado de ello, los mecanismos de bajo nivel y los detalles de la implementación podrían

ser extraídos innecesariamente. Los UIMS permiten a los desarrolladores de IU escribir especificaciones con lenguajes de especificación de alto nivel. Posteriormente, a finales de los 90, surgieron nuevas clases de dispositivos para acceder a servicios en la Web. Debido a esta diversidad de dispositivos, la creación de IU basada en modelos ha ganado importancia y permitió a los diseñadores especificar los diferentes aspectos de la IU en forma separada. XML apareció como una opción natural para capturar la especificación de esta gran variedad de IU. Nació una nueva familia de UIDL, que buscaban alcanzar los siguientes objetivos:

- Capturar los requisitos de IU para una definición abstracta que se mantiene estable en los diferentes contextos de desarrollo.
- Hacer un único diseño de IU para múltiples dispositivos, modalidades, plataformas y aparatos.
- Mejorar la reusabilidad del diseño de IU.
- Soportar la evolución, extensibilidad y adaptabilidad de la IU.
- Usar una descripción de IU que permita la generación automática de código de IU.

Las descripciones utilizadas se denominan *lenguaje de marcado o lenguaje de marcas* que es una forma de codificar un documento que, junto con el texto, incorpora etiquetas o marcas que contienen información adicional acerca de la estructura del texto o su presentación.

Se pueden encontrar dos tipos de UIDL:

- Por un lado están aquellos destinados a cubrir sólo la descripción de IU.
- Por otro, aquellos desarrollados para dar soporte al desarrollo basado en modelos y, que por lo tanto, no se limitan a describir la interfaz sino que facilitan mecanismos para describir, dependiendo de los casos, al usuario, sus tareas, su dominio, etc.

Continuando con la definición del UIDL se puede decir que la sintaxis se ocupa exclusivamente de la forma y estructura de los símbolos de un lenguaje, sin ninguna consideración dada a su significado.

La sintaxis abstracta se define como la estructura oculta de un lenguaje, su entorno matemático. La sintaxis concreta es la apariencia externa; la sintaxis visual se compone de cajas y flechas, es la representación clásica de una estructura gráfica. La sintaxis textual se describe, por ejemplo, utilizando un lenguaje basado en XML.

El objetivo de la estilística es ofrecer una representación de un conjunto de objetos definidos para facilitar su comprensión y manipulación en las herramientas. La representación puede ser de diferentes tipos (por ejemplo, gráfica, textual, etc).

La semántica es el significado de los modelos, es decir, que deben estar basados en meta-modelos. Si uno de los tres aspectos de la trilogía (semántica, sintaxis, estilística) no está definido rigurosamente, ya no se pueden garantizar las propiedades de calidad. Por ejemplo, un UIDL que carece de semántica puede sufrir de incorrección y falta de expresividad. Un UIDL que no posee estilística pueden causar la falta de estilo y, por tanto, de la falta de expresividad [INT06].

II.2.4. Proceso de diseño de la IU en un MB-UIDE

El diseño de la IU es el proceso de crear y refinar el MIU. En términos del diseño de la IU también hay desacuerdo sobre cuál es el mejor método para el modelado de la IU. En la figura II.8, se muestra un diagrama tradicional que describe el proceso de diseño en MB-UIDE. Algunos MB-UIDE pueden ofrecer herramientas de modelado para la edición de los modelos o asistentes de modelado para ayudar a los desarrolladores. De ellas se espera que eviten a los desarrolladores de IU los detalles tediosos de los modelos y su notación, pudiendo así enfocar su atención en el diseño del IU. Además, algunos MB-UIDE poseen asistentes de modelado que pueden realizar algunas funciones como pruebas de los modelos y esto ofrece a los desarrolladores retroalimentación sobre el proceso de diseño.

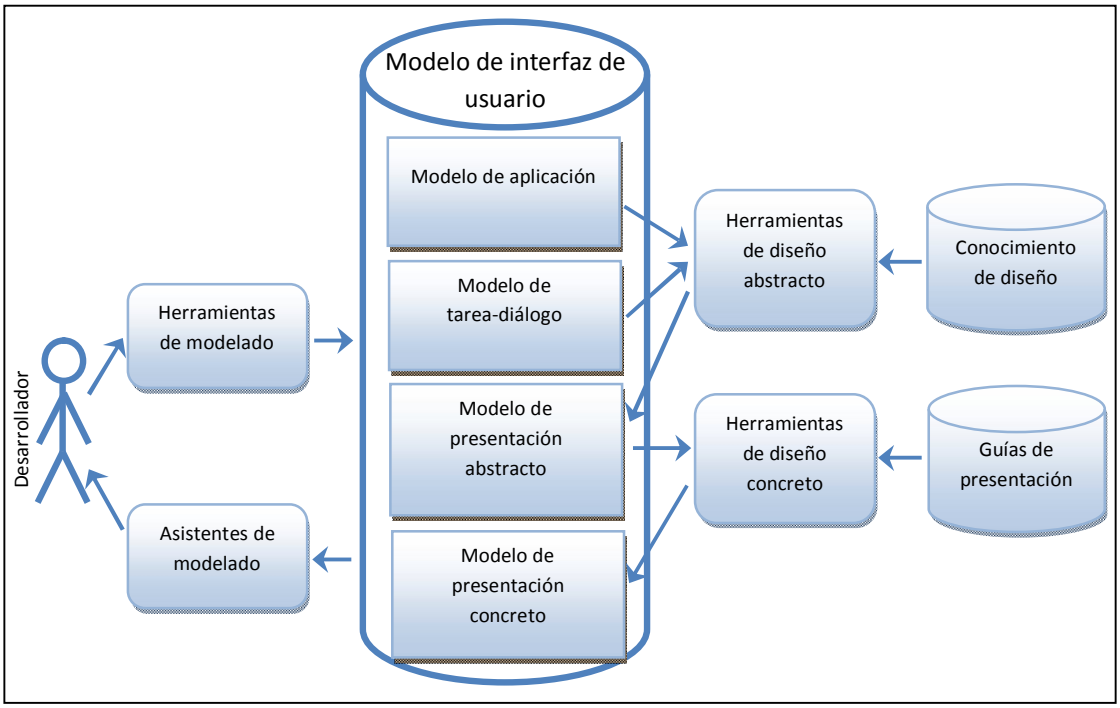


Figura II.8. Diseño de IU con MB-UIDE

II.2.4.1. Diseño centrado en el usuario

El Diseño centrado en el usuario (DCU) [THE09] es una filosofía de diseño moderna y ampliamente utilizada establecida en la idea de que los usuarios deben tener en cuenta el escenario en el diseño de cualquier sistema de computación. Los usuarios, diseñadores y personal técnico trabajan juntos para articular las aspiraciones, necesidades, y limitaciones del usuario y crear un sistema que considere estos elementos. Generalmente, los proyectos de diseño centrado en el usuario son completados con estudios etnográficos del ambiente en el cual los usuarios estarán interactuando con el sistema. Esta práctica es similar, pero no idéntica al Diseño Participativo, el cual enfatiza la posibilidad de que los usuarios finales contribuyan activamente a través de sesiones de diseño compartido y talleres.

Los principios para el desarrollo de dicho proceso pasan por:

1. Conocer al usuario y sus tareas, según lo cual los diseñadores deben saber quiénes son los usuarios, estudiando sus capacidades cognitivas, comportamientos, y actitudes, y por otro lado la naturaleza de las tareas que llevan a cabo.
2. Realizar medidas empíricas. En cuanto sea posible, y cuanto antes en el proceso de desarrollo, a los usuarios se les deben proporcionar prototipos y simulaciones con los que llevar a cabo sus tareas, y con ellos deben observarse, grabarse y analizarse las prestaciones y las reacciones que estos usuarios experimentan.
3. Aplicar un diseño iterativo. Fruto de observar problemas con la puesta en práctica de evaluaciones reiteradas, se debe proceder, a través del diseño, a eliminar aquellos problemas que surjan en una versión.

Existe un estándar internacional que es la base para muchas metodologías DCU. Este estándar (ISO 13407: Human-centred design process) define un proceso general para incluir las actividades centradas en el humano a través de un ciclo de vida de desarrollo, pero no especifica métodos exactos.

Basados en esta filosofía se derivan muchas metodologías o métodos de desarrollo de interfaces de usuario.

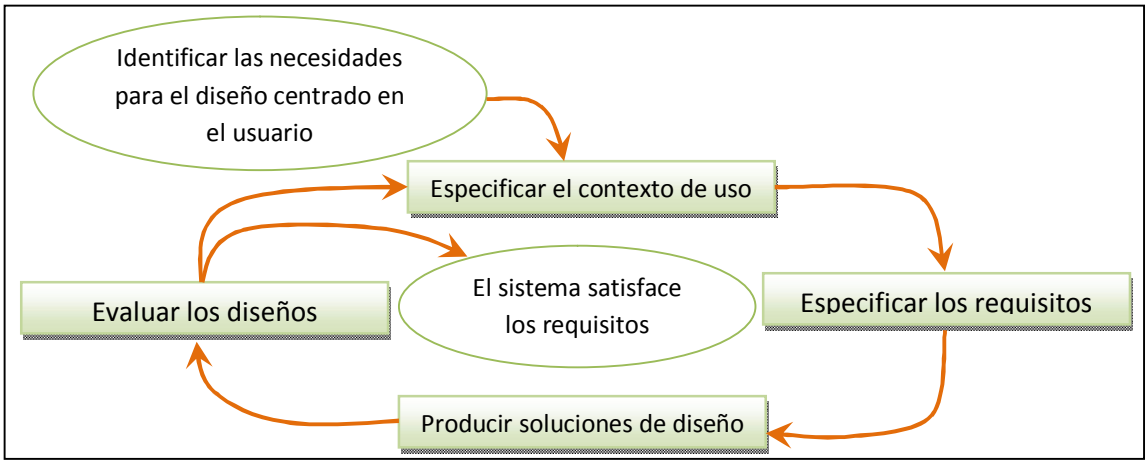


Figura II.9. Proceso de diseño centrado en el usuario, basado en [THE09]

Existen cuatro actividades principales en el ciclo de vida principal de este proceso (figura II.9):

1. Especificar el contexto de uso. Identificar las personas que usarán el producto, para qué lo usarán, y bajo qué condiciones lo usarán.
2. Especificar requisitos. Identificar cualquier requisito de negocios y objetivos de usuario que deben alcanzarse para que el producto sea exitoso.
3. Crear soluciones de diseño. Esta parte del proceso puede ser realizada en niveles, y se pueden construir desde conceptos sencillos a diseños completos.
4. Evaluar diseños. La parte más importante de este proceso es que la evaluación – idealmente a través de una prueba de usabilidad con usuarios reales – debe ser tan integral como lo son las pruebas de calidad para el desarrollo de software.

II.3. MARCO EMPÍRICO

II.3.1. INTRODUCCIÓN

Para la prueba del prototipo se ha considerado la generación de IU para algunas tareas que se llevan a cabo utilizando la plataforma Moodle.

Por lo tanto, en este marco se presenta una breve referencia al e-learning con sus principales ventajas y problemas que impiden su implementación en todo su potencial. También, se describen las características y los módulos más sobresalientes de la plataforma Moodle. Finalmente, se especifican las tareas pertenecientes a los módulos que se consideran para la prueba del prototipo.

II.3.2. E-Learning

II.3.2.1. Definición

El e-Learning es el suministro de programas educacionales y sistemas de aprendizaje a través de medios electrónicos. El e-Learning se basa en el uso de una computadora u otro dispositivo electrónico (por ejemplo, un teléfono móvil) para proveer a las personas de material educativo. La educación a distancia creó las bases para el desarrollo del e-Learning, el cual viene a resolver algunas dificultades en cuanto a tiempos, sincronización de agendas, asistencia y viajes, problemas típicos de la educación tradicional.

Así mismo, el e-Learning puede involucrar una mayor variedad de equipo que la educación en línea. El término de e-Learning o educación electrónica abarca un amplio paquete de aplicaciones y procesos, como el aprendizaje basado en Web, capacitación basada en computadoras, salones de clases virtuales y colaboración digital (trabajo en grupo) [MEN03].

II.3.2.2. Ventajas del e-Learning

A continuación se expresa lo que los expertos consideran como las ventajas más importantes de la educación electrónica [MEN03]:

Mayor productividad: Las soluciones de aprendizaje electrónico como la capacitación basada en Web (WBT, web-based training) y la capacitación basada en computadora (CBT computer-based training) permite a los alumnos estudiar desde su propio escritorio. La entrega directa de los cursos puede disminuir los tiempos muertos que implican una escasa productividad y ayuda a eliminar costos de viajes.

Entrega oportuna: Durante la puesta en marcha de un nuevo producto o servicio, el e-Learning puede proveer entrenamiento simultáneo a muchos participantes acerca de los

procesos y aplicaciones del nuevo producto. Un buen programa de e-Learning puede proveer la capacitación necesaria justo a tiempo para cumplir con una fecha específica de inicio de operaciones.

Capacitación flexible: Un sistema e-Learning cuenta por lo general con un diseño modular. En algunos casos, los participantes pueden escoger su propia ruta de aprendizaje. Adicionalmente, los usuarios pueden marcar ciertas fuentes de información como referencia, facilitando de este modo el proceso de cambio y aumentando los beneficios del programa.

Ahorros en los costos por participante: Tal vez el mayor beneficio del e-Learning es que el costo total de la capacitación por participante es menor que en un sistema tradicional guiado por un instructor. Sin embargo, los programas de e-Learning diseñados a la medida pueden de entrada ser más costosos debido al diseño y desarrollo de los mismos. Se recomienda llevar a cabo un análisis minucioso para determinar si el e-Learning es la mejor solución para sus necesidades de capacitación y adiestramiento antes de invertir en el proyecto.

II.3.2.3. Principales Problemas

Entre las principales barreras que han impedido la integración de estas tecnologías del e-Learning en los programas de capacitación, se encuentran [MEN03]:

I. Estructura organizacional y tradicionalismo.
II. La falta de ejemplos de mejores prácticas.
III. La falta de soporte y experiencia.
IV. La falta de comprensión y visión acerca del e-Learning.
V. La falta de recurso humano y aceptación por parte del usuario.
VI. Organizaciones y procesos tradicionales.
VII. La falta de de habilidad por parte de profesores e instructores, aunada a una actitud negativa.
VIII. Falta de acciones estratégicas.
IX. Falta de entrenamiento y soporte a los profesores e instructores.
X. El tiempo requerido para la preparación del material.

II.3.3. La plataforma e-Learning

Es una herramienta enfocada en la educación en la que se integran las tecnologías de información y elementos pedagógicos. Es un instrumento potente para la enseñanza que se utiliza con mayor frecuencia por las grandes ventajas que brinda.

Es de gran importancia que esta herramienta sea accesible y usable por la gran variedad de usuarios que desean aprender. Por ello las pruebas del prototipo se realizaron para un grupo de interfaces de e-learning y un grupo de usuarios con variadas características.

II.3.3.1. Moodle

Se toma como ejemplo Moodle [PLA09], un paquete de software para la creación de cursos y sitios Web basados en Internet. Es un proyecto en desarrollo diseñado para dar soporte a un marco de educación social constructivista.

II.3.3.1.1. Características generales

Moodle es un producto activo y en evolución. A continuación se enumeran algunas de sus características [PLA09]:

- **I.** Diseño general
- **II.** Administración de usuarios
- **III.** Administración de cursos
- **IV.** Módulo de Tareas
- **V.** Módulo de Consulta
- **VI.** Módulo Foro
- **VII.** Módulo Cuestionario
- **VIII.** Módulo Recurso
- **IX.** Módulo Encuesta
- **X.** Módulo Taller

I. Diseño General

El diseño general promueve una pedagogía constructivista social (colaboración, actividades, reflexión crítica, etc.). Es apropiado para el 100% de las clases en línea, así como también para complementar el aprendizaje presencial.

Tiene una interfaz de navegador de tecnología sencilla, ligera, eficiente y compatible.

Es fácil de instalar en casi cualquier plataforma que soporte PHP. Sólo requiere que exista una base de datos (y la puede compartir). Con su completa abstracción de bases de datos, soporta las principales marcas de bases de datos (excepto en la definición inicial de las tablas).

La lista de cursos muestra descripciones de cada uno de los cursos que hay en el servidor, incluyendo la posibilidad de acceder como invitado. Los cursos pueden clasificarse por categorías y también pueden ser buscados - un sitio Moodle puede albergar miles de cursos.

Se ha puesto énfasis en una seguridad sólida en toda la plataforma. Todos los formularios son revisados, las cookies encriptadas, etc. La mayoría de las áreas de introducción de texto (recursos, mensajes de los foros etc.) pueden ser editadas usando el editor HTML, tan sencillo como cualquier editor de texto de Windows.

II. Administración de Cursos

Un profesor sin restricciones tiene control total sobre todas las opciones de un curso, incluido el restringir a otros profesores. Puede elegir entre varios formatos de curso tales como semanal, por temas o el formato social, basado en debates.

Ofrece una serie flexible de actividades para los cursos: foros, glosarios, cuestionarios, recursos, consultas, encuestas, tareas, chats y talleres.

En la página principal del curso se pueden presentar los cambios ocurridos desde la última vez que el usuario entró en el curso, lo que ayuda a crear una sensación de comunidad. Todas las calificaciones para los foros, cuestionarios y tareas pueden verse en una única página (y descargarse como un archivo con formato de hoja de cálculo).

Permite un registro y seguimiento completo de los accesos del usuario. Se dispone de informes de actividad de cada estudiante, con gráficos y detalles sobre su paso por cada módulo (último acceso, número de veces que lo ha leído) así como también de una detallada "historia" de la participación de cada estudiante, incluyendo mensajes enviados, entradas en el glosario, etc. en una sola página.

Pueden enviarse por correo electrónico copias de los mensajes enviados a un foro, los comentarios de los profesores, etc. en formato HTML o de texto.

Posee escalas de calificación personalizadas - Los profesores pueden definir sus propias escalas para calificar foros, tareas y glosarios.

Los cursos se pueden empaquetar en un único archivo comprimido utilizando la función de "copia de seguridad". Éstos pueden ser restaurados en cualquier servidor Moodle.

III. Módulo de Tareas

Esta función de Moodle, permite al docente especificar la fecha final de entrega de una tarea y la calificación máxima que se les asigna a los alumnos. Los estudiantes pueden subir sus tareas (en cualquier formato de archivo) al servidor. Se registra la fecha en que se han subido. Se permite enviar tareas fuera de tiempo, pero el profesor puede ver claramente el tiempo de retraso.

Para cada tarea en particular, puede evaluarse a la clase entera (calificaciones y comentarios) en una única página con un único formulario. Las observaciones del profesor

se adjuntan a la página de la tarea de cada estudiante y se le envía un mensaje de notificación. El profesor tiene la posibilidad de permitir el reenvío de una tarea tras su calificación (para volver a calificarla).

IV. Módulo Cuestionario

En este módulo se pueden encontrar las siguientes características:

- Los profesores pueden definir una base de datos de preguntas que podrán ser reutilizadas en diferentes cuestionarios.
- Las preguntas pueden ser almacenadas en categorías de fácil acceso, y estas categorías pueden ser "publicadas" para hacerlas accesibles desde cualquier curso del sitio.
- Los cuestionarios se califican automáticamente, y pueden ser recalificados si se modifican las preguntas. Además pueden tener un límite de tiempo a partir del cual no estarán disponibles. El profesor puede determinar si los cuestionarios pueden ser resueltos varias veces y si se mostrarán o no las respuestas correctas y los comentarios.

  Las preguntas y las respuestas de los cuestionarios pueden ser mezcladas (aleatoriamente) para disminuir las copias entre los alumnos.
- Las preguntas pueden crearse en HTML y con imágenes o importarse desde archivos de texto externos.
- Los intentos pueden ser acumulativos, y acabados tras varias sesiones.
- Las preguntas de opción múltiple pueden definirse con una única o múltiples respuestas correctas.
- Pueden crearse preguntas de respuesta corta (palabras o frases), tipo verdadero/falso, de emparejamiento, aleatorias, numéricas (con rangos permitidos), o de respuesta incrustada (estilo "cloze") con respuestas dentro de pasajes de texto.
- Pueden crearse textos descriptivos y gráficos.

V. Módulo Encuesta

Este módulo cuenta con las siguientes características:

- Se proporcionan encuestas ya preparadas (COLLES, ATTLS) y contrastadas como instrumentos para el análisis de las clases en línea.
- Los informes de las encuestas están siempre disponibles, incluyendo muchos gráficos. Los datos pueden descargarse con formato de hoja de cálculo Excel o como archivo de texto CVS.
- La interfaz de las encuestas impide la posibilidad de que sean respondidas sólo parcialmente.

- A cada estudiante se le informa sobre sus resultados comparados con la media de la clase.

VI. Módulo Taller

- Permite la evaluación de documentos entre iguales, y el profesor puede gestionar y calificar la evaluación.
- Admite un amplio rango de escalas de calificación posibles.
- El profesor puede suministrar documentos de ejemplo a los estudiantes para practicar la evaluación.
- Es muy flexible y tiene muchas opciones.

Como en este trabajo se harán pruebas sobre los módulos cuestionarios, calificaciones y visualización del estudiante, a continuación se muestran capturas de pantallas de los módulos correspondientes en Moodle.

Los tipos de usuarios elegidos son dos estudiantes, un profesor y el padre de un alumno.

Módulo cuestionarios

Este módulo permite al profesor diseñar y aplicar cuestionarios. Existe una amplia variedad de Tipos de preguntas (opción múltiple, verdadero/falso, respuestas cortas, etc.).

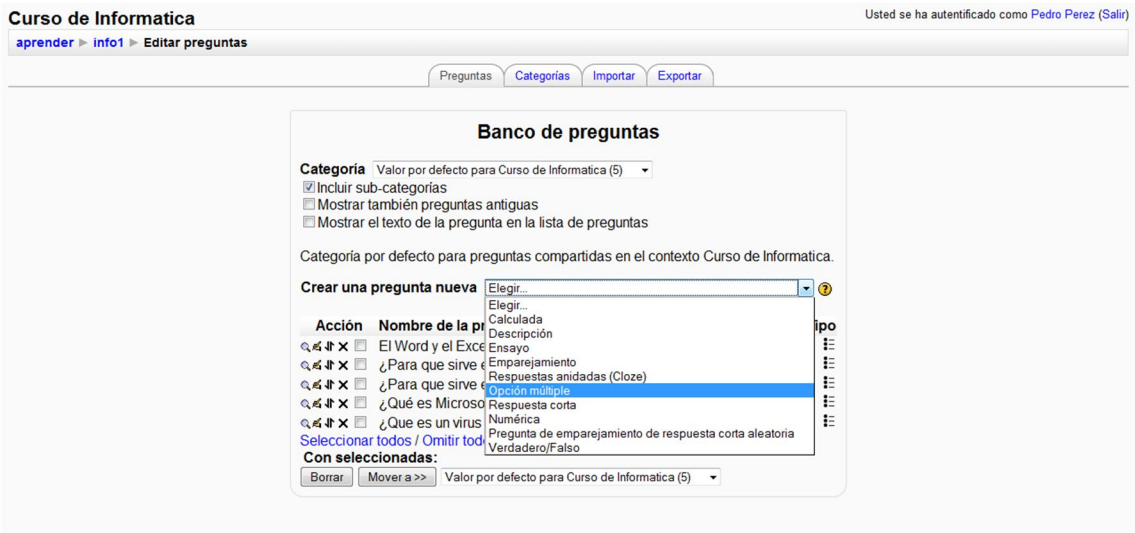


Imagen II.2. Selección de tipo de pregunta

En la imagen II.3 se observa las opciones que posee el profesor al momento de crear un nuevo cuestionario.

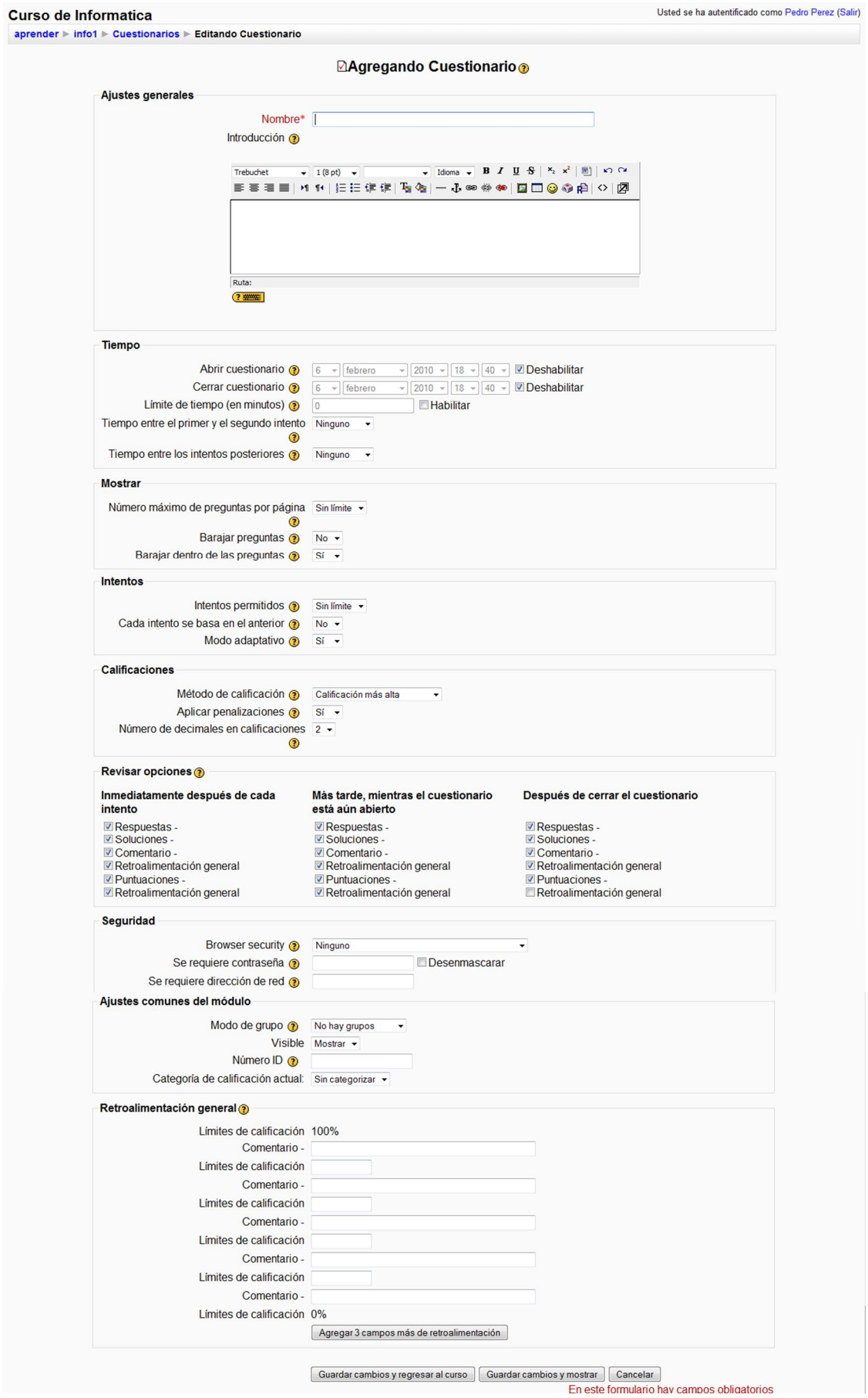


Imagen II.3. Módulo cuestionarios de la plataforma Moodle.

Mostrar Calificaciones

Las actividades provistas por el sistema, permiten ser configuradas de manera que las mismas puedan ser calificadas, imagen II.4. Las calificaciones de las distintas actividades propuestas en el curso pueden ser visualizadas por los usuarios, imagen II.5.

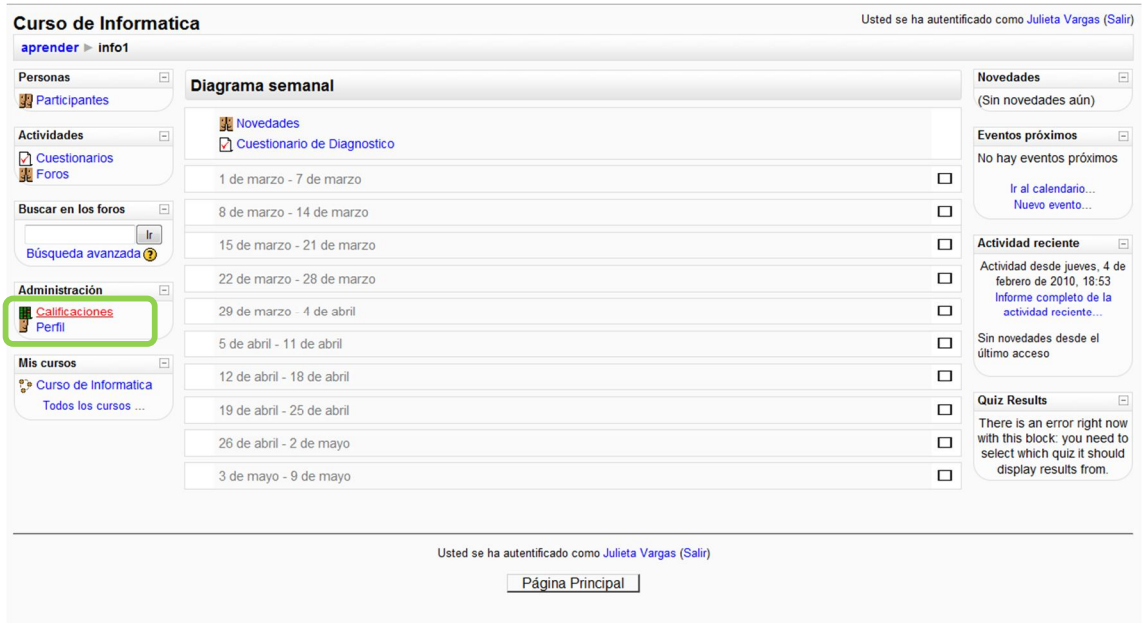


Imagen II.4. Selección de la opción para visualizar las calificaciones.

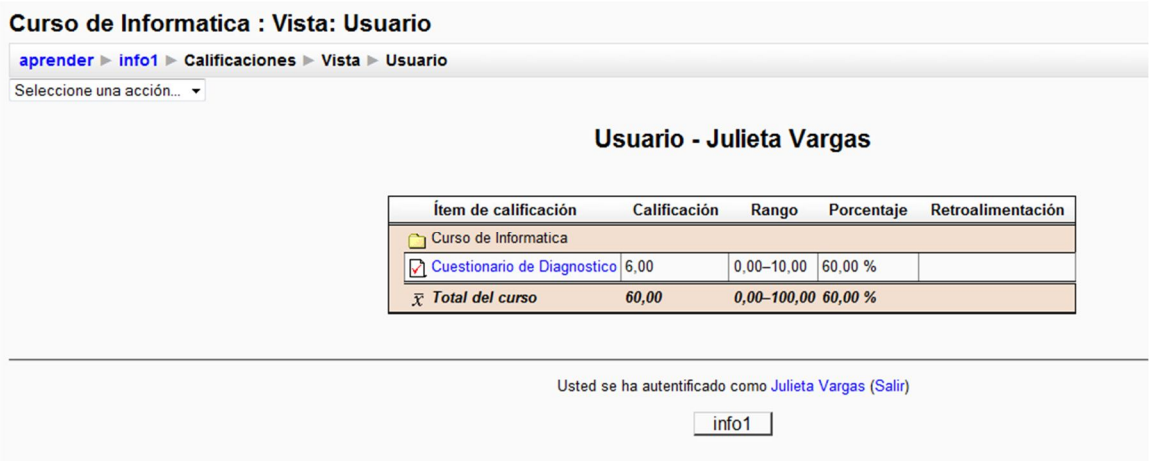


Imagen II.5. Visualización de las calificaciones.

Visualización del estudiante

Tras hacer clic en el nombre de un cuestionario (imagen II.6), aparecerá la página del cuestionario que muestra su nombre e introducción. Normalmente la introducción ofrece información sobre el propósito del cuestionario y la forma de evaluación. Es posible que la página muestre asimismo las fechas y horas de apertura y cierre del cuestionario (imagen II.7).

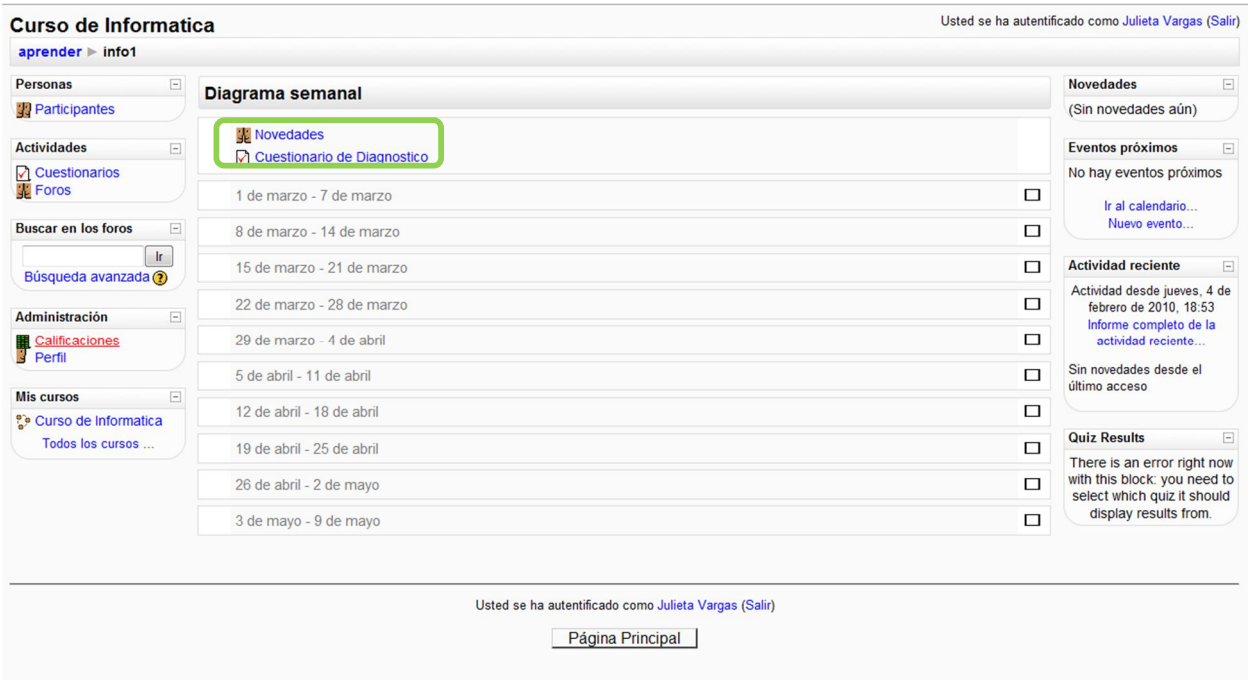


Imagen II.6. Selección de la opción para visualizar el cuestionario.

Imagen II.7. Visualización de la configuración del cuestionario.

Imagen II.8. Visualización del cuestionario.

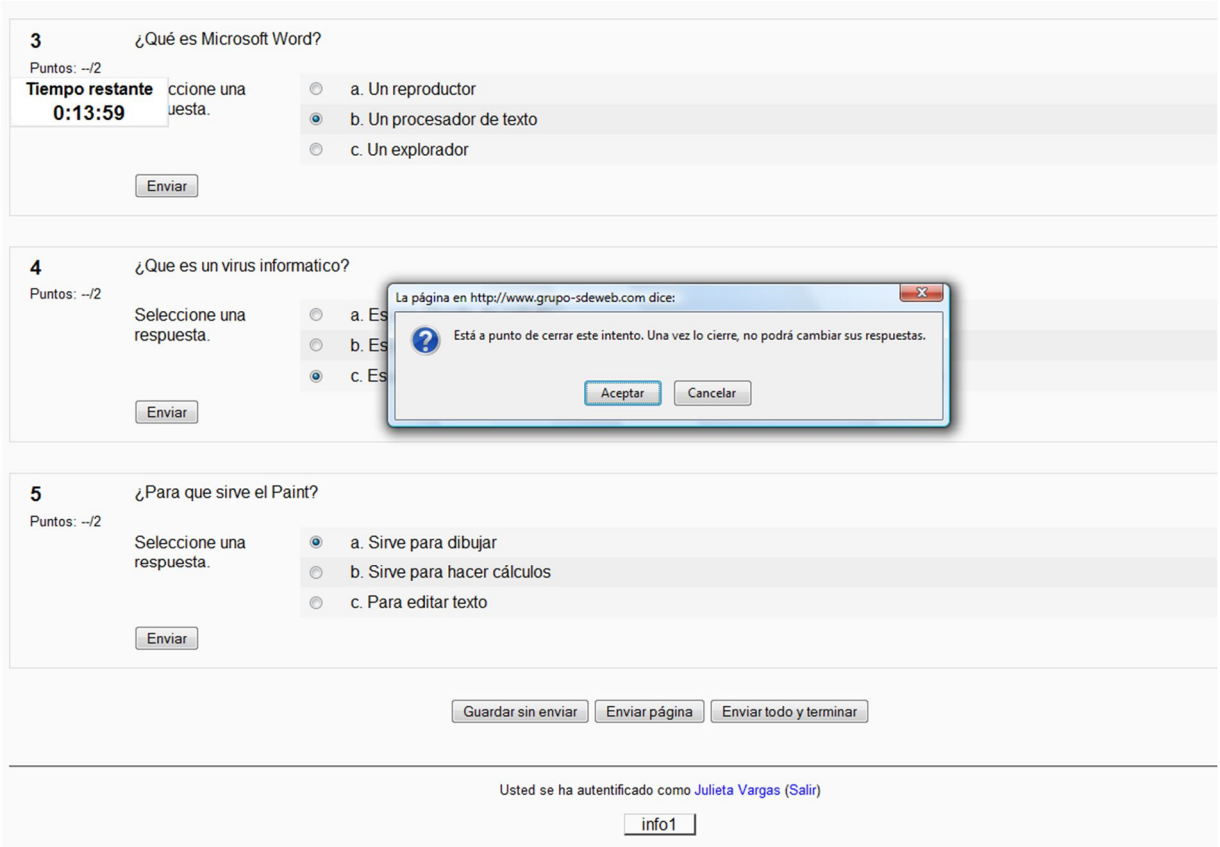


Imagen II.9. Mensaje de confirmación de envío de respuestas del cuestionario.

Imagen II.10. Visualización de resultados del cuestionario.

Imagen II.11. Visualización de las calificaciones para el cuestionario.

Capítulo III
Diseño y Desarrollo del Prototipo

CAPÍTULO III

DISEÑO Y DESARROLLO DEL PROTOTIPO

III.1. INTRODUCCIÓN

En este capítulo se presenta el desarrollo del prototipo para el diseño universal de interfaces de usuario basado en ontologías. Como se mencionó anteriormente, este prototipo permitirá diseñar interfaces de usuario mediante ontologías, dichas ontologías representarán los modelos de tareas, dominio, usuario y mapeo. Además, permitirá generar automáticamente, basándose en los modelos previos, el modelo de interfaz de usuario abstracta y, por último, los modelos de interfaz de usuario concreta considerando las características de los usuarios.

Para su construcción se siguen las etapas comunes del desarrollo del software. Lo que se agrega en este trabajo es el desarrollo de la ontología que utilizará el diseñador para especificar los modelos, por ello, existe una etapa de diseño y construcción de la misma. Además, el prototipo estará soportado por un conjunto de herramientas ya existentes. En el proceso de desarrollo se prevé una etapa de selección de las mismas. En la cual se elegirá un editor de ontologías y un lenguaje o tecnología para programar las reglas de transformación que luego permitirán generar automáticamente el modelo de interfaz de usuario abstracta y los modelos interfaz de usuario concreta.

En resumen, las etapas son las de especificación de requisitos, diseño y construcción de ontologías, identificación de las tecnologías que permitirán al diseñador manipular dichas ontologías, programación y pruebas del prototipo, (figura III.1).

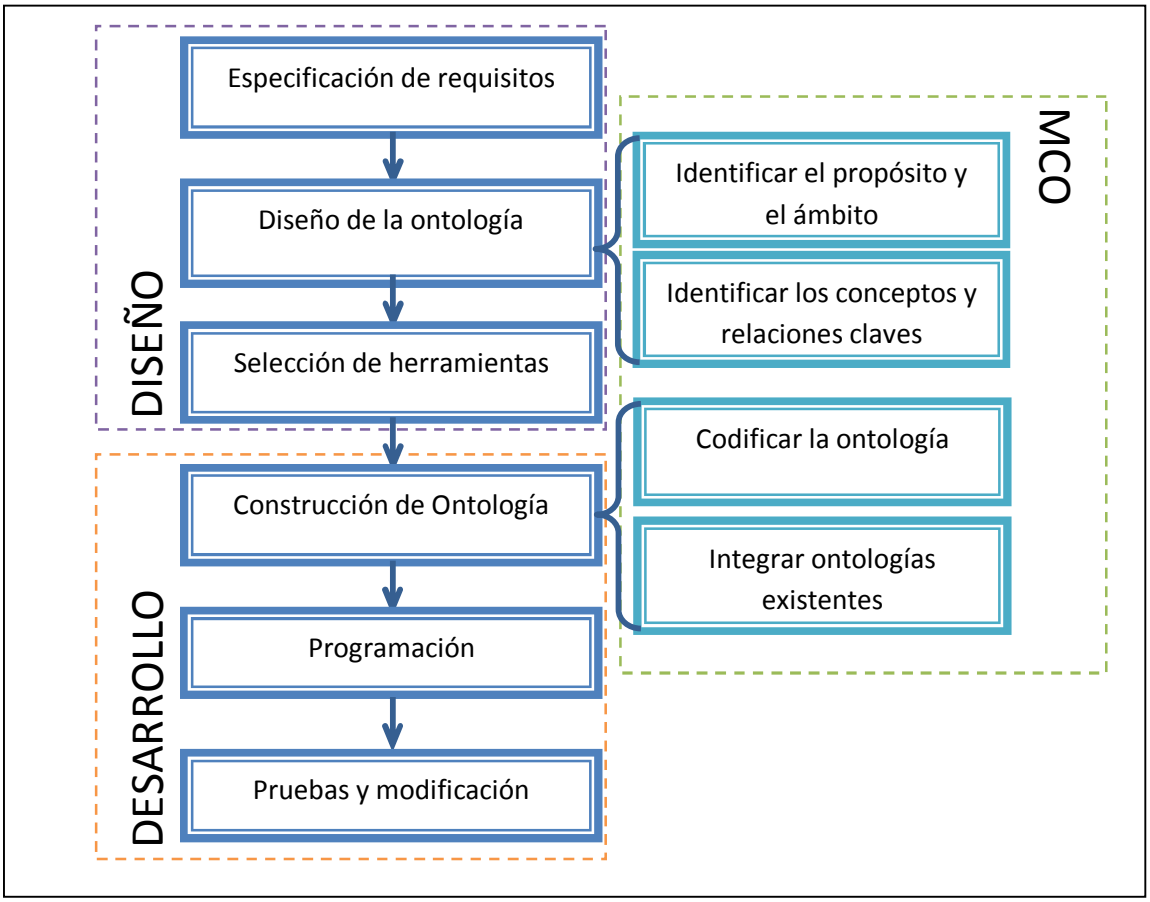


Figura III.1. Proceso de diseño, desarrollo y evaluación del prototipo.

III.2. DISEÑO DEL PROTOTIPO

III.2.1. INTRODUCCIÓN

En esta etapa se presentan las siguientes fases: especificación de requisitos, diseño de la ontología y selección de herramientas.

La *especificación de requisitos* se desarrolla y surge a partir de los problemas reconocidos en la investigación.

La fase de *diseño dela ontología*, en la que se diseña la que se utilizará para representar la IU, es decir, se utilizará para definir conceptos relacionados a los diferentes modelos de acuerdo al formalismo de UsiXML. Se identifican el propósito y el ámbito de la misma. Además, se agrega una etapa de identificación de conceptos y relaciones clave, lo cual permitirá identificar y seleccionar las ontologías (o parte de ellas) a reusar y las clases a construir que formarán la ontología de IU final.

Como última fase dentro del diseño se hace una *selección de herramientas* que permitirá el desarrollo del prototipo, aquí se tienen en cuenta herramientas utilizadas por la comunidad de investigadores y desarrolladores.

III.2.2. ESPECIFICACIÓN DE REQUISITOS

De la investigación realizada se desprenden los siguientes problemas:

1) Complejidad del lenguaje para describir los modelos, que muchas veces son difíciles de aprender y usar.
2) La mayoría de las herramientas para desarrollar IU tienen dificultades para considerar múltiples parámetros en los modelos utilizados.
3) Frecuentemente, los desarrolladores se enfrentan con el obstáculo de tener que aprender y usar nuevas herramientas y metodologías para desarrollar interfaces.
4) Dificultad para representar en los modelos de Interfaz de Usuario Concreta los parámetros correspondientes a los modelos de tareas, dominio y usuario. Es decir, se observa una falta de fidelidad con respecto a la correspondencia de estos modelos con el modelo concreto.

REQUISITOS

A partir de estos problemas se enuncian los siguientes requisitos:

Req.I. Utilizar ontologías para representar los modelos de IU, las preferencias de los usuarios y las guías de diseño.

Req.II. Se deberá permitir el uso de ontologías para especificar el conocimiento de los modelos de Tareas, de Dominio, de Mapeo y de Usuario.

Req.III. El prototipo permitirá la especificación de Guías de diseño mediante ontologías, dichas guías indicarán que elementos debe tener la interfaz de acuerdo a las características de los usuarios.

Requisitos funcionales

Req.IV. El prototipo deberá permitir la modificación del conocimiento representado en la ontología.

Req.V. El prototipo deberá permitir la modificación de la estructura de la ontología que representa los modelos, sus relaciones, atributos, etc.

Req.VI. El prototipo permitirá la modificación del conocimiento sobre las Guías de diseño encerrado en la ontología.

Req.VII. El prototipo permitirá la modificación de la estructura de la ontología que representan las Guías de diseño.

Req.VIII. El prototipo deberá generar automáticamente el modelo de AUI desde el conocimiento encerrado en las clases de la ontología que representan los modelos de Tareas, de Dominio, de Mapeo y de Usuario.

Req.IX. El prototipo deberá generar el modelo de CUI desde el modelo de AUI tomando en cuenta el conocimiento encerrado en las clases de la ontología que representan las Guías de diseño.

III.2.3. DISEÑO DE ONTOLOGÍA

En el proceso del diseño de la ontología se utilizaron las características y las fases más importantes y pertinentes a este trabajo de varias metodologías para construcción de ontologías [MIZ95, GRU96, GRU94, NOY05]**,** este proceso permitirá obtener la estructura de conocimientos necesaria para representar los modelos de IU.

Identificación el propósito y el ámbito

Propósito: permitir el modelado del conocimiento sobre una IU. Se desea que un diseñador de interfaces pueda crear un modelo de IU mediante el uso de esta ontología. Como se comentó anteriormente, hasta ahora, para representar conceptualmente una interfaz se utilizan diferentes modelos que en conjunto forman el modelo completo de una IU. Existen proyectos que pretenden estandarizar un lenguaje de representación de estos modelos. Aquí se trabajará con UsiXML y los modelos de usuario, dominio, interfaz abstracta y concreta propuestos por el mismo. Este lenguaje está basado en XML y en este trabajo se utilizará OWL para reescribir el lenguaje. También se desea permitir el reuso de la estructura de la ontología y del conocimiento representado en ella. Para el modelo de usuario se necesita representar los datos de los usuarios desde el nombre, edad, dirección, etc., hasta características de la personalidad. En cuanto al modelo de tareas y de mapeo, se requiere poseer clases que representen la notación de UsiXML. Por último, se necesitan ciertas clases que representen guías de diseño que permitan determinar los elementos de interfaz más adecuados para usuarios con características particulares.

Ámbito: En el presente trabajo el ámbito de la ontología es el diseño de IU que está formado por diferentes elementos como: modelo de IU, compuesto por diferentes submodelos, y las guías de usabilidad. La ontología se utilizará para clasificar los elementos de tales submodelos. Los submodelos del modelo de IU son: modelo de tareas, de dominio, de mapeo, de AUI, de CUI, de usuario (formado por preferencias y características de los usuarios).

Identificar los conceptos y relaciones claves

Parte de la ontología de IU final será representada por la estructura propuesta por usiXML.

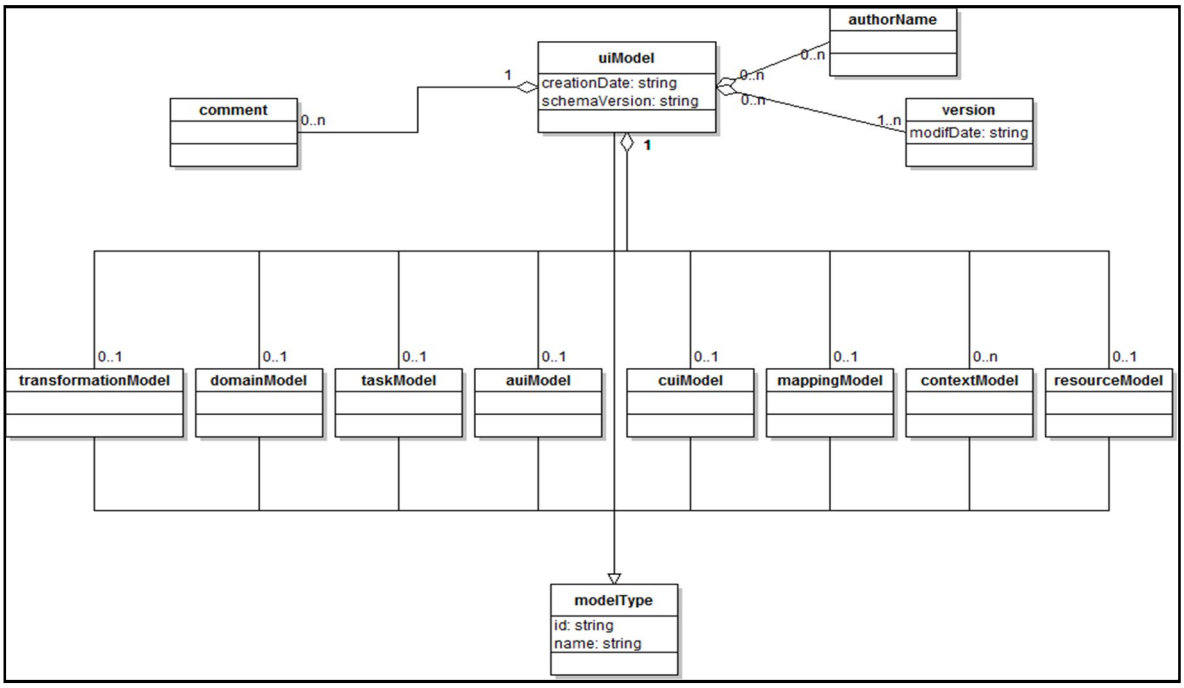


Figura III.2. Esquema de modelos de UsiXML.

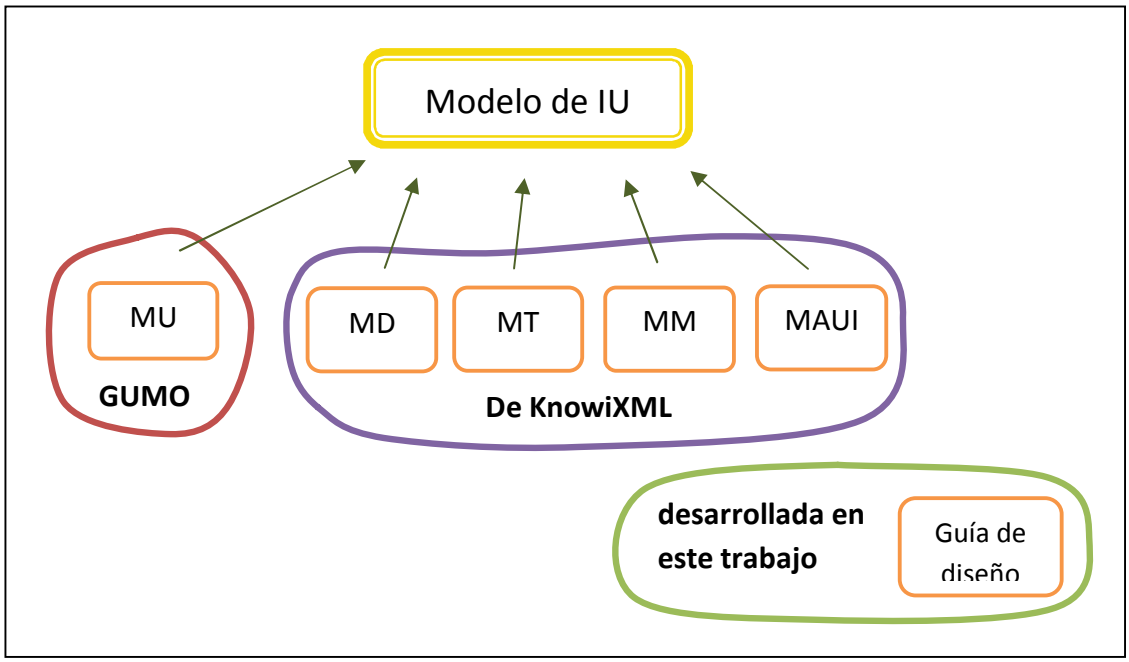
UsiXML es un lenguaje para describir interfaces que permite separar esta descripción en diferentes modelos.

El UiModel es el modelo más alto que posee las características comunes compartidas por todos los submodelos componentes de la IU. El modelo de dominio es una descripción de la clase de objetos manipuladas por un usuario mientras interactúa con el sistema, por ejemplo para un sistema de e-learning los elementos serían las notas, evaluaciones, cursos, etc. El modelo de tareas es el modelo que describe las tareas interactivas como son vistas por el usuario final que interactúa con el sistema, por ejemplo, ver notas, tomar evaluación, realizar curso, etc. El modelo de contexto describe el contexto dentro del cual el usuario final está llevando a cabo una tarea interactiva, por ejemplo, si se trata de una aplicación web el usuario puede estar utilizando la aplicación en cualquier lugar y sobre cualquier tipo de computadora. El modelo abstracto define un esquema de interacción de espacios y navegación, y objetos abstractos de selección que son independientes de cualquier modalidad de interacción (por ejemplo, gráfica, vocal, video, etc.), de cualquier contexto de uso o de plataforma en la cual será utilizado el sistema. El modelo concreto transforma en objetos concretos el modelo abstracto para un contexto de uso dado, es decir, los elementos concretos finales de la interfaz serán ya dependientes de la plataforma, tipos de usuarios, etc., en la que se utilizará el sistema.

Teniendo en cuenta la estructura, propiedades y atributos de estos modelos definidos en UsiXML, se construye la ontología.

Sin embargo, cabe mencionar que el modelo de contexto de uso incluye el modelo de usuario además de información del entorno y plataforma en donde se utilizará el sistema. Como en este trabajo, el objetivo principal es poder desarrollar interfaces universales, es decir, interfaces que satisfagan las características de todos los usuarios, en lugar de utilizar el modelo de contexto de uso en su totalidad, se utilizará solamente el modelo de usuario.

Previamente se analizó la posibilidad de utilizar ontologías que permitan representar la totalidad de la estructura o al menos parte de ella. Se realizó una investigación exploratoria en busca de ontologías a reutilizar. El resultado obtenido se puede observar en el esquema de la figura III.3.



FiguraIII.3. Esquema de ontologías simple.

Se concluyó modelar lo encerrado en rojo (modelo de usuario) con la ontología GUMO, lo encerrado con violeta (modelo de dominio, tareas, mapeo y de interfaz de usuario abstracta) con la ontología de modelos de interfaz de usuario con formalismo UsiXML [FUR04] y lo que está encerrado en verde será desarrollado en este trabajo.

Además de esta ontología que engloba toda la información de la interfaz en modelos, se necesitan algunas clases más como, por ejemplo, las clases para las guías de diseño.

En conclusión, se reusarán las ontologías GUMO para representar el modelo de usuario, la ontología creada para la herramienta KnowiXML para representar los modelos de dominio, tareas, mapeo, interfaz de usuario abstracta. Además, en este trabajo se crearán las clases

Guias para representar a las guías de diseño, *isAllocatedTo* para representar la relación entre el usuario y las tareas, y por último *GUMORelationship* para representar las relaciones entre los usuarios y sus características personales. La figura siguiente representa gráficamente el resultado de esta etapa.

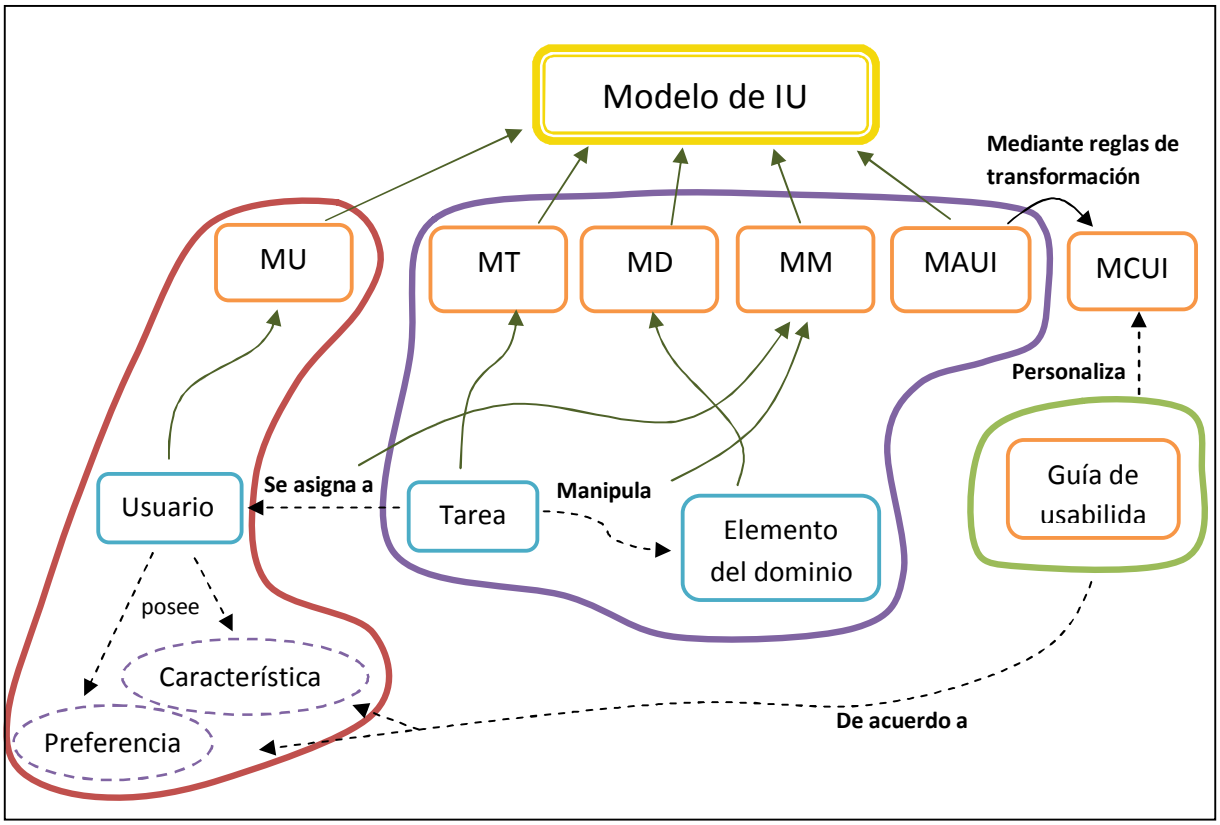


Figura III.4. Esquema de ontologías completo.

A continuación se explicará el proceso de construcción de dichas clases.

Guías de Diseño

Como se expresó en el apartado anterior se deberá crear la clase *Guias* que permitirá modelar guías de diseño. Con esta clase se puede mostrar cómo el prototipo trabaja diferenciando cada uno de los usuarios de acuerdo a sus características particulares, ya que existen guías para diferentes tipos de usuario. Estas guías determinarán los elementos concretos finales que debe poseer la interfaz concreta de acuerdo a ciertas características de los usuarios.

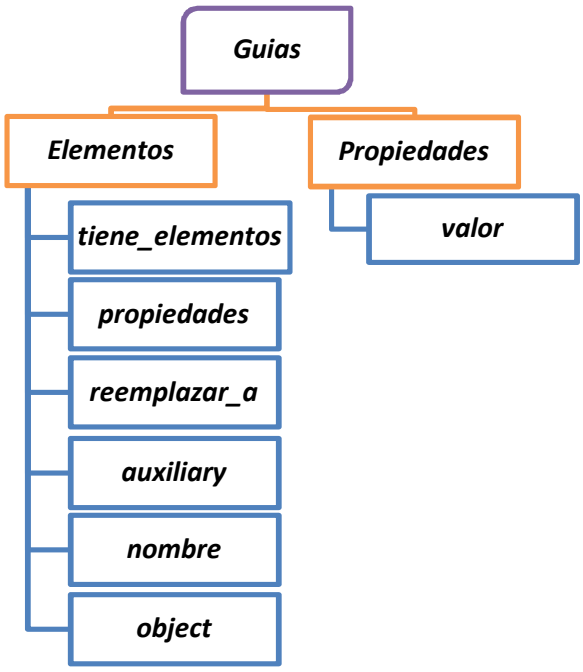


Figura III.5. Diseño de la clase *Guias*.

Esta clase estará compuesta de las subclases *Elementos* y *Propiedades*. La clase *Elementos* posee los elementos concretos que la guía indica que deben utilizarse en una interfaz para una situación particular. Contendrá los atributos:

- *tiene_elementos*: lista de los elementos concretos que deben utilizarse.
- *propiedades*: lista de propiedades de los elementos concretos.
- *reemplazar_a*: indica qué elementos han de ser reemplazados.
- *auxiliary*, *nombre*, *object*: servirán para identificar las características de GUMO que tienen relación con la guía, por ejemplo con la característica nivel cognitivo "bajo".

La clase *Propiedades* se refiere a las propiedades que los elementos concretos pudieran tomar para determinada guía. Las propiedades posibles son definidas en la documentación de UsiXML y las que se utilizarán en este trabajo son explicadas en el capítulo II, en el apartado II.2.1.5.1 UsiXML. La clase contendrá el siguiente atributo:

- *valor*: Cada una de las propiedades de los elementos concretos pueden tener diferentes valores, por ejemplo ***textSize=20***

Por ejemplo se podría tener la siguiente guía de diseño:

"No utilizar texto muy pequeño para el texto del cuerpo, para links y botones para personas que poseen nivel de vista bajo".

Para la clase *Elementos* perteneciente a la clase *Guias* se tendría la siguiente instancia.

- *auxiliay*: posee
- *object*: Nivel de vista bajo
- *tiene_elementos*: outputtext
- *reemplazarPor*: outputtext size 16

Esta guía podría ser utilizada para reemplazar los elementos outputtext, con tamaño de letra por defecto 12, de UsiXML por otro outputtext con tamaño de letra 16 para los usuarios que poseen un nivel de vista bajo.

IsAllocatedTo

Esta clase pertenece al UIDL UsiXML, al no estar incluida en la ontología seleccionada para reusar, se decidió agregarla a la misma.

Esta clase será utilizada para representar las relaciones entre los usuarios y las tareas que pueden realizar.

La estructura de esta clase se presenta en la figura III.6.

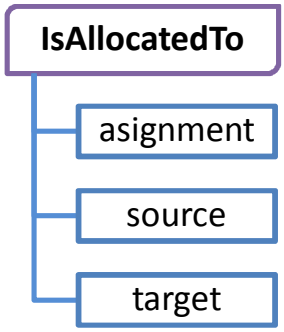


Figura III.6. Diseño de la clase *IsAllocatedTo*.

GumoRelationship

Mediante GUMO se instancian los usuarios, elementos del contexto de uso, características de los usuarios, entre otras cosas. Pero para representar las relaciones de los usuarios con las características que poseen, como la edad, género, experiencia, etc., es necesario agregar una clase que posea los atributos *subject, auxiliary* y *object*, como se vio en el marco referencial, que permitirá establecer las relaciones de los usuarios con sus características.

Por ejemplo, se podría utilizar la clase para representar la siguiente relación:

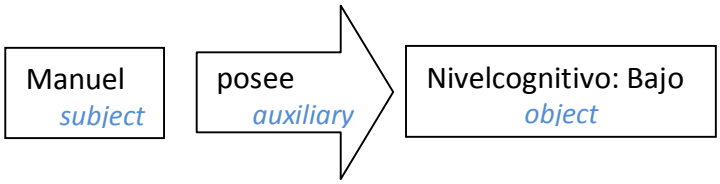


Figura III.7. Ejemplo de una instancia con *GumoRelationship*.

Se diseñó la clase *GumoRelationship* para relacionar las instancias de la ontología GUMO, y esta se visualiza en la figura III.8.

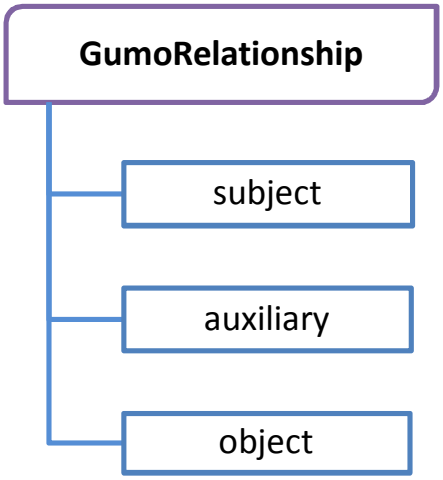


Figura III.8. Diseño de la clase *GumoRelationship*.

III.2.4. SELECCIÓN DE HERRAMIENTAS

En esta etapa se realizó un análisis sobre las herramientas más utilizadas por la comunidad de investigación y de desarrollo de software. Para su selección se tuvo en cuenta la cantidad y calidad de la documentación existente y la comunidad que colabora en el desarrollo o que comparte su experiencia de uso con tales herramientas. Luego de pasar por ese proceso las herramientas seleccionadas fueron: Protégé como editor de ontologías, Jess (JessTab) como motor de reglas de transformación y lenguaje para producirlas.

III.3. DESARROLLO

III.3.1. CONSTRUCCIÓN DE LA ONTOLOGÍA

Habiendo diseñado previamente la ontología completa para modelar la IU y decidido reusar ciertas ontologías que cubren el propósito, en esta etapa se procede a desarrollar la ontología final.

En la fase previa se decidió reusar dos ontologías (GUMO y ontología de modelos con formalismo UsiXML) y desarrollar clases que completen el propósito (*Guias*, *isAllocatedTo*).

III.3.2. CODIFICACIÓN DE LA ONTOLOGÍA

En este apartado se procederá con el desarrollo de las clases mencionadas utilizando Protégé.

Guías de diseño

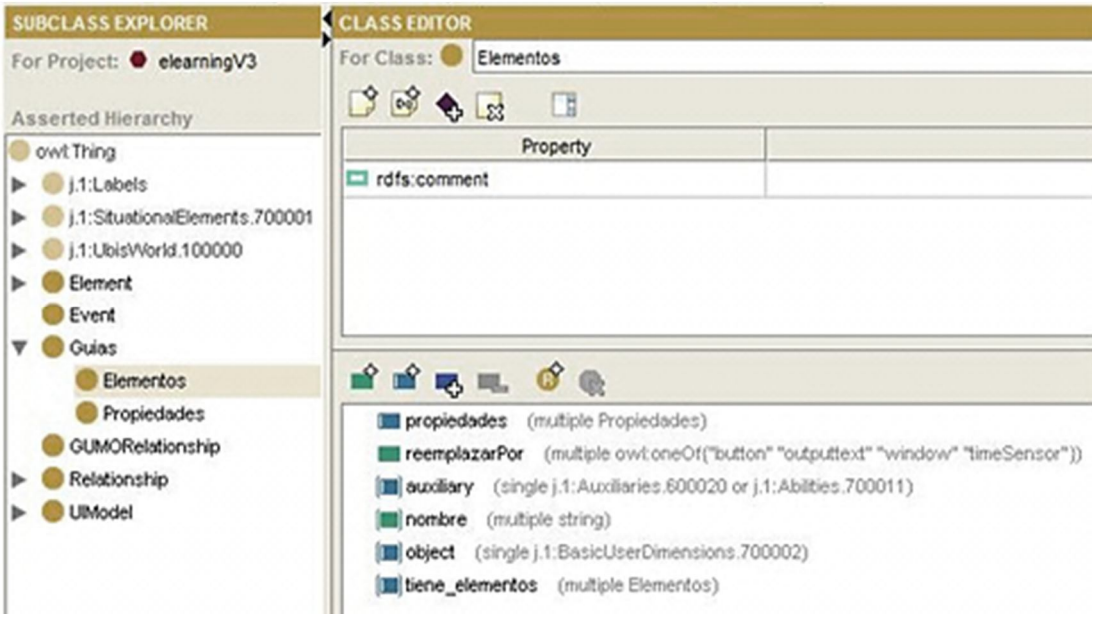


Imagen III.1. Clase *Elementos* perteneciente a la clase *Guias.*

Se observa en la imagen III.1 que la clase *Elementos* posee los atributos *auxiliary* y *object* que se relacionan con GUMO, mediante estos se pueden instanciar características que los usuarios podrían tener. Posee también los atributos *reemplazarPor* y *tiene_elementos* que indican qué elementos concretos reemplazar y por cual hacerlo, respectivamente. En la imagen III.2 se encuentra la clase *Propiedades* que permite instanciar las propiedades de los elementos instanciados en el atributo *tiene_elementos* de la clase anterior.

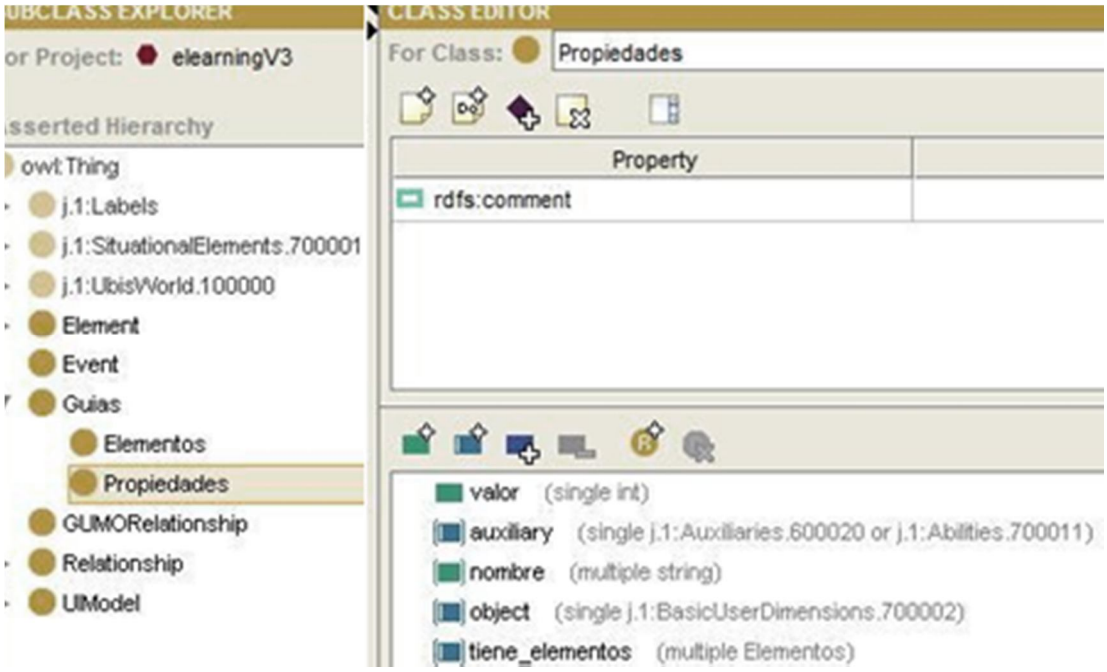


Imagen III.2. Clase *Propiedades* perteneciente a la clase *Guias.*

IsAllocatedTo

Esta clase es subclase del modelo de mapeo. Permitirá por medio de sus atributos conocer la relación existente entre las tareas y los usuarios que pueden realizarlas.

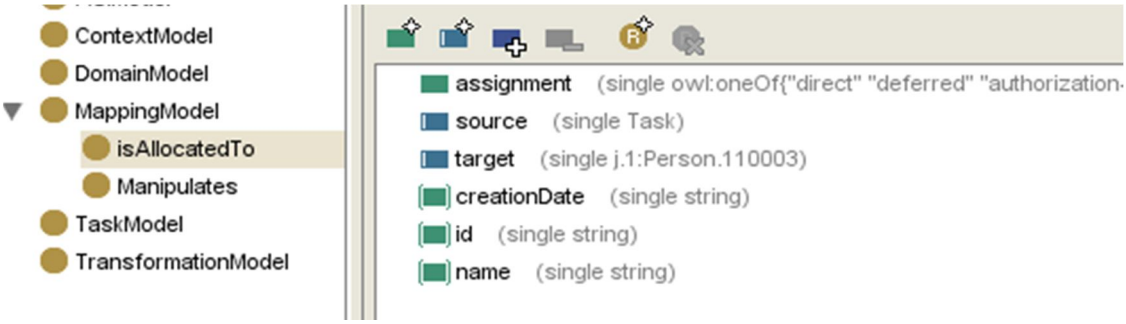


Imagen III.3. Clase *isAllocatedTo* del modelo de mapeo.

GUMORelationship

La clase *GUMORelationship* es la necesaria para poder representar la terna (*subject*, *auxiliary, object)* mencionada anteriormente.

Se observan en la imagen III.4 las propiedades utilizadas: *subject*, *auxiliary* y *object*.

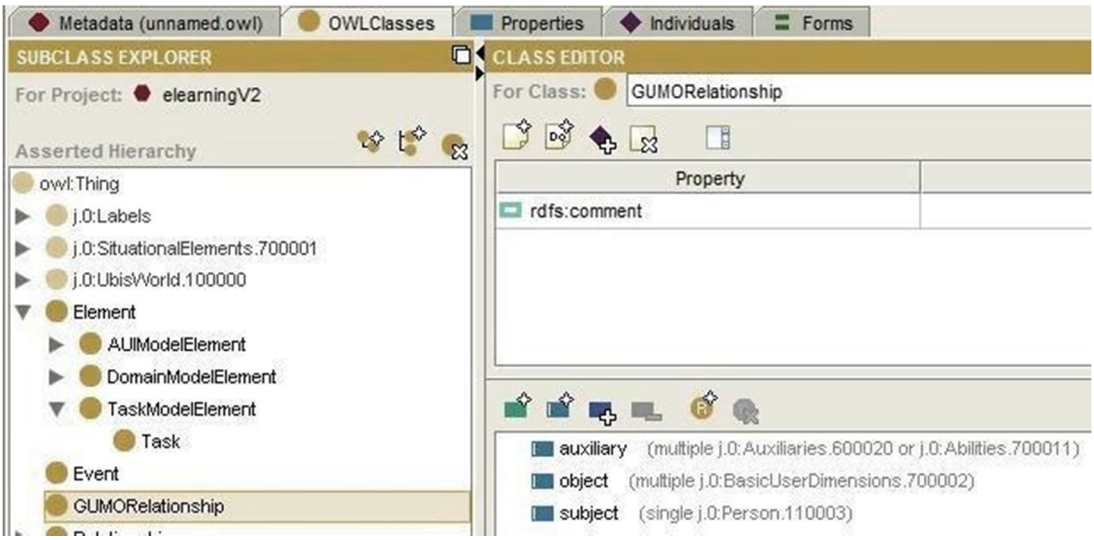


Imagen III.4. Propiedades de la clase *GUMORelationship.*

III.3.3. INTEGRAR ONTOLOGÍAS EXISTENTES

En base a la investigación previamente realizada y bajo el análisis de los problemas y requisitos que surgen de ellos se ha decidido ampliar la ontología con contenido ya existente en ontologías como GUMO y la ontología de modelos con formalismo UsiXML, para ampliar la universalidad y contenido semántico de esta propuesta. A continuación se explica más detalladamente el proceso que llevó a seleccionar tales ontologías.

Ontología de modelos con formalismo UsiXML

La idea de un lenguaje común para describir un sistema para mejorar la comprensión y comunicación entre los humanos tuvo como resultado UML el cual ya es un estándar para visualizar, especificar y documentar los modelos que se crean durante la construcción de software.

En el caso de la IHM existen varios lenguajes para describir interfaces que buscan convertirse en un estándar. De todos ellos se considera los más importantes a UsiXML y XIML ya que no sólo son lenguajes para describir los componentes de una interfaz sino también para describir un modelo de interfaz de usuario. Es decir, un lenguaje para organizar todo lo relacionado con la interfaz y su desarrollo, como las tareas, los usuarios, los elementos del dominio y hasta los elementos concretos de la interfaz.

Se observó que UsiXML posee una gran comunidad con muchos trabajos de investigación y herramientas para modelar y generar interfaces. Esto no sucede para el caso de XIML, entonces se decidió utilizar finalmente UsiXML.

Dentro de estos trabajos de investigación existe uno en el cual desarrollaron una herramienta llamada KnowiUI la cual utiliza ontologías con la sintaxis de UsiXML, pero

como se explicó en los antecedentes no está orientada al diseño universal, la herramienta no tiene soporte y por lo tanto no es funcional. Entonces, se decide reutilizar dicha ontología en este trabajo.

Los creadores de la herramienta KnowiUI crearon una ontología para representar UsiXML, es decir, en lugar de utilizar la escrita en XML reescribieron la sintaxis en el lenguaje RDF[1]. En esta ontología pueden instanciarse el modelo de dominio (Domain Model), el modelo de tareas (Task Model), el modelo de mapeo (Mapping Model) y el modelo de interfaz de usuario abstracta (AUI Model).

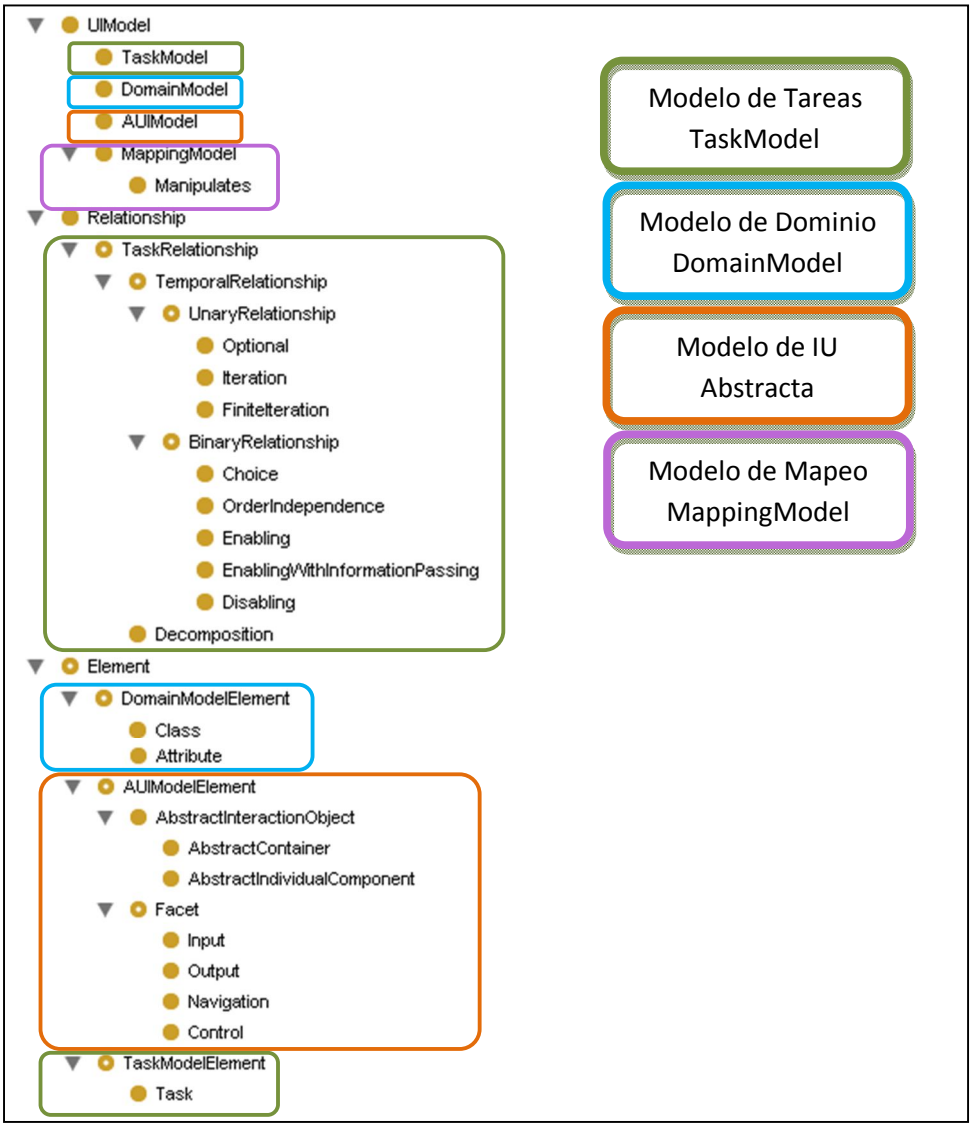


Imagen III.5. Estructura de la ontología con formalismo UsiXML.

[1]**RDF** es un modelo estándar para el intercambio de datos en la Web. RDF tiene características que facilitan el intercambio de los datos incluso si los esquemas subyacentes son diferentes.

Esta ontología es usada por KnowiUI e incluye las clases y atributos de los modelos de tareas, dominio, mapeo y de interfaz abstracta. La estructura de esta ontología se muestra en la imagen III.5.

A continuación se muestran las estructuras y propiedades de los distintos modelos que componen la ontología.

Modelo de dominio

En el modelo de dominio se reflejan la clase de objetos que el usuario podrá manipular mediante la interfaz y que están en relación con la aplicación en particular de que se trate, por ejemplo si se trata de una tienda de venta de libros, uno de los objetos más importantes del dominio serán los libros.

Se observan ahora las propiedades y las subclases de este modelo.

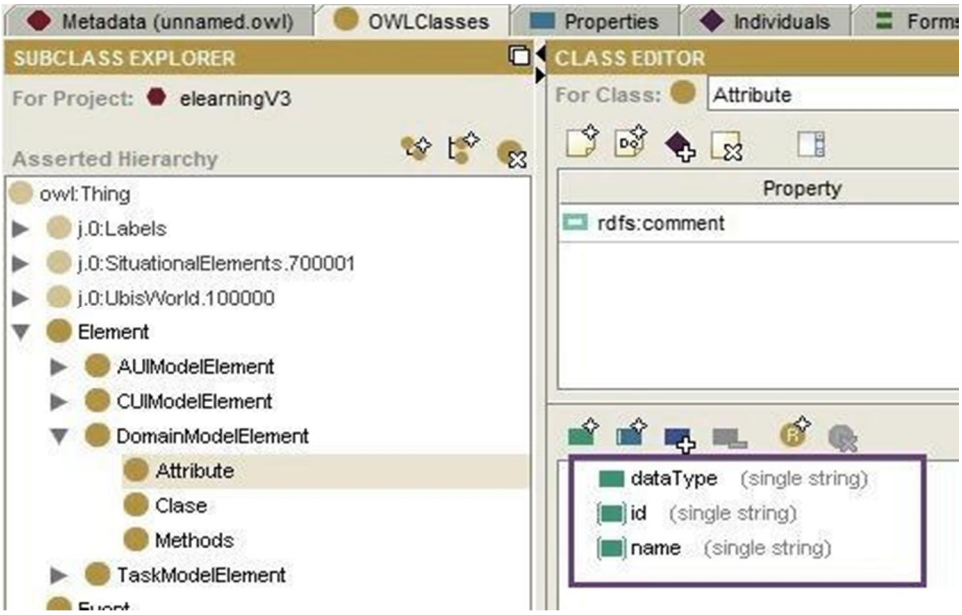


Imagen III.6. Propiedades de la Subclase *Attribute* del modelo de dominio.

En primer lugar la clase *Attribute* que posee los atributos *id*, *name* y *dataType,* que podrá ser utilizada para un objeto en particular con un identificador, su nombre y el tipo de dato. Por ejemplo, para representar el atributo fecha se podría tener la siguiente instancia:

- *id*: 1
- *name*: Fecha de publicación
- *dataType*: date

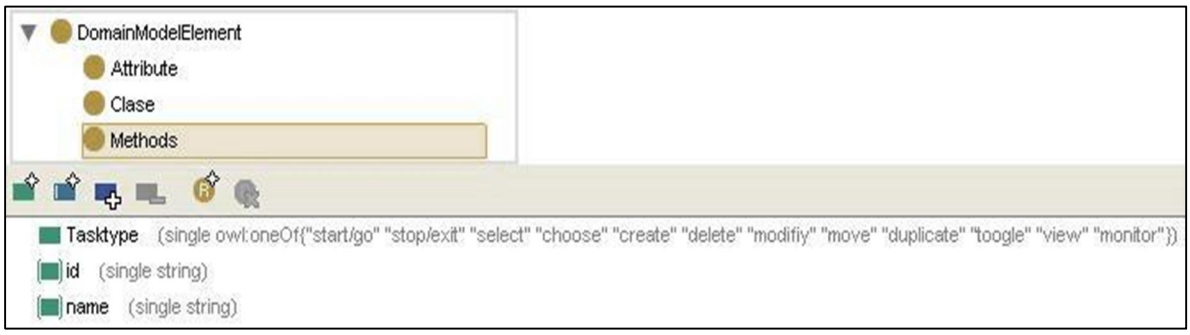


Imagen III.7. Propiedades de la Subclase *Methods* del modelo de dominio.

Dentro de la clase *Methods* se instancian las operaciones que pueden realizarse sobre los elementos de la clase *Clase* a la cual pertenecen. Por ejemplo, podría tener los métodos CRUD (Create, Read, Update y Delete).

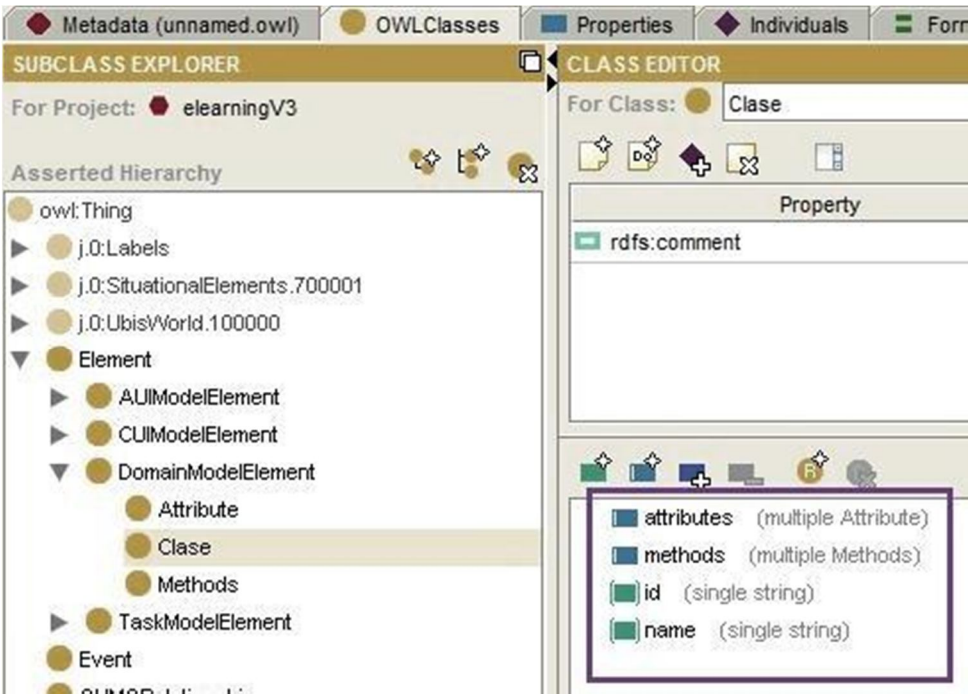


Imagen III.8. Propiedades de la Subclase *Clase* del modelo de dominio.

En la clase *Clase* se agrupan ciertas instancias de las clases *Attribute* y *Methods*.

Modelo de Tareas

El modelo de tareas, tiene las propiedades: *id*, *name*, *taskModel*, *taskItem*, *taskType*, *unaryRelationship*, *binaryRelationship*, *decomposition*. Para más detalle se puede recurrir a la documentación de UsiXML [USI07].

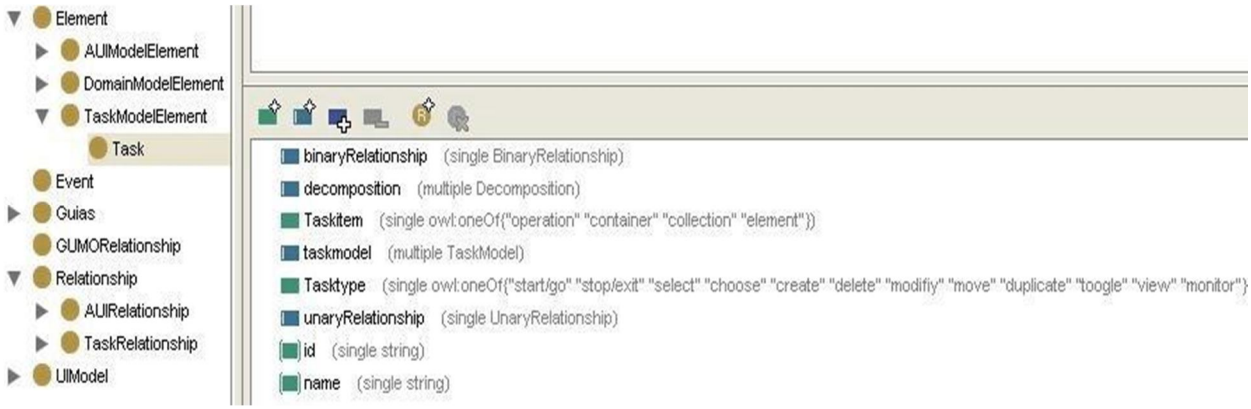


Imagen III.9. Modelo de tareas y sus propiedades.

Las tareas se refieren a las actividades que pueden realizar los usuarios con la aplicación para manipular los objetos de dominio, como por ejemplo dar de alta un libro, listar libros por autor, etc.

Modelo de Mapeo

El modelo de mapeo posee las subclases *Manipulates* e *isAllocatedTo* las cuales son utilizadas para instanciar relaciones entre tareas y los elementos del dominio, y entre los usuarios y las tareas que pueden realizar.

En primer lugar *Manipulates*, posee las propiedades *targetManipulate* y *sourceManipulate* las cuales representarían las tareas y los elementos del dominio respectivamente.

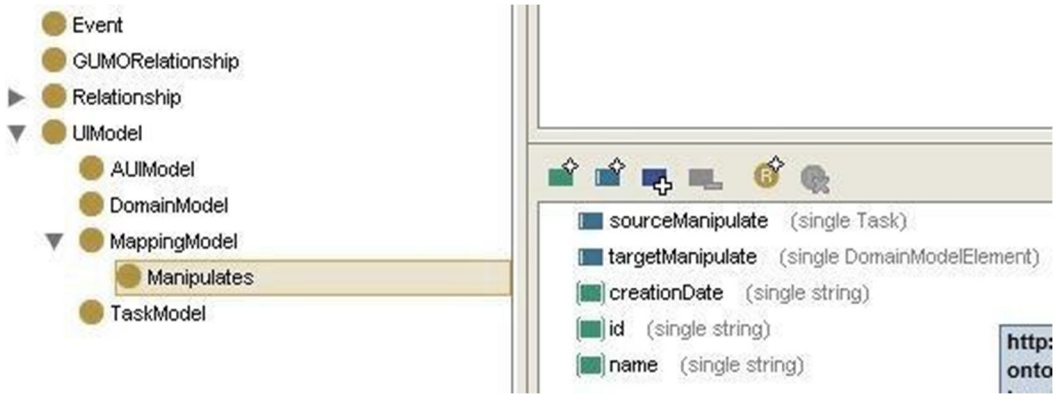


Imagen III.10. Propiedades de la clase *Manipulates* del modelo de mapeo.

Por ejemplo para la tarea *Seleccionar fecha* le correspondería el objeto de dominio *Fecha de nacimiento*.

Las principales propiedades de *isAllocatedTo* son *source* y *target* que permitirían representar por ejemplo la relación del usuario *Manuel* con la tarea *Seleccionar fecha.*

Imagen III.11. Propiedades de la clase *isAllocatedTo* del modelo de mapeo

Modelo de interfaz de usuario abstracta

Como se explicó en los marcos referenciales en el modelo de interfaz de usuario abstracta (MAUI) se describen las potenciales interfaces de usuario independiente de cualquier tecnología y sin considerar aun las características de los usuarios.

Está compuesto por dos clases principales la clase *AbstractInteractionObject* y la clase *Facet*, estas a su vez cuentan con varias subclases que son las que se instancian en la primera etapa del proceso de generación de IU.

La clase *AbstractInteractionObject* tiene dos subclases que la componen y son: *AbstractContainer* y *AbstractIndividualComponent*, haciendo una analogía con las interfaces de usuario finales estas dos clases corresponderían a ventanas y a elementos incluidos en dichas ventanas (como contenedores de botones, de campos de texto, de etiquetas, etc.).

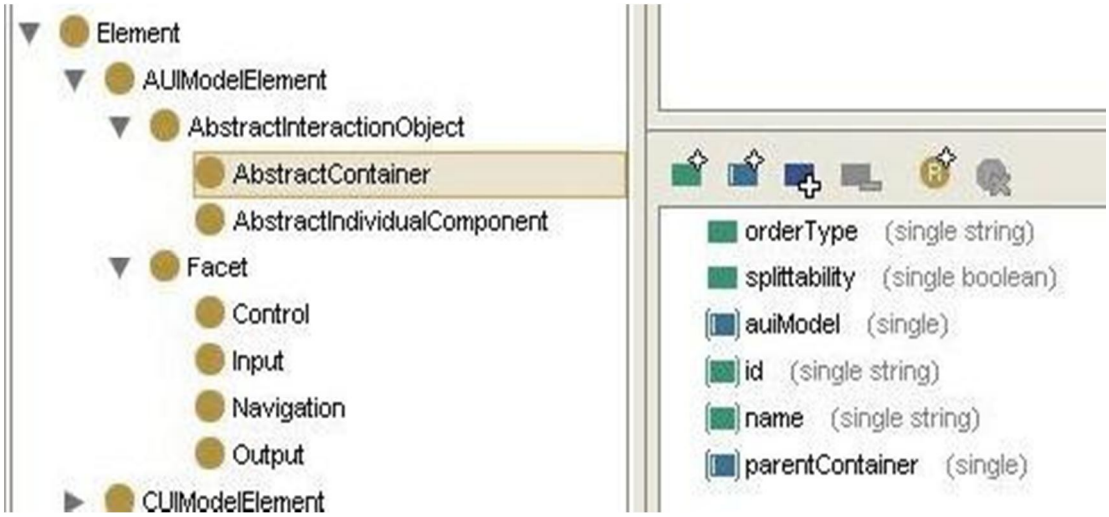


Imagen III.12. Propiedades de la clase *AbstractContainer*.

En la imagen III.12 se observan las propiedades que forman la clase *AbstractContainer*.

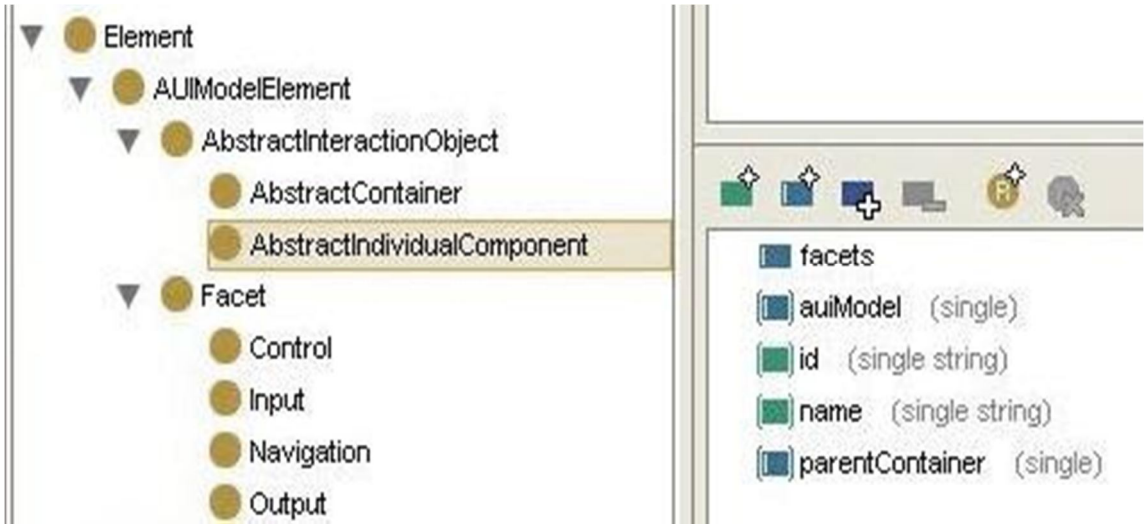


Imagen III.13. Propiedades de la Clase *AbstractIndividualComponent.*

En la imagen III.13 se pueden observar las propiedades de la clase *AbstractIndividualComponent*. Estas son: *facets* (agrupa las instancias de la clase *Facet* que pertenecen al componente abstracto individual en cuestión), *auiModel*, *id, name*, *parentContainer* (referencia al *AbstractContainer* en el que se debe colocar el componente abstracto individual)

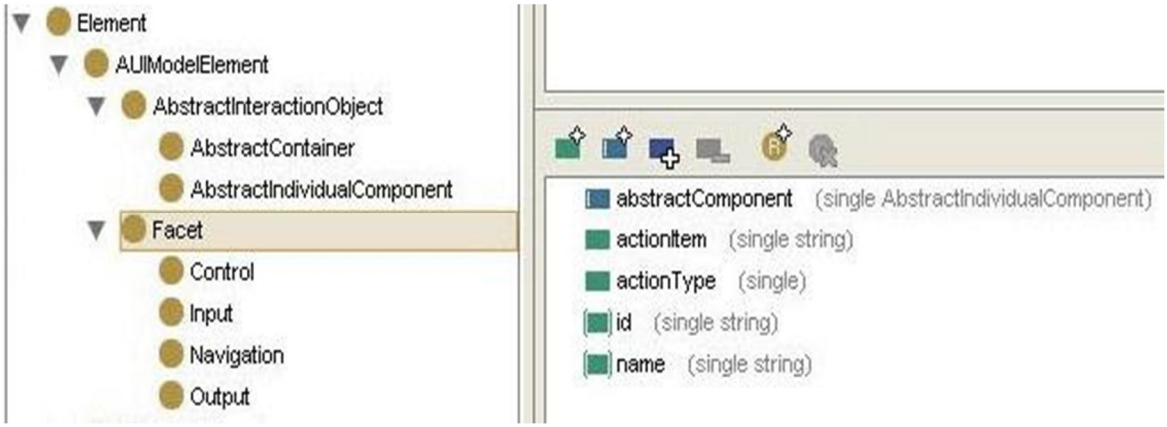


Imagen III.14. Propiedades de la clase *Facet.*

Las propiedades y subclases de la clase *Facet* se observan en la imagen III.14, las propiedades son: *abstractComponent, actionItem, actionType, id* y *name.* Y las subclases que la componen son: *Control, Input, Navigation, Output*.

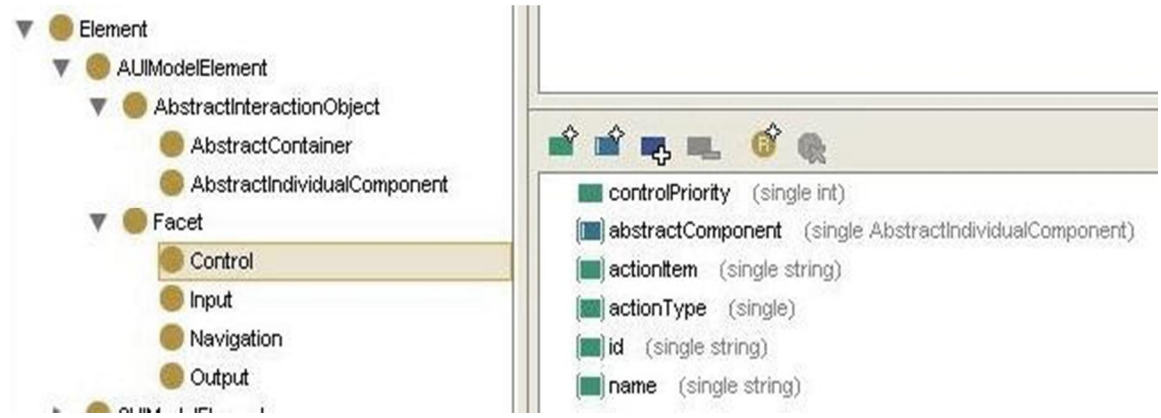


Imagen III.15. Propiedades de la clase *Control.*

En la imagen III.15 se pueden ver las propiedades de la clase *Control*, estas son: *controlPriority, abstractComponent, actionItem, actionType, id* y *name.*

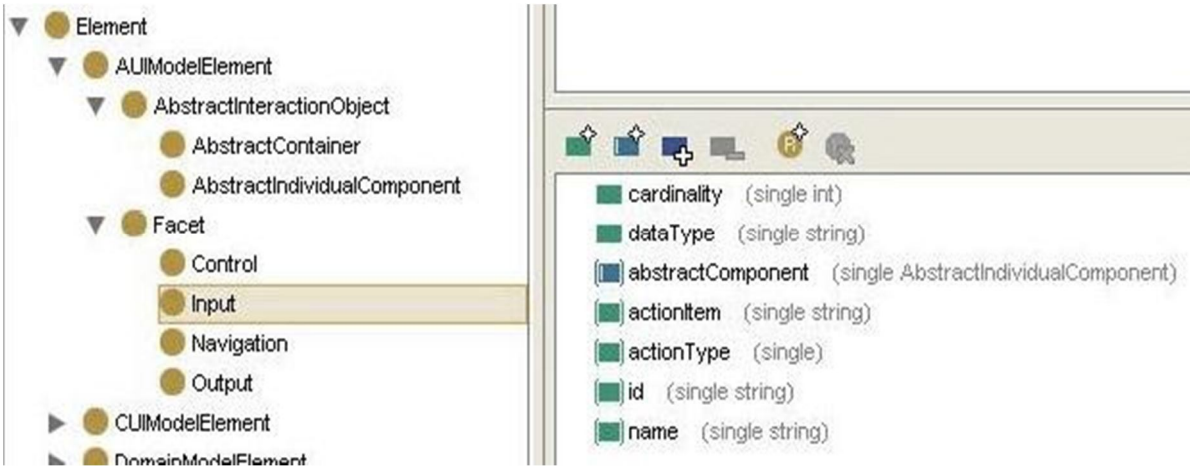


Imagen III.16. Propiedades de la clase *Input.*

Las propiedades de la clase *Input* se pueden observar en la imagen III.16, y son: *cardinality, dataType, abstractComponent, actionItem, actionType, id* y *name.*

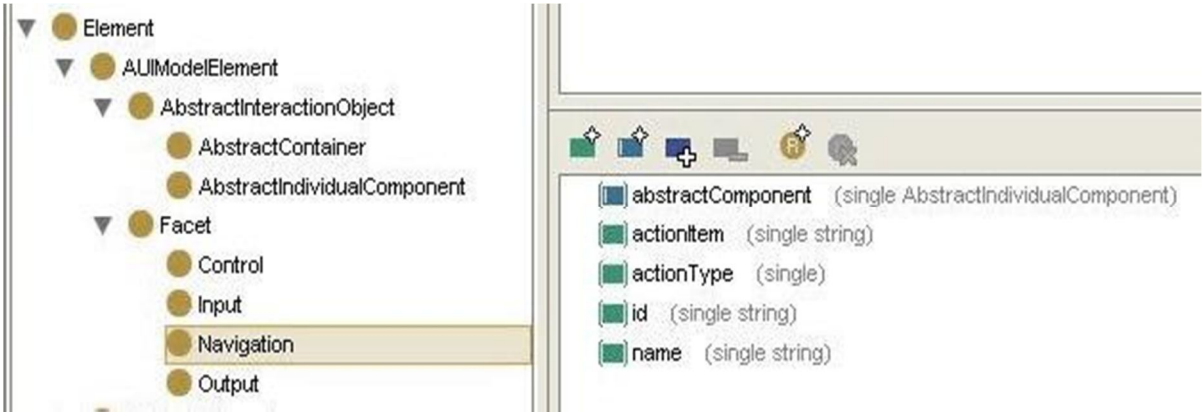


Imagen III.17. Propiedades de la clase *Navigation.*

En la imagen III.17 se pueden ver las propiedades de la clase *Navigation*, estas son: *abstractComponent, actionItem, actionType, id* y *name.*

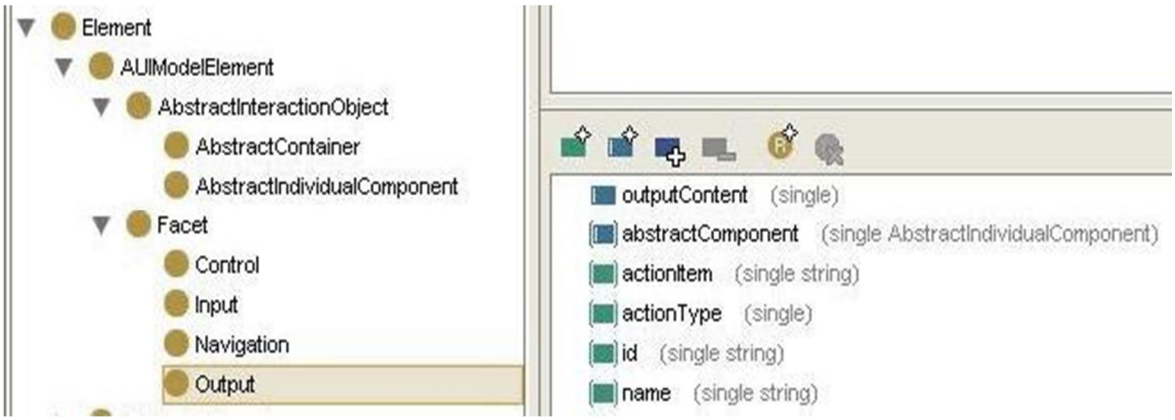


Imagen III.18. Propiedades de la clase *Output.*

En la imagen III.18 se pueden ver las propiedades de la clase *Output*, estas son: *controlPriority, abstractComponent, actionItem, actionType, id* y *name.*

Representación de una interfaz de usuario abstracta

A modo de ejemplo se muestra el siguiente gráfico empleando la notación basada en L. Constantine [CON96] para prototipos canónicos abstractos.

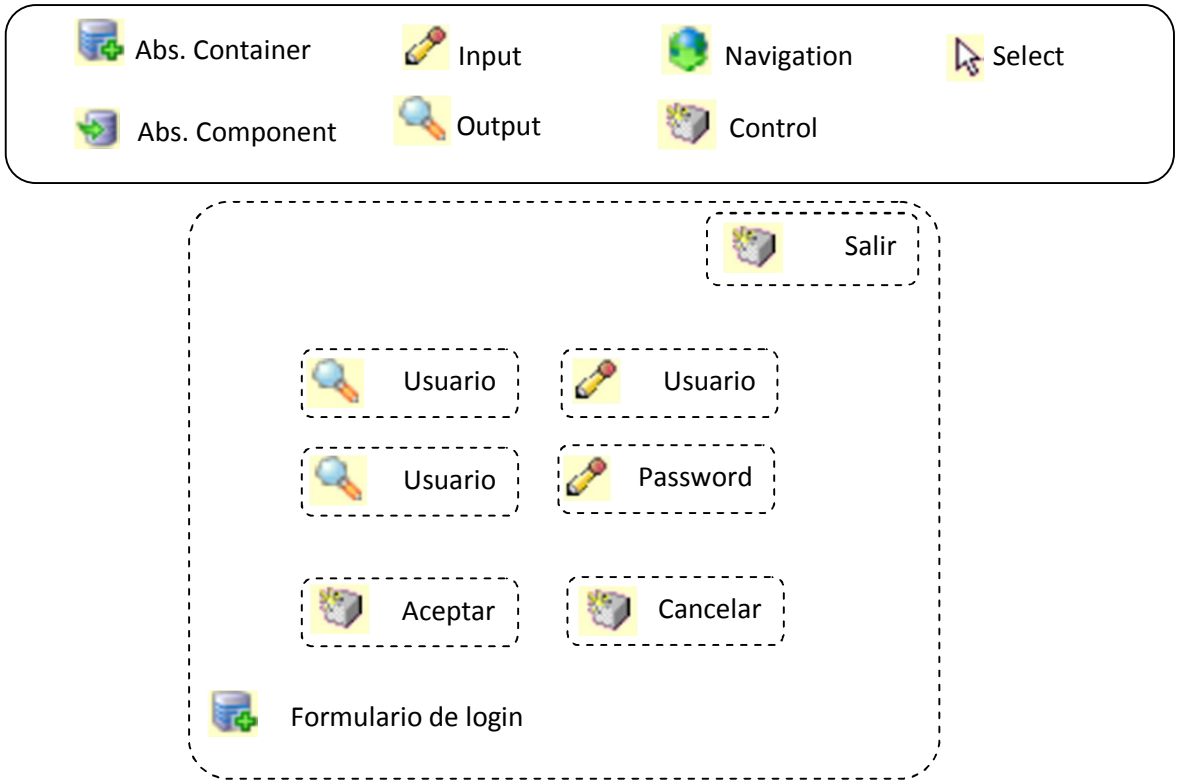


Imagen III.19. Ejemplo de interfaz de usuario abstracta.

En la imagen III.19 se observa una interfaz de una aplicación bastante común, se trata de un formulario de login en donde el usuario debe colocar su nombre de usuario y contraseña (password). Los botones Salir, Aceptar y Cancelar como elementos de tipo *Control*, los

mensajes de Usuario y Password como elementos de tipo *Output*, los campos en donde ingresar el nombre de usuario y password son elementos de tipo *Input* y por último la ventana que contiene a todo como un elemento de tipo *AbstractContainer*.

Esta se representa una interfaz de usuario abstracta, es decir, una interfaz independiente de la plataforma, tecnología, tipo de usuario, etc. Del ejemplo no se puede deducir si se trata de una interfaz para una aplicación web, de escritorio o de algún otro tipo.

Un posible resultado de la interfaz anterior puede ser el que se observa en la imagen siguiente.

Imagen III.20. Ejemplo de interfaz de usuario final.

Ontología para el modelado de usuarios (GUMO)

Se eligió GUMO, en base a la investigación y el análisis realizado previamente, porque se destaca como una ontología con proyección hacia el futuro. Posee un sitio en el cual se puede navegar por la ontología, se pueden observar ejemplos de instancias cargadas en la misma y, lo más importante, permite la contribución de los usuarios ya que los creadores impulsan el desarrollo colaborativo de las ontologías.

Además de todo esto, la ontología puede ser descargada desde el sitio, lo que no ocurre con el resto ya que se tratan principalmente de trabajos de investigación.

Se observaron los trabajos de investigación enfocados en utilizar ontologías para modelar sistemas, principalmente web. Por ejemplo OntoWeaver [LEI04] y Ontology-Based Method for Universal Design of User Interfaces [FUR07], pero en ninguno de estos trabajos existe la posibilidad de descargar y por lo tanto reutilizar las ontologías.

A continuación se explica el proceso seguido para obtener la ontología final a partir de las etapas previamente realizadas. Se ha creado un proyecto en Protégé en el formato OWL. Se ha exportado la ontología de modelos con formalismo UsiXML, del formato RDF al OWL. Luego se ha importado GUMO sin ningún inconveniente ya que está desarrollada en OWL. Finalmente se han creado las clases como se puede ver en la etapa ***Diseño de Ontología***.

III.3.4. PROGRAMACIÓN

Como se observó previamente, existe un modelo de Mapeo, el cual sirve para establecer las relaciones entre los elementos de los demás modelos. Quizás la tarea más difícil de llevar a cabo, es la de decidir las relaciones entre los elementos de los modelos. Estas relaciones se dan entre todos los modelos, tanto entre el modelo de tareas y el de dominio, como así también entre estos últimos y el modelo abstracto. La diferencia está en que, por ejemplo, el modelo de tareas y el de dominio se encuentran en el mismo nivel, mientras que el modelo abstracto se encuentra en un nivel más bajo, figura III.9.

Figura III.9. Niveles de los modelos.

Para pasar progresivamente desde el nivel más alto a un nivel más bajo existe un sistema de transformación que sugiere que cada nivel puede ser alcanzado a través de una serie de transformaciones. Una transformación aplica reglas (de transformación) a modelos iniciales para tener como resultado otros modelos. Para especificar tales transformaciones, UsiXML consta de una estructura subyacente de grafos, gracias a la cual cualquier regla de transformación de grafos puede ser aplicada a una especificación UsiXML.

Los sistemas de transformación se basan en la teoría de gramáticas de grafos [ROZ97]. Se explican los beneficios de las transformaciones de grafos en Program Transformation Systems [PAR83].

En UsiXML, el enfoque de transformación está apoyado por el entorno TransformiXML [LIM04], el cual permite la expresión de las relaciones entre los modelos y la definición y aplicación de las reglas de transformación.

Por el momento no se encuentra una versión publica disponible, por lo cual, en este trabajo no se utilizará el modelo de transformación de UsiXML. Se decidió crear las reglas de transformación utilizando el plug-in para Protégé, JessTab.

Se realizarán las transformaciones desde el modelo de tareas y dominio al modelo abstracto y de este último y el modelo de usuarios al modelo concreto.

Se utilizaron reglas tomadas de [FUR04] para generar las transformaciones al modelo abstracto y para generar el modelo concreto se utilizó el criterio propio tomando en cuenta solamente las características de los usuarios.

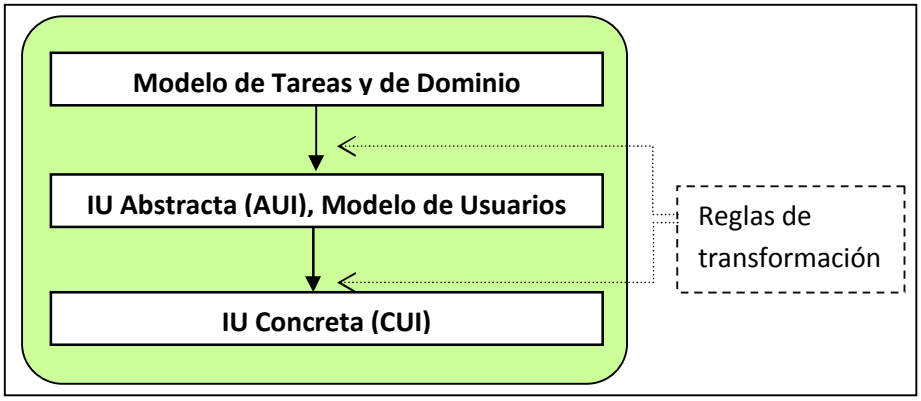


Figura III.10: Proceso de Transformación.

Primero se debe crear el conjunto de reglas necesarias para producir el modelo de interfaz de usuario abstracta, imagen III.21.

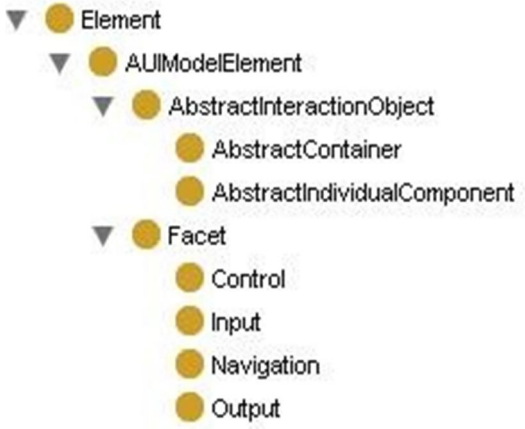


Imagen III.21. Modelo de interfaz de usuario abstracta.

Comenzando con los elementos *AbstractContainer* y *AbstractIndividualComponent* para seguir con los *Facets* que incluyen los elementos *Output*, *Input*, *Navigation* y *Control*.

Se usan las siguientes convenciones para describir las reglas desarrolladas.

R	Relación
T	tarea
^	operador lógico *y*
v	operador lógico *o*
=>	operador lógico *entonces*

En la tabla III.1 se enumeran las reglas creadas para el prototipo con una breve explicación. A continuación de esta tabla se detallarán las reglas desde la número 1 a la 8, correspondientes a la creación de los *AbstractContainer*, *AbstractIndividualComponent* y *Output*. En el Anexo se detallarán las reglas restantes.

Tabla III.1. Reglas de Transformación del prototipo.

Reglas para generar Modelo Abstracto		
Número	**Nombre**	**Uso**
	AbstractContainer y AbstractIndividualComponent	
1	RelacionesBinarias	En primer lugar se recorren las relaciones binarias entre tareas, de tipo *Enabling* y *Choice*.
2	NombreRelacionUnaria	De las tareas que cumplen la condición anterior, se obtiene el nombre de las tareas que poseen una relación unaria.
3	crearContenedorAbstracto	Si la relación unaria de la tarea fuente es de la clase *FiniteIteration* y la tarea objetivo de la clase *Optional* se genera un *AbstractContainer*.
4	eliminarContenedorRepetido	Se eliminan *AbstractContainer* repetidos
5	crearComponenteIndividualAbstracto	Para cada *AbstractContainer* generado se crea una instancia de *AbstractIndividualComponent*.
	Output	
6	AUIFacetTareaDominio	Se obtienen las relaciones entre las tareas y los elementos del dominio.
7	Atributos	De las relaciones anteriores se obtienen los elementos de dominio que pertenecen a la clase *Clase*.
8	CrearOutput	Se crean los *Outupt*.
	Input	
9	AttrTareaTipo	Se toman las tareas que posean las propiedades *Tasktype=select* y *Taskitem=element*. Esto nos indica que se trata de una tarea que solicita el ingreso de un elemento del modelo de dominio.
10	Aux1	Se obtiene el nombre de la tarea mencionada anteriormente.
11	CrearInput	Se crean finalmente los *Input*.
12	EliminarOutput	En el momento de crear los *Output* se lo hizo para todos los atributos. Algunos de ellos en realidad deberían ser *Input*, por eso ahora se los elimina.
	Navigation	
13	TareasEnabling	Se obtienen las tareas que posean la relación *Enabling*.
14	NombreTareasEnabling	Se obtienen los nombres de las tareas anteriores.
15	CrearNavigation	Se crean los *Navigation*.
16	eliminarNavigationRepetido	Se eliminan elementos *Navigation* repetidos.
	Control	
17	Metodos	De las relaciones entre tareas y dominio, se obtienen los elementos del dominio pertenecientes a la clase *Methods*.
18	CrearControl	Se crean los elementos de tipo *Control*.

Reglas para generar Modelo Concreto		
Número	**Nombre**	**Uso**
19	UsuariosTareas	Se obtienen las relaciones de los usuarios y sus respectivas tareas.
20	UsuariosGuias	Se obtienen los elementos concretos de las guías de usabilidad que le corresponden a cada uno de los usuarios.
21	UsuariosGuiasGenerales	Elementos de guías que se aplican a todos los usuarios por igual.
22	UsuariosElementosGuias_1	Comienza el proceso para obtener más detalles de los elementos de las guías de diseño que corresponden a cada usuario.
23	UsuariosElementosGuias_2	Segundo paso del proceso de obtención de detalles de los elementos.
24	UsuariosElementosGuias_3	Tercer paso del proceso de obtención de detalles de los elementos.
25	UsuariosElementosGuias_4	Finaliza el proceso de especificación de los elementos y sus detalles para cada usuario.
26	UsuariosTareasComponent	Se define un arreglo que contendrá la relación entre los usuarios, las tareas y los componentes.
27	ComponentesAbstractos	Obtiene un *AbstractIndividualComponent* que le corresponde a cada elemento concreto del arreglo creado anteriormente.
28	Ventanas_x_guias	Genera ventanas especificadas en lenguaje XML para UsiXML, con sus atributos determinados por las guías de diseño.
29	resto_Ventanas	Genera ventanas especificadas en lenguaje XML para UsiXML, con sus atributos por defecto.
30	Button_x_guias	Genera botones especificados en lenguaje XML para UsiXML, con sus atributos determinados por las guías de diseño.
31	resto_Button	Genera botones especificados en lenguaje XML para UsiXML, con sus atributos por defecto.
32	Datepicker_x_guias	Genera calendarios especificados en lenguaje XML para UsiXML, con sus atributos determinados por las guías de diseño.
33	resto_Datepicker	Genera calendarios especificados en lenguaje XML para UsiXML, con sus atributos por defecto.
34	Listbox_x_guias	Genera listados de opciones especificados en lenguaje XML para UsiXML, con sus atributos determinados por las guías de diseño.
35	resto_Listbox	Genera listados de opciones especificados en lenguaje XML para UsiXML, con sus atributos por defecto.
36	Inputtext_x_guias	Genera campos de texto especificados en lenguaje XML para UsiXML, con sus atributos determinados por las guías de diseño.
37	resto_Inputtext	Genera campos de texto especificados en

		lenguaje XML para UsiXML, con sus atributos por defecto.
38	link_x_guias	Genera hipervínculos especificados en lenguaje XML para UsiXML, con sus atributos determinados por las guías de diseño.
39	resto_link	Genera hipervínculos especificados en lenguaje XML para UsiXML, con sus atributos por defecto.
40	Outputtext_x_guias	Genera etiquetas especificadas en lenguaje XML para UsiXML, con sus atributos determinados por las guías de diseño.
41	resto_Outputtext	Genera etiquetas especificadas en lenguaje XML para UsiXML, con sus atributos por defecto.
42	Timesensor_x_guias	Genera cronómetros especificados en lenguaje XML para UsiXML, con sus atributos determinados por las guías de diseño.
43	resto_Timesensor	Genera cronómetros especificados en lenguaje XML para UsiXML, con sus atributos por defecto.

AbstractContainer y AbstractIndividualComponent

Para crear los *AbstractContainer* se aplica la siguiente regla:

Si dos tareas (T1 y T2 pertenecientes a *TaskModel*) están relacionadas mediante un tipo de relación binaria de habilitación (*Enabling*) o de selección (*Choice*), y la relación unaria de una tarea (T1) es "Iteración finita" (*FiniteIteration*) y la relación unaria de la otra tarea relacionada (T2) es "opcional" (*Optional*) entonces crear un "Contenedor Abstracto" (*AbstractContainer*).

Si T1 R T2 ^ (R = Enabling **v** R = Choice) ^ T1 R T1^ (R = finiteIteration)^ T2 R T2 ^(R = Optional) => instanciar clase *AbstractContainer*

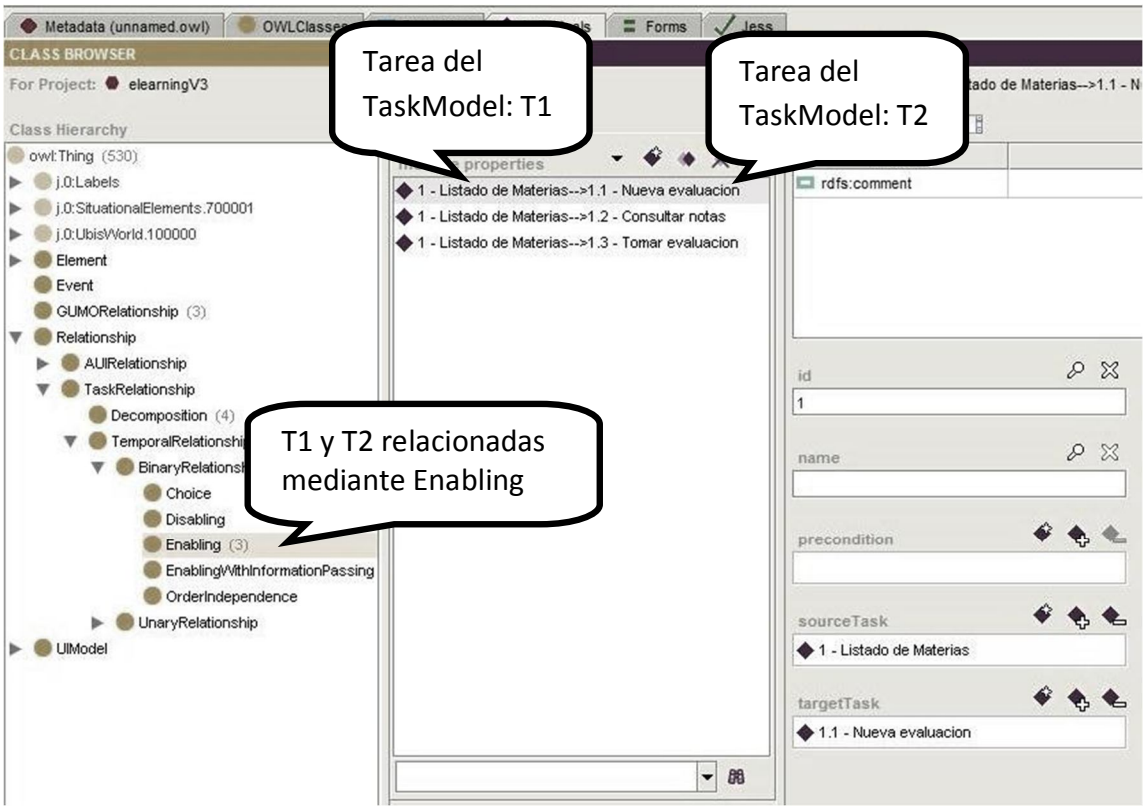


Imagen III.22. Instancias de las relaciones binarias *Enabling*.

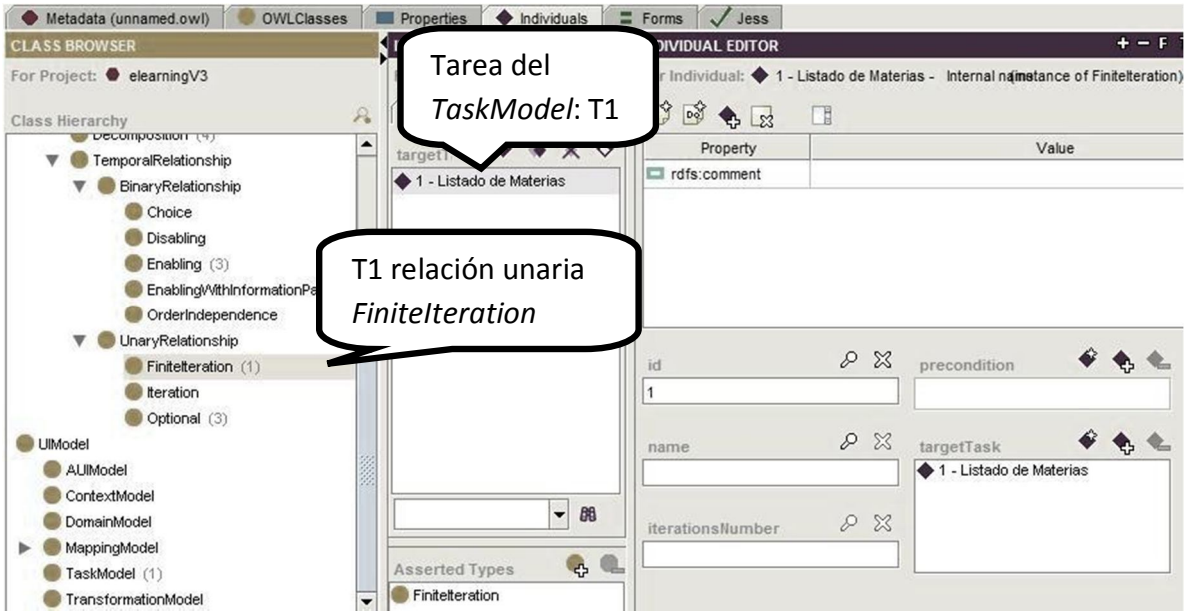


Imagen III.23. Instancias de las relaciones unarias *FiniteIteration*.

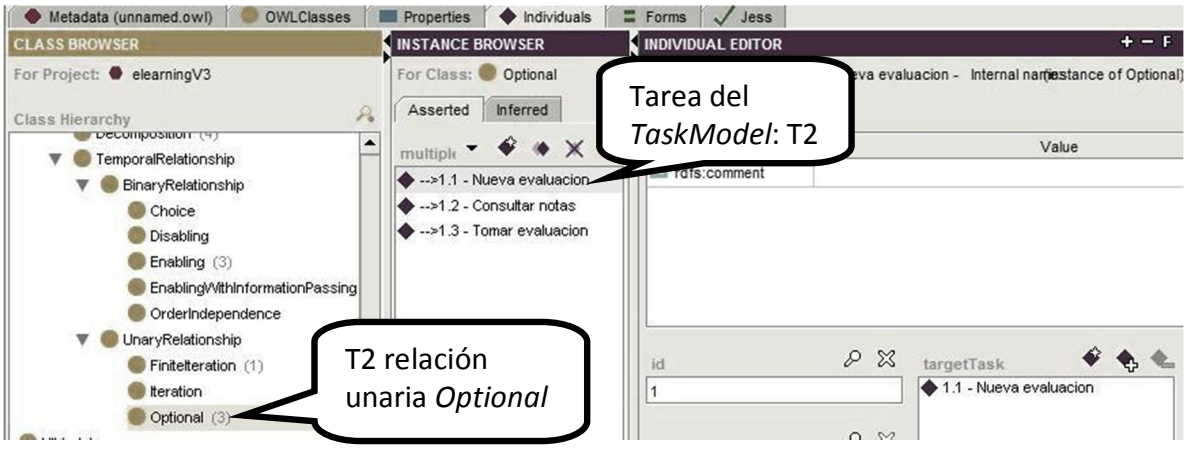


Imagen III.24. Instancias de las relaciones unarias *Optional*.

En primer lugar se recorren las relaciones binarias de tipo *Enabling* y *Choice*. De cada una de ellas se toman los atributos *id*, *sourceTask* y *targetTask*. Los últimos dos atributos son referencias a instancias del modelo de tareas. De las tareas a las cuales hacen referencia se obtiene el nombre y su relación unaria en un hecho (*fact*) que equivaldría a un array de información.

```
(defruleRelacionesBinarias;Nombre de la regla
(object
  (is-aEnabling|Choice);Relaciones de tipo Enabling o Choice
  (sourceTask ?sourct)
  (targetTask ?targt)
  (id ?id))
=>
(assert (RelacionBinaria; Se crea fact con nombre RelacionBinaria
(slot-get ?sourct name); Nombre tarea fuente
 (slot-get ?sourct unaryRelationship); Relación unaria de tarea fuente
 (slot-get ?targt name) ;Nombre tarea objetivo
(slot-get ?targt unaryRelationship);Relación unaria de tarea objetivo
  ?id ))) ;Id de relación binaria
```

Se obtiene el nombre de la clase a la cual pertenece la relación unaria de las tareas del *fact* anterior.

```
(defruleNombreRelacionUnaria
 (RelacionBinaria; Nombre del fact creado en la regla anterior
   ?nombrest
   ?runariast
   ?nombretargt
```

```
      ?runariatargt
      ?id)
        =>
    (assert (RelacionesUnarias ;Nombre del fact a crear
    ?nombrest
    (class ?runariast) ;Clase de la relación unaria de la tarea fuente
     ?nombretargt
   (class ?runariatargt) ;Clase de la relación unaria de la tarea objetivo
    ?id)))
```

En esta regla se comparan los nombres de las clases obtenidas en el paso anterior, si la relación unaria de la tarea fuente es de la clase *FiniteIteration* y la tarea objetivo de la clase *Optional* se debe generar un *AbstractContainer*.

```
   (defrule crearContenedorAbstracto
     (RelacionesUnarias
      ?nombrest
      ?runariast&:(= ?runariast FiniteIteration) ;Pregunta si la relación unaria de la tarea fuente es de la clase FiniteIteration
      ?nombretargt
      ?runariatargt&:(= ?runariatargt Optional) ;Pregunta si la tarea objetivo de la clase Optional
      ?id)
      =>
   ;Se crea instancia de clase AbstractContainer
   (make-instanceof AbstractContainer
     (name ?nombretargt)
     (id ?id)))
   (make-instanceof AbstractContainer
     (name ?nombrest)
     (id ?id)))
   (defrule eliminarContenedorRepetido  (object    (is-a AbstractContainer)  (OBJECT ?objAbs1)  (name ?nomA)) (object    (is-a AbstractContainer) (OBJECT ?objAbs2&~?objAbs1)  (name ?nomA))  =>  (unmake-instance ?objAbs2))
```

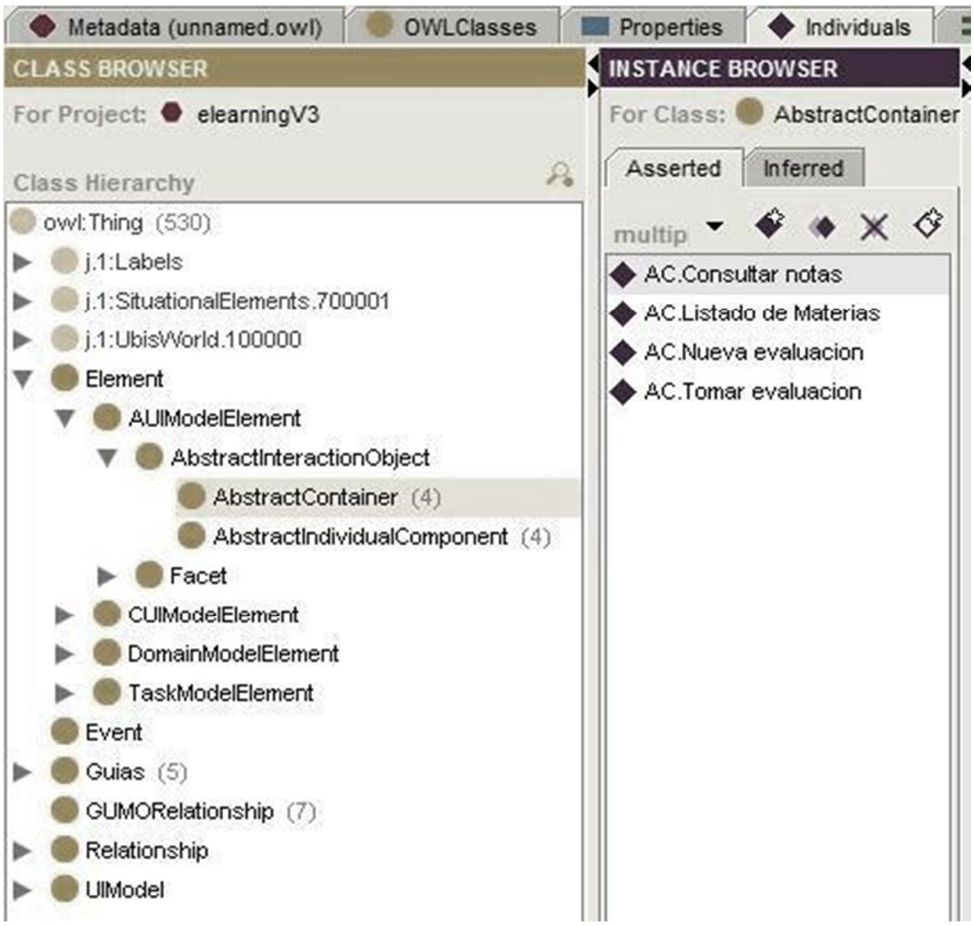


Imagen III.25. Instancias de la clase *AbstractContainer.*

Se generan ahora cada uno de los *AbstractIndividualComponent*. En este caso, para cada una de las instancias de *AbstractContainer*se crea una instancia de *AbstractIndividualComponent* con la propiedad *name* y *parentContainer* como se observa en la imagen III.26.

```
(defrulecrearComponenteIndividualAbstracto
  (object
    (is-aAbstractContainer)
    (OBJECT ?obj)
    (name ?nombreAC))
     =>
 (make-instance ofAbstractIndividualComponent
(name ?nombreAC)
  (parentContainer ?obj)))
```

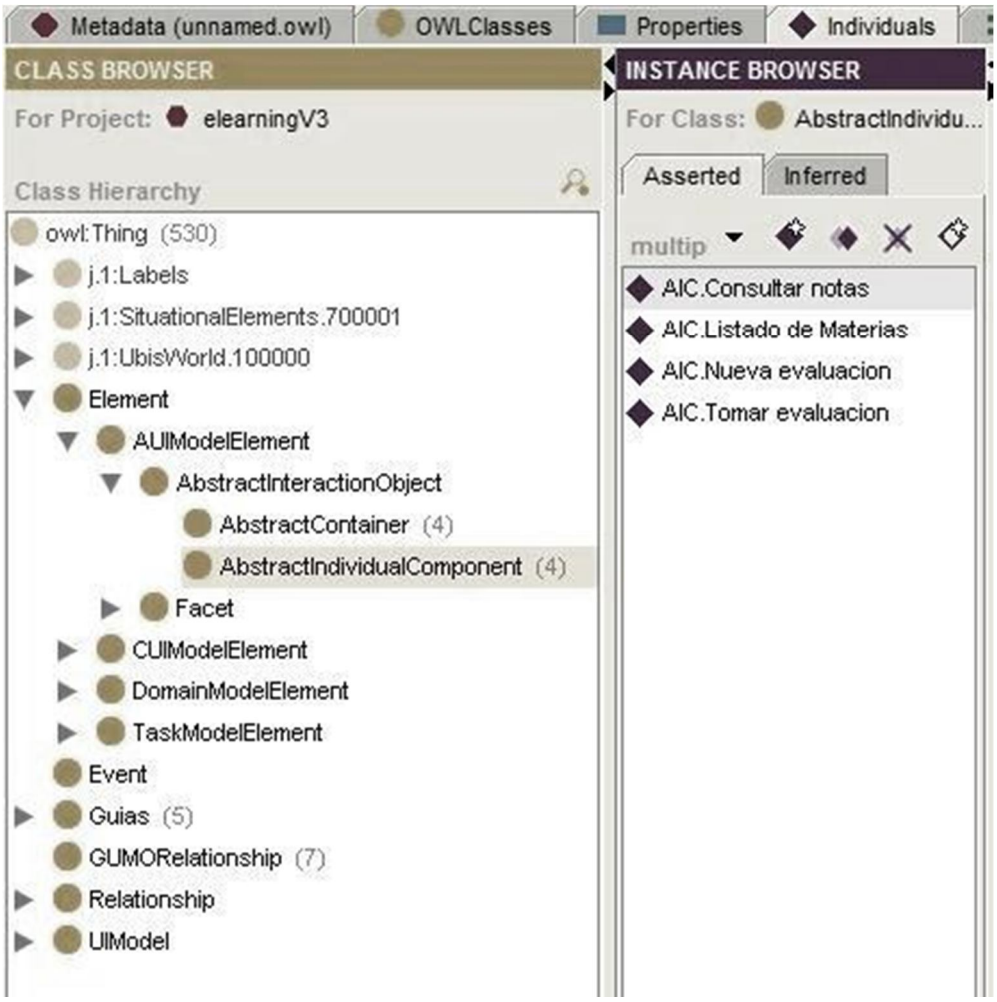


Imagen III.26. Instancias de la clase *AbstractIndividualComponent.*

A continuación se muestran las reglas creadas para poder generar las instancias de cada una de los *Facets* del modelo AUI.

Output

Se crean todos los *Output*. Se recorre la clase *Manipulates* en busca de las relaciones entre las tareas y los elementos del dominio como se observa en imagen III.27.

Para crear los *Output* se aplica la siguiente regla:

Si una tarea (T1 perteneciente a *TaskModel*) está relacionada con una *Clase* del modelo de dominio (C1) entonces crear un "Elemento de salida" (*Output*) para todos los atributos de dicha *Clase*.

Si T1 R C1 ^ C1 = *Clase* => instanciar clase *Output*

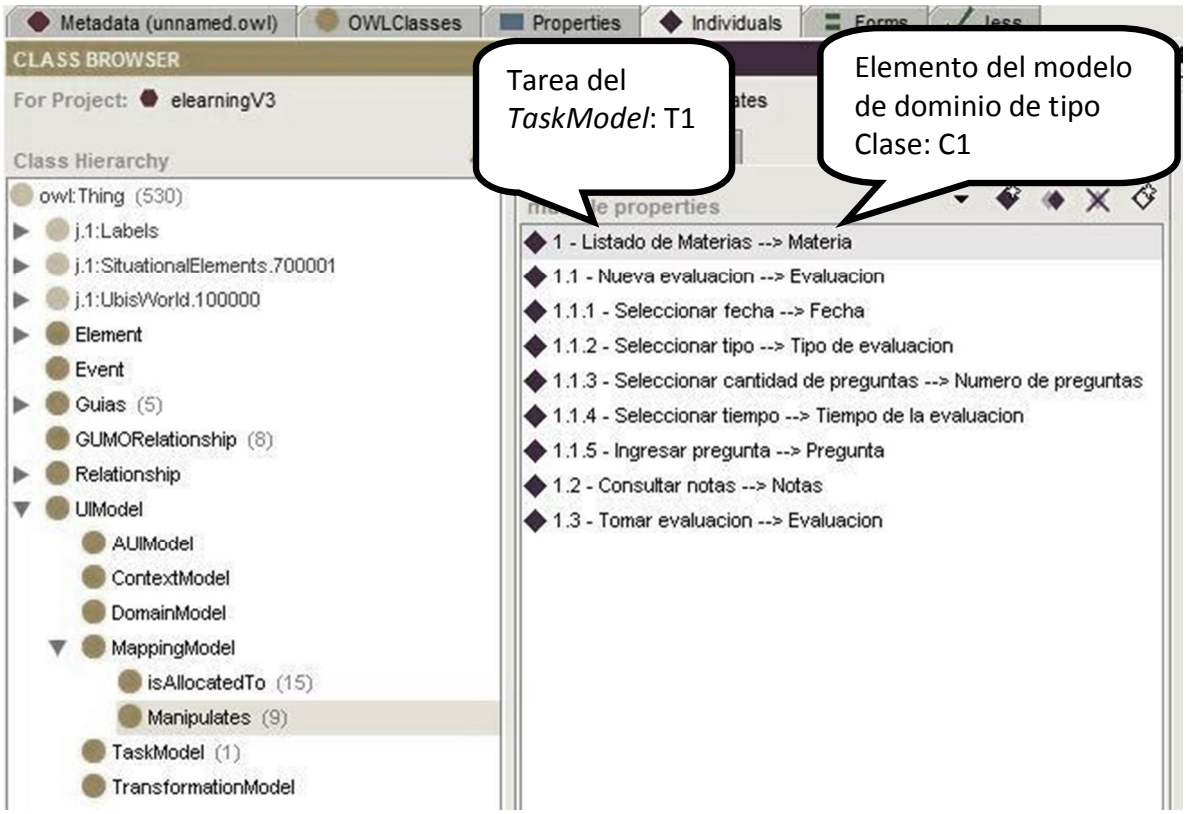


Imagen III.27. Instancias de la clase *Manipulates.*

Se obtiene de cada una de las relaciones el atributo *sourceManipulate* que toman una instancia de la clase *Tareas* y el atributo *targetManipulate* que pertenece al modelo de dominio, específicamente a las clases *Clase* o *Attributes* que están incluidas en el mismo.

```
(defrule AUIFacetTareaDominio
  (object
   (is-a Manipulates)
(sourceManipulate ?smanip)
   (targetManipulate ?tmanip))
    =>
;Se guarda en el fact el nombre de la tarea y el elemento del modelo de dominio
(assert (TareDominio
  (slot-get ?smanip name)
  ?tmanip)))
```

Luego se determina a que clase del modelo de dominio pertenece el elemento *targetManipulate* que se encuentra en el *factTareaDominio* y en la variable *?tmanip*.

```
   (defrule Atributos
    (TareDominio
      ?smanip ;El nombre de la tarea
      ?tmanip&:(= (class ?tmanip) Clase)) ;Si es de tipo "Clase"
       =>
```

```
;Obtenemos los atributos de la clase
(assert
 (TareAtrb
  ?smanip
  (slot-get ?tmanip attributes))))
```

Finalmente se crean los *Output* recorriendo el *fact* creado anteriormente. Para cada uno de los elementos del mismo se obtienen instancias de la clase *AbstractIndividualComponent* y la clase *Task* que estén relacionadas con la variable *?smanip*.

```
(defruleCrearOutput
 (TareAtrb
  ?smanip ;El nombre de la tarea
  $?attrb)     ;El simbolo $ hace referencia a un array
;Se obtiene la instancia de la clase AbstractIndividualComponenty Task que posea el
mismo nombre en ?smanip, obtenido del factTareAtrb
(object
(is-a AbstractIndividualComponent)
  (OBJECT ?obj)
  (name ?smanip))
(object
(is-a Task)
   (name ?smanip)
   (Taskitem ?titem)
   (Tasktype ?ttype))
=>
;Para cada uno de los atributos en $?tmanip se crea una instancia de Output
(foreach ?variable $?attrb
 (make-instanceof Output
  (name (slot-get ?variable  name))
(abstractComponent ?obj)
  (actionItem ?titem)    ;?titem obtenida de recorrer la clase Task
  (actionType ?ttype)))) ;?ttype obtenida de recorrer la clase Task
```

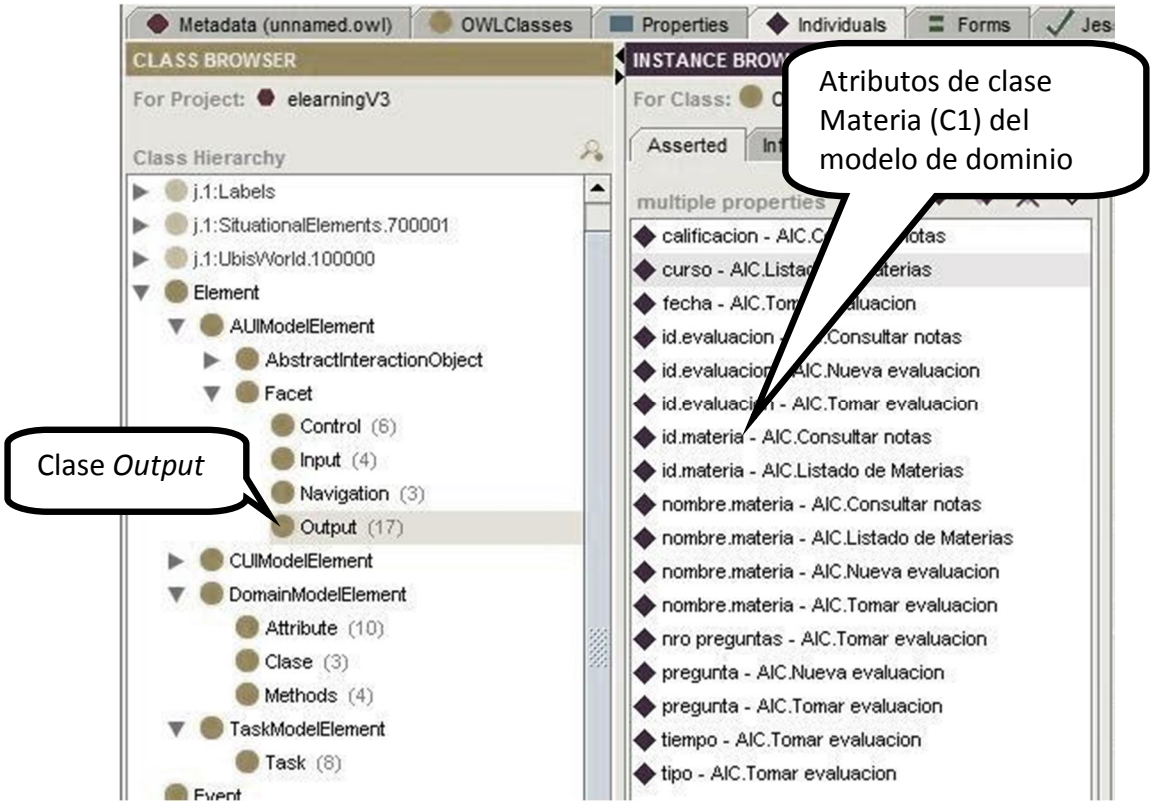


Imagen III.28. Instancias de la clase Output.

III.3.5. GUÍAS METODOLÓGICAS PARA EL PROCESO DE DISEÑO DE LA IU

El usuario deberá seguir un conjunto de etapas para poder obtener las IU deseadas. A continuación se detallan las etapas que forman parte del proceso de diseño de las IU.

1. INSTANCIAR LA ONTOLOGÍA DE MODELOS: para comenzar el proceso de diseño de las IU el desarrollador o diseñador, usuario del prototipo, deberá volcar todos sus conocimientos sobre el sistema a desarrollar y los detalles de los usuarios de dicho sistema en la ontología de modelos. Para ello cuenta con cuatro modelos en los que deberá ingresar tales conocimientos, esta tarea se denomina instanciación. Dichos modelos se listan a continuación:
    a. Modelo De Dominio
    b. Modelo De Tareas
    c. Modelo De Usuario
    d. Modelo De Mapeo
2. EJECUTARLAS REGLAS DE TRANSFORMACIÓN: Luego de la instanciación de la ontología el desarrollador o diseñador deberá ejecutar las reglas que generarán los siguientes modelos:
    a. Modelo Abstracto
    b. Modelo de Interfaces Concretas

Este proceso se desarrolla con detalle en el anexo C: Manual de usuario.

III.4. PRUEBAS/MODIFICACIÓN

III.4.1. PLANIFICACIÓN DE PRUEBAS

Para realizar las pruebas sobre el prototipo se tendrán en cuenta las siguientes características: completitud, correctitud y usabilidad.

- Completitud: se utilizarán las variables: usuarios, tareas y dominio.
  - Usuarios: se buscará la aceptación de los diferentes indicadores que puede contemplar, estos son: roles, características físicas, cognitivas y demográficas.
  - Tareas y dominio: se buscará la aceptación de las tareas y de los elementos del dominio respectivamente.
- Correctitud: se diseñarán y ejecutarán casos de prueba, para conocer el grado de concordancia de las interfaces finales con los modelos de usuario, de tareas y de dominio.
- Usabilidad: se crearán y utilizarán listas de verificación que permitan determinar si cumple con las características de aprendibilidad y extensibilidad.

Capítulo IV
Validación del Prototipo

CAPÍTULO IV

VALIDACIÓN DEL PROTOTIPO

IV.1. INTRODUCCIÓN

En este capítulo final se describen los detalles de la generación de múltiples IU concretas de acuerdo a las preferencias de los usuarios.

Como se mencionó anteriormente en este trabajo, el diseño de interfaces con el prototipo propuesto seguirá el proceso de desarrollo de la ingeniería de la usabilidad de *The usability professionals' association* [THE09]. Al utilizar el UIDL UsiXML, el producto de las diferentes actividades serán los modelos del mismo.

Se observa nuevamente en la figura IV.1 el proceso de desarrollo de la ingeniería de la usabilidad.

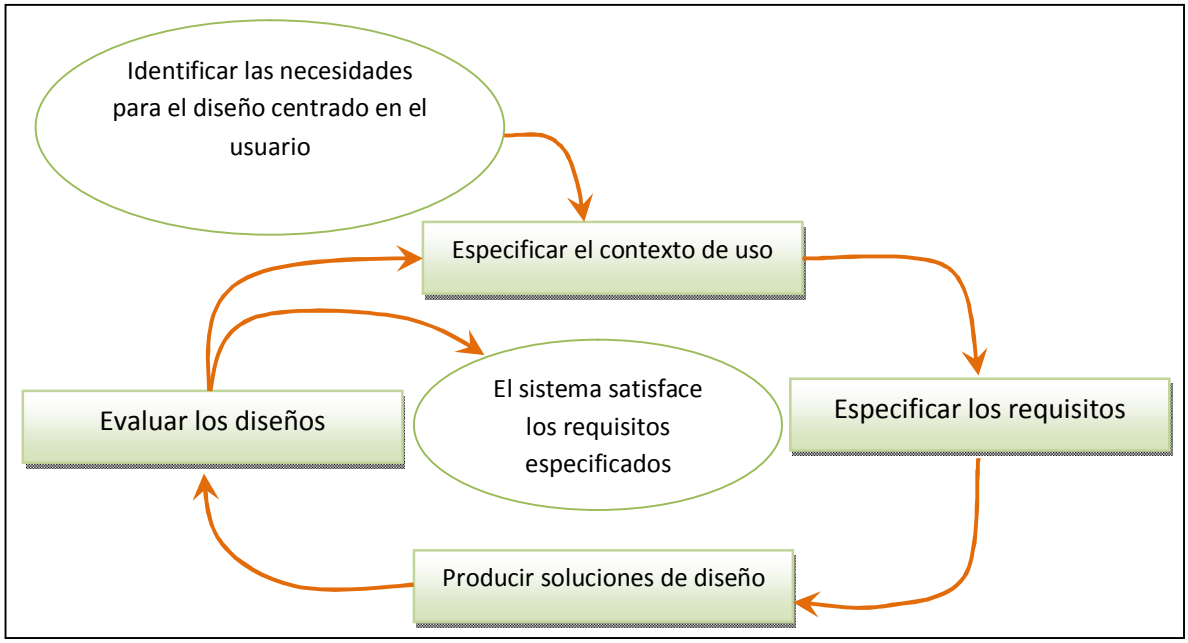


Figura IV.1. Proceso de diseño centrado en el usuario

Al ser la ingeniería de usabilidad un proceso de diseño centrado en el usuario, la elicitación de requisitos se encuentra, por un lado, relacionado con las preferencias de los usuarios que utilizarán el sistema y por otro lado relacionado con el perfil de los mismos. En las pruebas el enfoque estará en el perfil de cada usuario que utilizará el sistema.

Para realizar la prueba del prototipo se tomará como ejemplo el dominio de una aplicación web de e-learning. A continuación se seguirán los pasos del proceso de desarrollo y se irán identificando los elementos de los modelos.

IV.2. ESPECIFICAR CONTEXTO DE USO

Se comienza con la primera actividad que es la identificación de los usuarios. Se definirán usuarios que serán de diferentes tipos y poseerán diferentes características. Luego de identificarlos y caracterizarlos se procederá a volcar esta información en el modelo de usuarios.

IV.2.1. IDENTIFICACIÓN DE USUARIOS

Para el ejemplo del sistema de e-learning se utilizarán cuatro usuarios cuyos nombres son los siguientes:

- → Manuel
- → Julieta
- → Jorge
- → Pedro

IV.2.2. CARACTERIZACIÓN DE USUARIOS

Se identifican las características que poseen los usuarios mencionados anteriormente:

Tabla IV.1. Caracterización de los usuarios.

Características	**Valores**			
Nombre	Manuel	Julieta	Jorge	Pedro
Edad	12	12	38	56
Nivel cognitivo	Bajo			
Zurdo			Si	
Nivel visual		bajo		bajo
Rol	Estudiante	Estudiante	Padre	Profesor

IV.2.3. DEFINICIÓN DE LOS MODELOS CONCEPTUALES

Para la definición del modelo de usuario se utilizan las clases de la ontología GUMO. Esta ontología posee una gran cantidad de clases para caracterizar a los usuarios. Solamente se utilizarán las clases necesarias para instanciar los valores de la tabla anterior.

IV.2.3.1. Modelo de Usuarios

Se observa en la tabla IV.2 el nombre de las clases de GUMO utilizadas.

Tabla IV.2. Modelo de usuarios.

Clases GUMO	Modelo de usuario			
Person	Manuel	Julieta	Jorge	Pedro
Age	12	12	38	56
CognitiveLoad	Bajo			
LeftHanded			Si	
AbilityToSee		bajo		bajo
Rol	Estudiante	Estudiante	Padre	Profesor

En la tabla IV.2 se resume lo que sería el modelo de usuario, simplemente los valores de todos los usuarios para cada una de las clases. Se mostrará más adelante la ubicación de las clases que permiten contener esta información dentro de GUMO.

IV.3. ESPECIFICAR REQUISITOS

De esta fase se obtienen los modelos de tareas, de dominio y de mapeo. Como se consideró que es más importante destacar el funcionamiento del prototipo al momento de interpretar las características del usuario, no se especifican requisitos particulares de los usuarios sobre las interfaces. Los requisitos que se consideran son los siguientes:

1. El alumno debe poder tomar una nueva evaluación.
2. El alumno debe poder consultar las notas de las evaluaciones.
3. El padre del alumno debe poder consultar las notas.
4. El profesor debe poder crear una nueva evaluación.

IV.3.1. DEFINICIÓN DE MODELOS CONCEPTUALES

Se instanciarán las clases que se corresponden con el modelo de tareas, de dominio y de mapeo.

IV.3.1.1. Modelo de Tareas

En este caso, se crean las instancias en la clase Task que pertenece a la clase *TaskModelElement* de la ontología de UsiXML. Se puede observar en la tabla IV.3 las propiedades de la clase y los valores que toma cada una de las instancias.

Tabla IV.3. Instanciación de la clase *Task*.

Propiedades	Instancias				
Id	1	1.1	1.1.1	1.1.2	1.1.2.1
Name	Ver listado de cursos	Ver curso	Consultar notas	Nueva evaluación	Seleccionar fecha
Tasktype	View	start/go	view	start/go	select
Taskitem	Collection	collection	collection	collection	element
Taskmodel	E learning	E learning	E learning	E learning	E learning
unaryRelationship	Iteration → 1 - Ver listado de Cursos	Optional → 1.1 - Ver curso	Optional → 1.1.1 - Consultar notas	Optional → 1.1.2 - Nueva evaluacion	
binaryRelationship			Enabling → 1.1.1 - Consultar notas → 1 - Ver curso	Enabling → 1.1.2 - Nueva evaluación → 1 - Ver curso	
Decomposition					→ 1.1.2 - Nueva evaluacion

Modelo de Tareas			
Id	1.1.2.2	1.1.2.3	1.1.3
Name	Seleccionar tipo	Seleccionar cantidad de preguntas	Tomar evaluación
Tasktype	Select	select	start/go
Taskitem	Element	element	collection
Taskmodel	E learning	E learning	E learning
unaryRelationship			Optional → 1.1.3 - Tomar evaluacion
binaryRelationship			Enabling → 1.1.3 - Tomar evaluación → 1.1 - Ver curso
Decomposition	→ 1.1.2 - Nueva evaluacion	→ 1.1.2 - Nueva evaluacion	

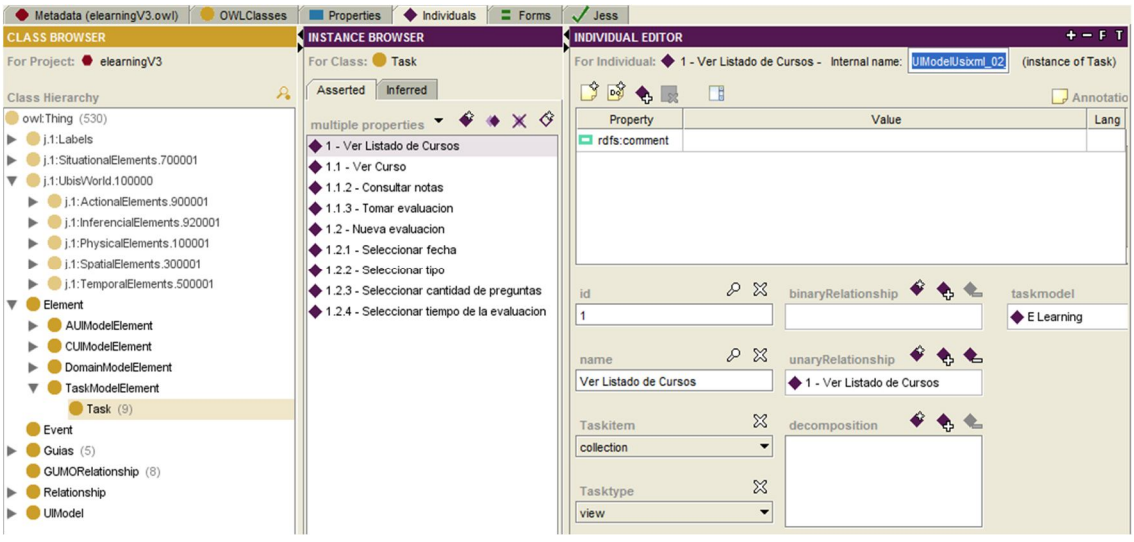


Imagen IV.1: Instancias del modelo de tareas.

Las propiedades unaryRelationship, binaryRelationship y Decomposition indican las relaciones entre las tareas. Se puede observar estas relaciones en el siguiente gráfico:

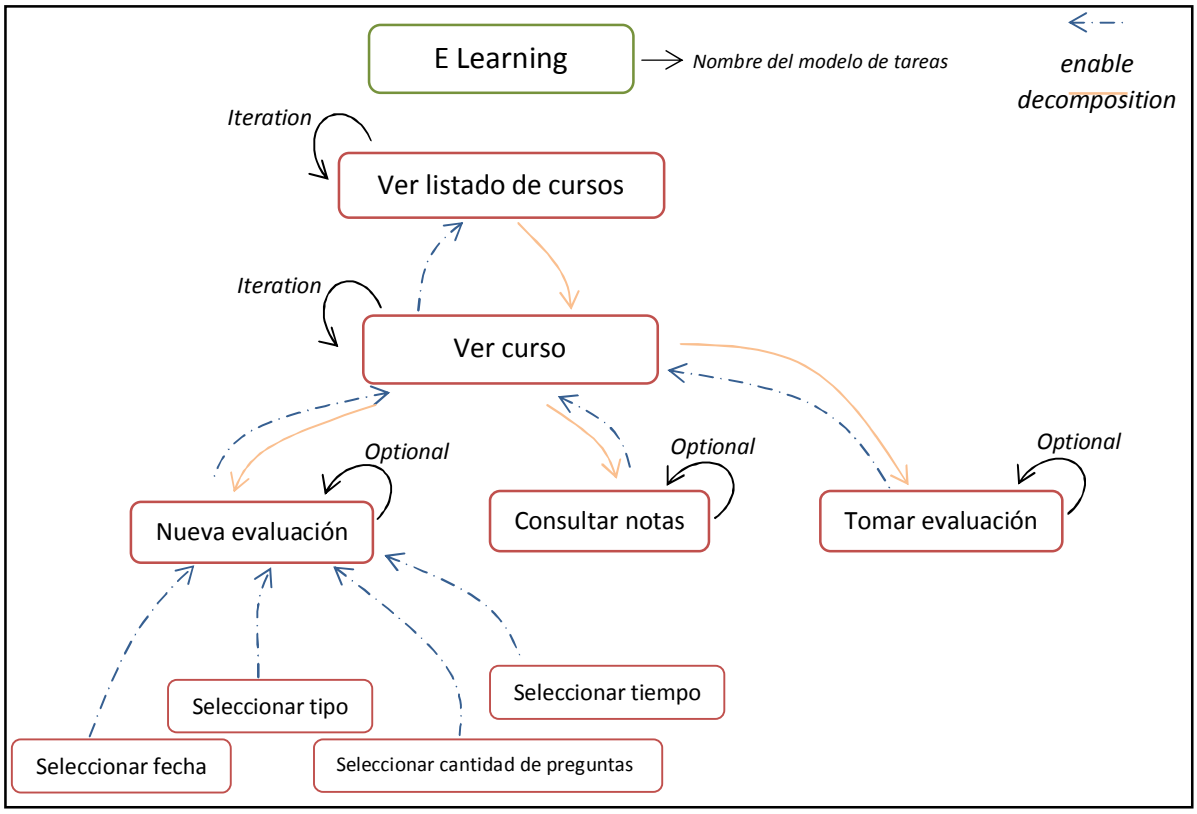


Figura IV.2: Relaciones entre las tareas.

IV.3.1.2. Modelo de Dominio

Para crear el modelo de dominio se instancian las clases *Attribute, Clase* y *Methods*. Cada una de las instancias de *Clase* puede poseer o no un conjunto de atributos y métodos.

Tabla IV.4. Instanciación del modelo de *dominio*. Subclase *Atributos*.

Modelo de dominio: *Atributos*	
name	**dataType**
Nombre del curso	string
Fecha	date
Calificación	integer
Pregunta	string
Numero de preguntas	integer
Tipo de evaluación	symbol
id.evaluacion	integer
id.curso	integer
Tiempo de la evaluación	time

Tabla IV.4. Instanciación del modelo de dominio. Subclase *Clases*.

Modelo de dominio: *Clases*				
clases	Curso	Evaluacion	Notas	EvaluacionAlumno
atributtes	•Nombre del curso •id.curso	•id.evaluacion •Fecha •Tiempo de la evaluación •Número de preguntas •Nombre del curso •Tipo de evaluación •Pregunta	•Nombre del curso •Calificación •id.evaluacion •id.curso	•id.evaluacion •Fecha •Tiempo de la evaluación •Número de preguntas •Nombre del curso •Tipo de evaluación •Pregunta
methods		•Crear evaluación		•Enviar resultados

La relación entre las clases, atributos y métodos se puede observar en la figura IV.3:

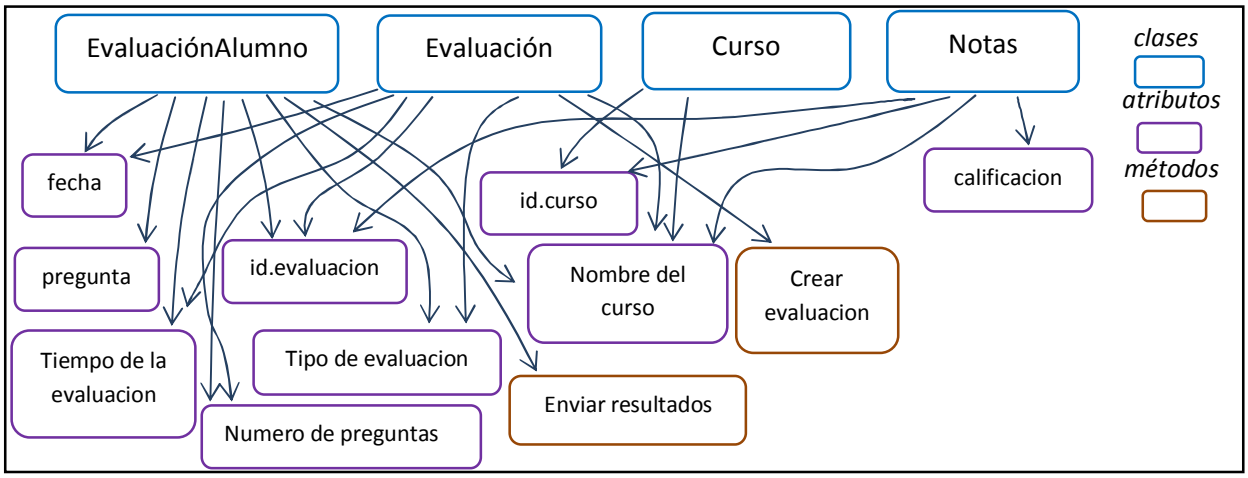


Figura IV.3: Relaciones entre las clases, atributos y métodos.

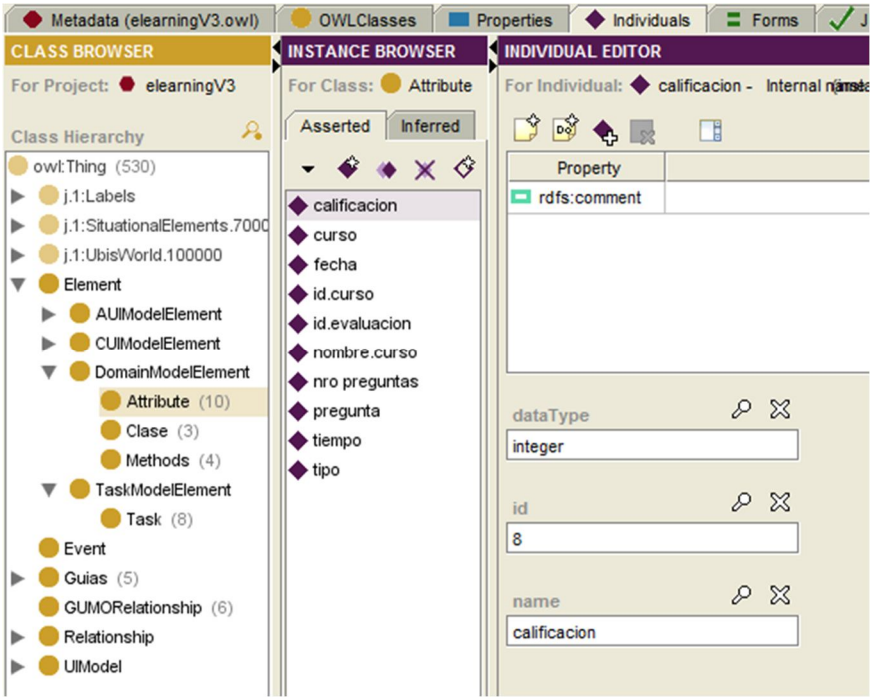


Imagen IV.3: Instancias de *Attribute* en el modelo de dominio.

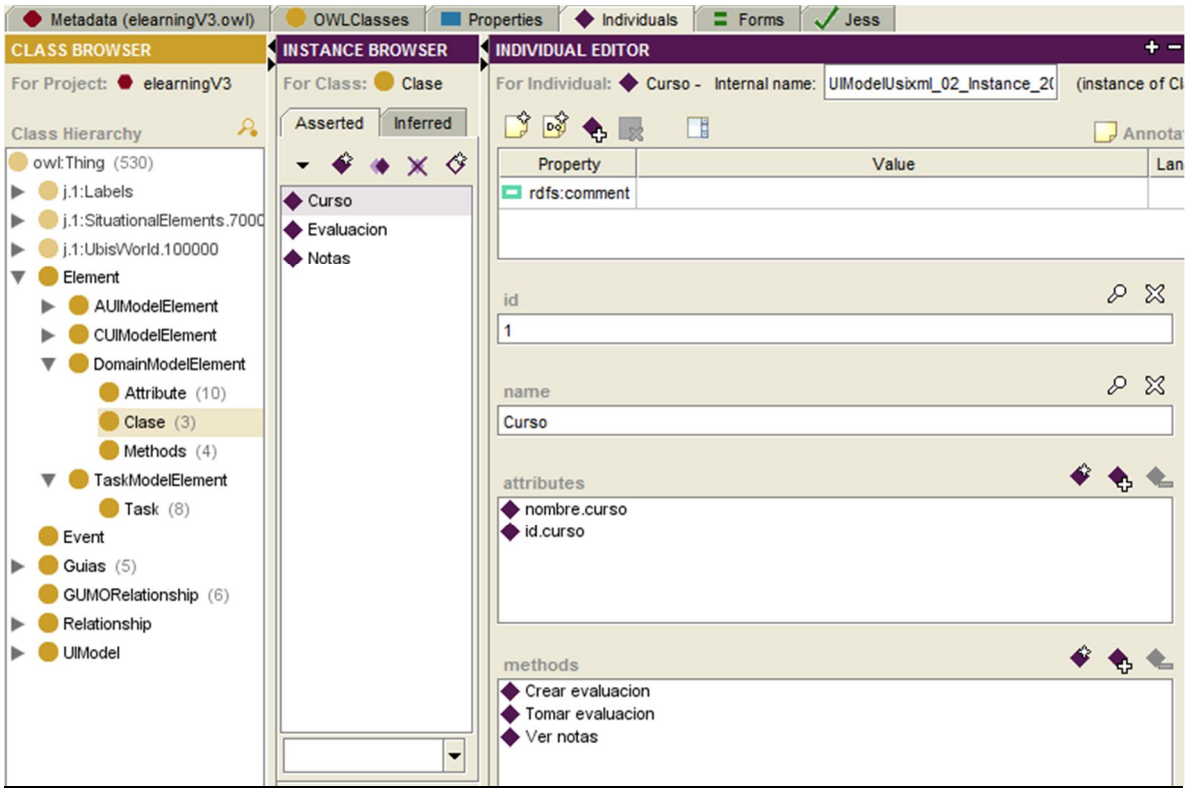


Imagen IV.2. Instancias de Clase en el modelo de dominio.

IV.3.1.3. Modelo de Mapeo

Para crear el modelo de mapeo se instancia la clase *Manipulates*. La tabla IV.5 muestra la relación entre las tareas y los elementos del dominio.

Tabla IV.5. Relación entre las tareas y los elementos del dominio.

Id	sourceManipulate	targetManipulate
1	Ver listado de Cursos	Curso
2	Nueva evaluación	Evaluacion
3	Seleccionar fecha	fecha
4	Seleccionar tipo	Tipo
5	Seleccionar cantidad de preguntas	nro preguntas
6	Consultar notas	notas
7	Tomar evaluacion	EvaluacionAlumno

tareas

Clase de dominio

Artributo de dominio

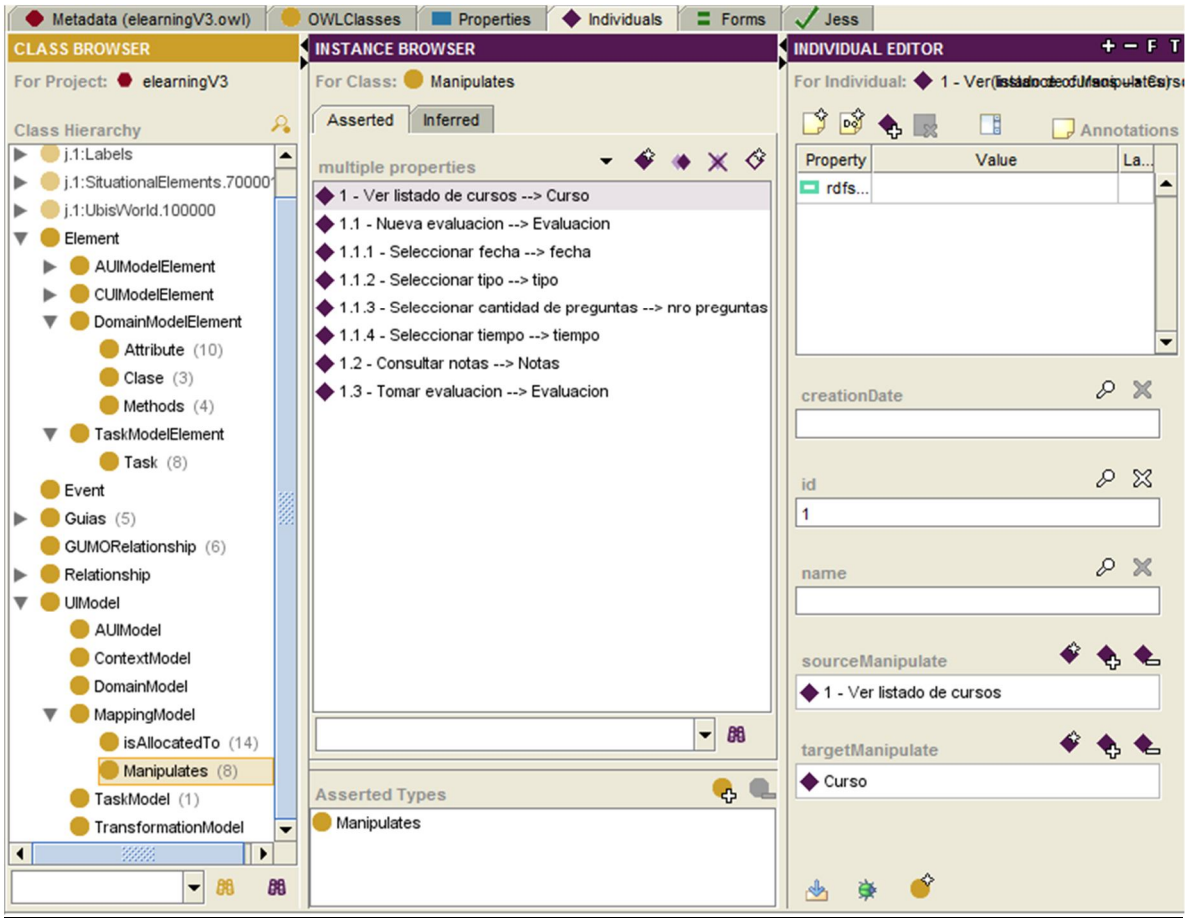


Imagen IV.4. Instancias de la clase *Manipulates* en el modelo de Mapeo.

IV.4. CREAR SOLUCIONES DE DISEÑO

IV.4.1. GENERACIÓN DE MÚLTIPLES IU ABSTRACTAS

Para poder generar las IU abstractas es necesario cumplir con los requisitos de los usuarios además de las guías de diseño.

IV.4.2. GUÍAS DE DISEÑO

Las guías de diseño se obtuvieron de *Research-Based Web Design & Usability Guidelines* [KOY04] y *Beyond ALT Text: Making the Web Easy to Use for Users with Disabilities* [NIE01], entre otros y hacen referencia a usuarios especiales. Se utilizarán las relacionadas con las características de los usuarios de este ejemplo.

Niños entre 3 a 15 años

G1. Utilizar íconos. Utilizar imágenes en lugar de botones.

- ✓ Usar Element imageComponent; propiedades: Id, help, hyperLinkTarget (especifica el nombre del archivo al que se accede haciendo clic en la imagen)

Personas con nivel cognitivo debajo de lo normal

G2. Mayor tiempo para terminar la evaluación y utilización de audio para textos.

- ✓ Element TimeSensor; propiedades: time. Se aumenta la cantidad de tiempo en un 20%.
- ✓ Usar Element VocalOutput; propiedades: volume, intonation, pitch, isInterrupible.

Personas zurdas

G3. Ubicar elementos de tipo menú o botón a la izquierda.

- ✓ Usar Element button; propiedad glueHorizontal; valor: left.

Personas con disminución de la vista.

G4. No utilizar texto muy pequeño para el texto del cuerpo, para links y botones.

- ✓ Usar Element OutputText; propiedades: textSize. Esta es una propiedad que se hereda desde 2DgraphicalIndividualComponent.

GUÍAS GENERALES

G5. Subrayado para todos los links

- ✓ Usar Element OutputText; propiedades: isUnderlined. Esta es una propiedad que se hereda desde 2DgraphicalIndividualComponent.

Tabla IV.5: Guías de diseño por usuario

Guías / Usuarios	G1	G2	G3	G4	G5
Manuel	*	*			*
Julieta	*			*	
Jorge			*		*
Pedro				*	*

Una vez creadas las guías de diseño, se crean las reglas para procesarlas y generar los elementos concretos.

Antes de realizar las pruebas para cada uno de los indicadores se ejecutarán las reglas para generar las interfaces sin considerar las características de los usuarios. Como se mencionó anteriormente, la información que se encuentra en los modelos de tareas y de dominio y sus relaciones produce el modelo de interfaz abstracta, que a su vez, permite generar interfaces concretas. Se ejecutaron las reglas y se obtuvieron las siguientes tres interfaces:

Interfaz 1: Ver listado de Cursos

1. **<window name=*Ver Listado de Cursos*>Ver listado de cursos**
2. **<outputtext url=*curso* name= *Curso*> Curso / >**
3. **<outputtext url=*profesor* name= *Profesor*> Profesor / >**

</window>

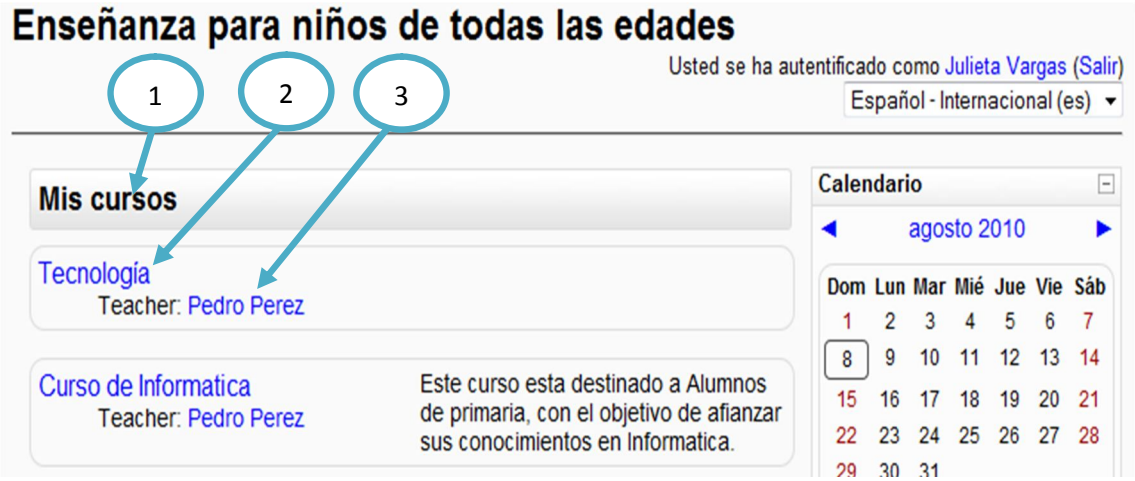


Imagen IV.5. Interfaz de Moodle para ver el listado de cursos.

Interfaz 2: Ver Curso

1. **<window name=*Ver Curso*>Ver curso**
2. **<outputtext url=*Consultar notas* name=*Consultar notas*>Consultar notas / >**
3. **<outputtext url=*tomar evaluacion* name=*Tomar evaluacion*>Tomar evaluacion / >**

</window>

Imagen IV.6. Interfaz de Moodle para ver un curso.

Interfaz 3: Consultar notas

```
1. <window name=Consultar notas>Calificaciones
2. <outputtext name=id.curso>id.curso />
3. <outputtext name=Nombre del curso>Nombre del Curso />
4. <outputtext name=id. evaluacion>id. evaluación />
5. <outputtext name=calificacion>calificación />
   </window>
```

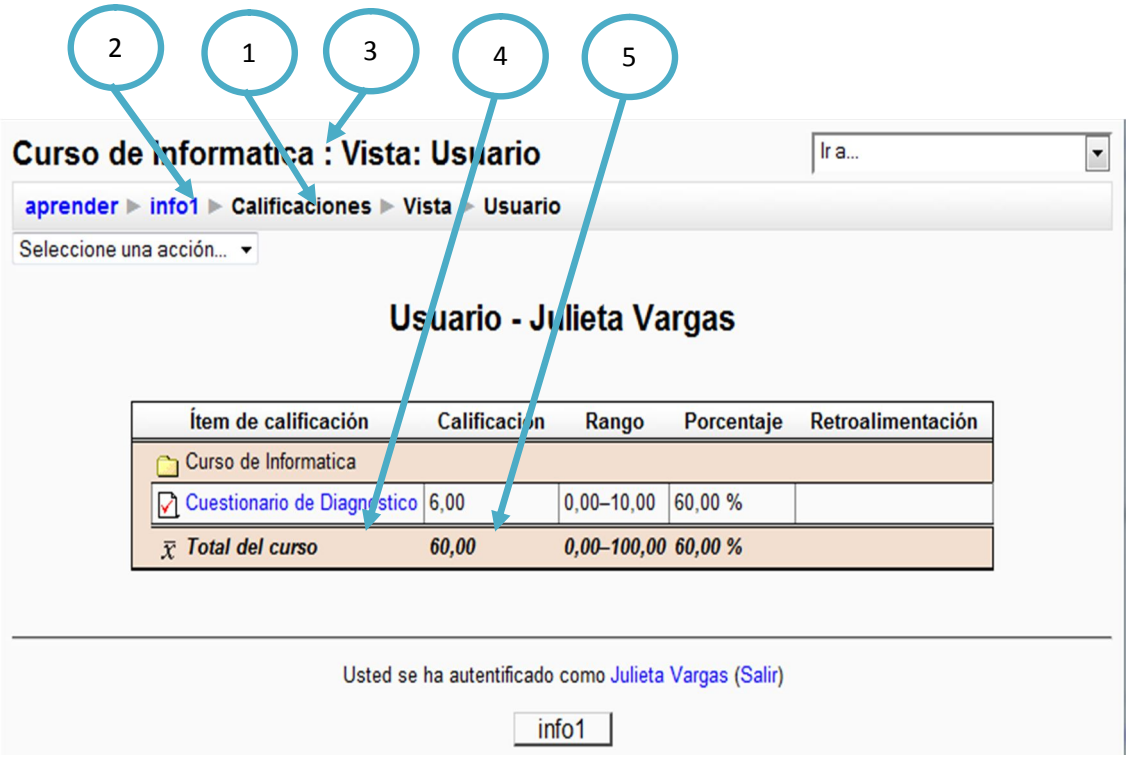


Imagen IV.7. Interfaz de Moodle para ver las notas.

Interfaz 4: Tomar evaluación

```
1. <window name=Tomar evaluacion>Nombre de evaluación
2. <outputtext name=fecha> fecha />
3. <outputtext name=id. evaluacion> id. evaluación />
4. <outputtext name=Nombre del curso> Nombre del curso />
5. <outputtext name=Tipo de evaluacion>Tipo de evaluación />
6. <timesensor name=tiempo de la evaluacion> tiempo de la evaluación />
   </window>
```

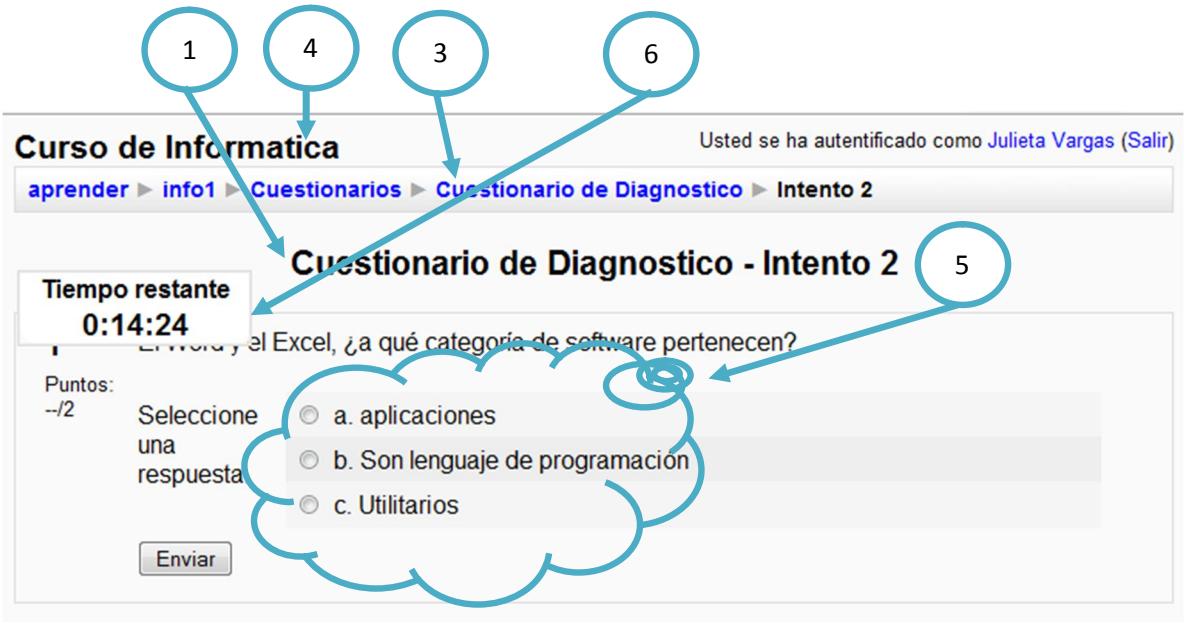


Imagen IV.8. Interfaz de Moodle para realizar una evaluación.

2) La fecha es asignada al finalizar la evaluación.

5) Tipo de evaluación: viene determinado por la forma en que el alumno puede resolver las actividades o preguntas. En el ejemplo el tipo de cuestionario es múltiple opción.

Interfaz 5: Nueva evaluación

1. **<window name=*Nueva evaluacion*>Agregando cuestionario**
2. **<outputtext name=*Nombre del curso*> Nombre de la curso />**
3. **<datepicker name=*fecha*> fecha de evaluacion />**
4. **<outputtext name=*id. evaluacion*> id. evaluación />**
5. **<listbox name=*Tipo de evaluacion*>Tipo de evaluación />**
6. **<listbox name=*tiempo de la evaluacion*> Tiempo de la evaluación />**
7. **<listbox name=*Numero de preguntas*> Número de preguntas />**

   </window>

4) El *id. evaluacion* es otorgado por el sistema al momento de crear una evaluación.

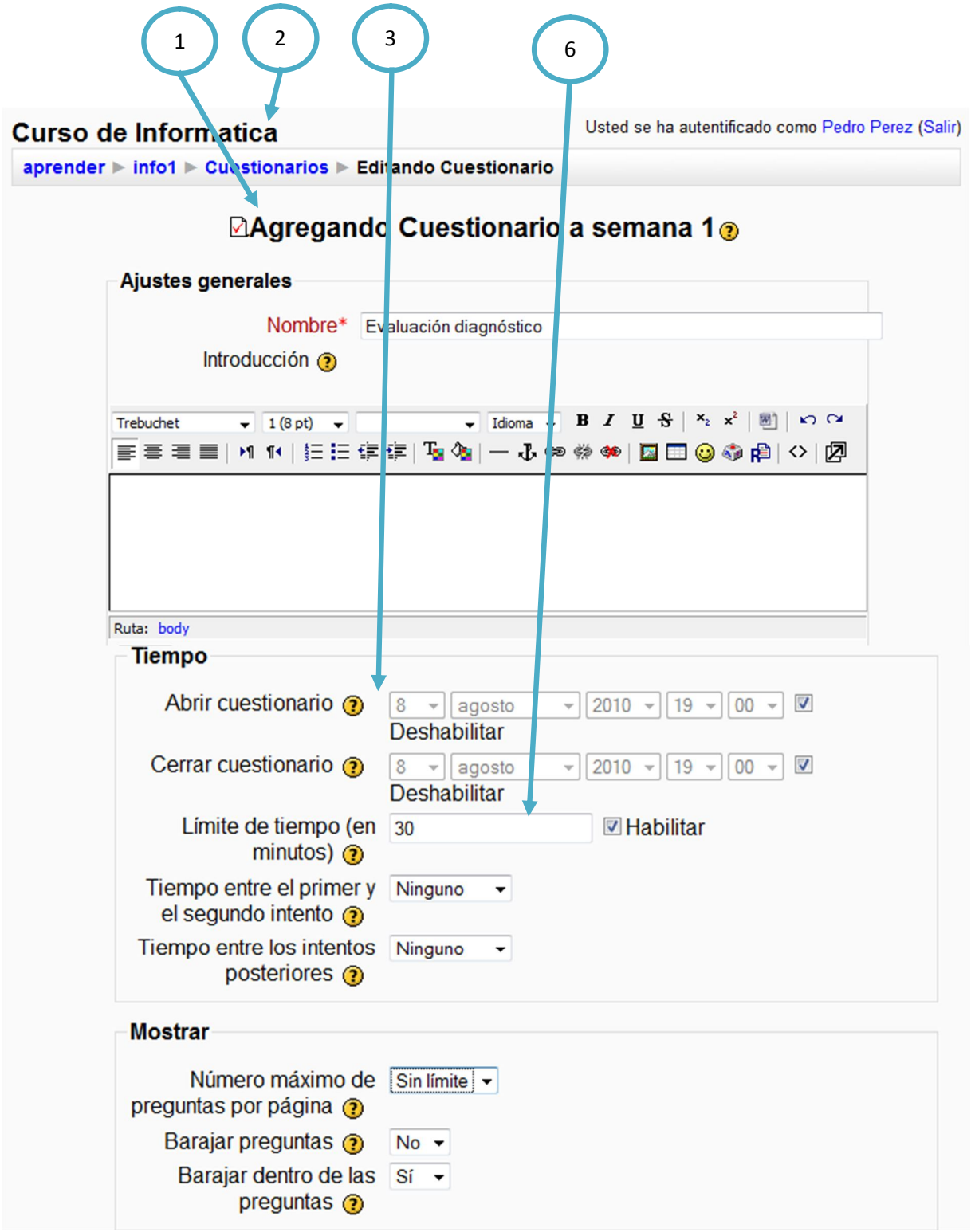


Imagen IV.9. Interfaz de Moodle para crear una nueva evaluación.

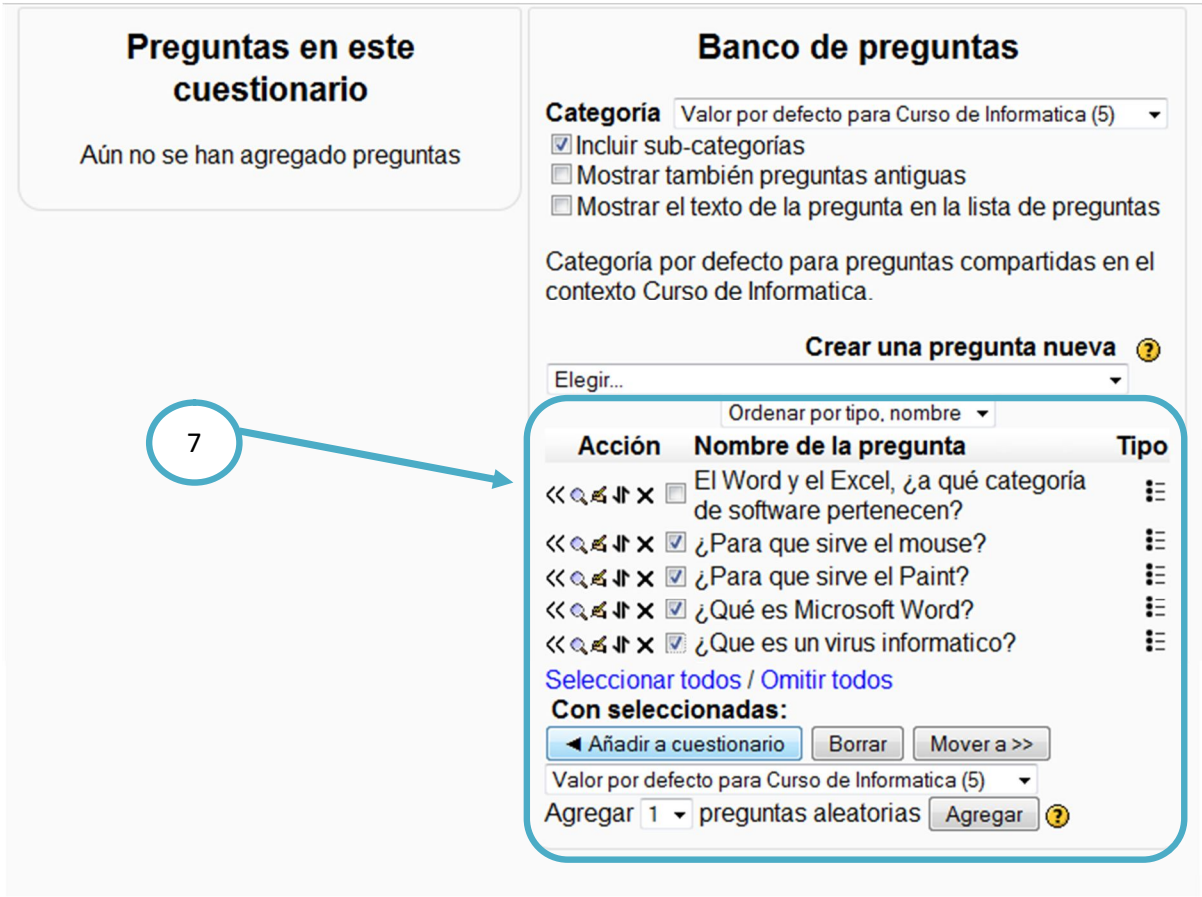


Imagen IV.7. Interfaz de Moodle para ver o agregar preguntas.

5) El tipo de evaluación es definido a partir de las preguntas seleccionadas, puede ser: Descripción, Ensayo, Respuestas anidadas, Opción múltiple, Respuesta corta, Numérica, Verdadero/Falso.

7) La cantidad de preguntas se establece en el momento de seleccionarlas.

El siguiente paso es ejecutar las reglas considerando el modelo de usuario. De esta forma se podrá observar si se generan múltiples interfaces que consideran correctamente las características de los usuarios.

IV.5. CASOS DE PRUEBA

Para comprobar el cumplimiento de los objetivos se deben definir variables y operacionalizarlas. Las variables son características observables referidas en este caso al prototipo. En la operacionalización se pasa a un nivel más concreto y específico a efectos de poder observar y medir las variables. Se dividen las variables en dimensiones, indicadores y posibles valores a tomar. Por último se especifica la manera en la cual se realizarán las pruebas de las mismas.

VARIABLES		OPERACIONALIZACIÓN DE LAS VARIABLES			INSTRUMENTOS
INDEPENDIENTES	DEPENDIENTES	DIMENSIONES	INDICADORES	VALORES POSIBLES	
PROTOTIPO	COMPLETITUD	USUARIOS	ROLES: Aceptación de los permisos	SI - NO	Diseño y ejecución de casos de prueba para medir cada indicador.
			CARACTERÍSTICAS FÍSICAS: Aceptación de las características físicas	SI - NO	
			CARACTERÍSTICAS COGNITIVAS Aceptación de las características cognitivas	SI - NO	
			CARACTERÍSTICAS DEMOGRÁFICAS Aceptación de las características demográficas	SI - NO	
		TAREAS	Aceptación de las tareas	SI - NO	
		DOMINIO	Aceptación de los elementos del dominio	SI - NO	
PROTOTIPO	CORRECTITUD	Grado de concordancia de las interfaces finales con los modelos de usuario, de tareas y de dominio.		Porcentaje	Diseño y ejecución de casos de prueba para medir cada indicador.
PROTOTIPO	USABILIDAD	APRENDIBILIDAD	• Esfuerzo requerido en el aprendizaje de uso del prototipo. • Existencia de herramientas de aprendizaje en el uso del prototipo		Creación y utilización de checklists para medir cada indicador
		EXTENSIBILIDAD	• Posibilidad de personalización de la funcionalidad del prototipo.		

IV.5.1. COMPLETITUD

Desarrollar una herramienta con características de completitud con respecto a los usuarios, tareas y elementos del dominio.

Descripción:

Para probar si la completitud se cumple se ejecutaron las reglas de transformación para generar las interfaces finales y se corroboró si las mismas se corresponden con la información de los modelos de usuario, tareas y dominio en su totalidad.

IV.5.1.1. Dimensión de usuario

Indicador: *Roles*

Se comenzará analizando la correspondencia de las interfaces con la relación que existe entre modelo de usuario y el modelo de tareas. Esta relación del modelo de tareas con el modelo de usuarios permite deducir las tareas permitidas para cada uno de ellos.

Por lo tanto no deberían generarse interfaces para aquellos usuarios sin relación con dicha tarea. Tampoco deben generarse enlaces a dichas interfaces.

Lo importante en esta prueba es observar que las interfaces generadas no posean elementos para tareas no permitidas, es decir, no debe existir un elemento de ningún tipo con la propiedad *name* que tenga el mismo nombre de una tarea no permitida.

Se observa en la siguiente tabla la relación de los usuarios con las tareas. Se deducen de estas relaciones los permisos de cada uno de los usuarios.

Tabla IV.6: Relación de Usuarios con sus tareas.

Usuario / Tareas	Manuel	Julieta	Jorge	Pedro
Ver listado de cursos	✓	✓	✓	✓
Consultar notas	✓	✓	✓	✓
Nueva evaluación	×	×	×	✓
Seleccionar fecha	×	×	×	✓
Seleccionar tipo	×	×	×	✓
Seleccionar cantidad de preguntas	×	×	×	✓
Seleccionar tiempo	×	×	×	✓
Tomar evaluacion	✓	✓	×	×

Usuario: **Manuel**

Interfaces generadas:

Ver listado de cursos

```
<window name=Ver Listado de cursos> Ver Listado de Cursos >
<vocalOutput name=id.curso> id.curso <vocalOutput/>
<vocalOutput name=Nombre del curso> Nombre del curso <vocalOutput/>
<outputtext url=Tomar evaluacion ("IsUnderlined") name=Tomar evaluacion> Tomar evaluación
<outputtext/>
 <outputtext url=Consultar notas ("IsUnderlined") name=Consultar notas> Consultar notas
<outputtext/>
</window>
```

Consultar notas

```
<window name=Consultar notas>Consultar notas />
<vocalOutput name= id.curso> id.curso <vocalOutput/>
<vocalOutput name= Nombre del curso>Nombre del curso <vocalOutput/>
<vocalOutput name= id. evaluacion> id. evaluacion <vocalOutput/>
<vocalOutput name= calificacion> Calificacion <vocalOutput/>
    </window>
```

Tomar evaluación

<window name=*Tomar evaluacion*>Tomar evaluacion/ >
<vocalOutput name= *fecha*> Fecha <vocalOutput/>
<vocalOutput name= *id. evaluacion*> id. evaluacion <vocalOutput/>
<vocalOutput name= *Nombre del curso* > Nombre del curso <vocalOutput/>
<vocalOutput name= *Tipo de evaluacion*>Tipo de evaluacion <vocalOutput/>
<vocalOutput name= *nro preguntas*>Numero de preguntas <vocalOutput/>
<vocalOutput name= *pregunta*>Pregunta <vocalOutput/>
<timesensor name= *tiempo de la evaluación* ("time+20")>Tiempo de la evaluacion<timesensor/>
<imageComponent name=Enviar resultados ("Id") ("hyperLinkTarget") ("help") >Enviar resultados <imageComponent />
</window>

Se analizan los resultados obtenidos.

Usuario: Manuel				
Tareas	**Permitida**	**Generada**	**Interfaz**	**Elementos concreto generado**
Ver listado de cursos	✓	✓	Ver listado de cursos	<window name=*Ver listado de cursos*>Ver listado de cursos</window>
Consultar notas	✓	✓	Consultar notas	• <window name=*Consultar notas*>Consultar notas</window> • <outputtext url=*Consultar notas* ("IsUnderlined") name= *Consultar notas*> Consultar notas </outputtext>
Nueva evaluación	×	×		
Seleccionar fecha	×	×		
Seleccionar tipo	×	×		
Seleccionar cantidad de preguntas	×	×		
Seleccionar tiempo	×	×		
Tomar evaluación	✓	✓	Tomar evaluación	• <window name=*Tomar evaluacion*>Tomar evaluación</window> • <outputtext url=*Tomar evaluacion* ("IsUnderlined") name= *Tomar evaluacion*>Tomar evaluación </outputtext>

La tabla anterior muestra que para el usuario Manuel se generan elementos concretos únicamente para las tareas permitidas. Por lo tanto los resultados obtenidos son los esperados.

Indicador	**Usuario**	**Valor del indicador**
Roles	Manuel	SI

Usuario: **Julieta**

Interfaces generadas:

Listado de Cursos

<window name=*Ver listado de cursos*>Ver listado de cursos / >
<outputtext ("textSize") name= *id.curso*> id.cursos / >
<outputtext ("textSize") name= *Nombre del curso*> Nombre del curso / >
<outputtext ("textSize") url=*Tomar evaluacion* ("IsUnderlined") name= *Tomar evaluacion*>Tomar evaluacion / >
<outputtext ("textSize") url=*Consultar notas* ("IsUnderlined") name= *Consultar notas*> Consultar notas / >
<window>

Consultar notas

<window name=*Consultar notas*> Consultar notas >
<outputtext ("textSize") name= *id.curso*> id.curso / >
<outputtext ("textSize") name= *Nombre del curso*> Nombre del curso / >
<outputtext ("textSize") name= *id. evaluacion*> id. evaluacion / >
<outputtext ("textSize") name= *calificacion*> Calificacion / >
<window>

Tomar evaluación

<window name=*Tomar evaluacion*>Tomar evaluacion/ >
<outputtext ("textSize") name= *fecha*> Fecha />
<outputtext ("textSize") name= *id. evaluacion*> id. evaluacion />
<outputtext ("textSize") name= *Nombre del curso*> Nombre del curso />
<outputtext ("textSize") name= *Tipo de evaluacion*>Tipo de evaluacion />
<outputtext ("textSize") name= *nro preguntas*>Numero de preguntas />
<outputtext ("textSize") name= *pregunta*>Pregunta />
<timesensor name= *tiempo de la evaluación*> Tiempo de la evaluacion <timesensor />
<imageComponent name=Enviar resultados ("Id") ("hyperLinkTarget") ("help") >Enviar resultados </imageComponent>
</window>

Se hace un análisis de los resultados obtenidos.

Usuario: Julieta				
Tareas	**Permitida**	**Generada**	**Interfaz**	**Elementos concreto generado**
Listado de Cursos	✓	✓	Listado de Cursos	<window name=*Listado de Cursos*>Listado de Cursos</window>
Consultar notas	✓	✓	Consultar notas	• <window name=*Consultar notas*>Consultar notas</window> • <outputtext url=*Consultar notas* ("IsUnderlined") name= *Consultar notas*> Consultar notas </outputtext>
Nueva evaluación	×	×		
Seleccionar fecha	×	×		
Seleccionar tipo	×	×		
Seleccionar cantidad de preguntas	×	×		
Seleccionar tiempo	×	×		
Tomar evaluación	✓	✓	Tomar evaluación	• <window name=*Tomar evaluacion*>Tomar evaluación</window> • <outputtext url=*Tomar evaluacion* ("IsUnderlined") name= *Tomar evaluacion*>Tomar evaluación </outputtext>

La tabla anterior muestra que para Julieta se generan elementos concretos únicamente para las tareas permitidas. Por lo tanto se obtuvieron los resultados esperados.

Indicador	**Usuario**	**Valor del indicador**
Roles	Julieta	SI

Usuario: **Jorge**

Interfaces generadas:

Listado de Cursos

<graphicalAlignment name=*Listado de Cursos*>Listado de Cursos

<outputtext name= *id. curso* > id.curso<outputtext/>

<outputtext name= *Nombre del curso*> Nombre del curso <outputtext/>

<outputtext url=*Consultar notas* ("IsUnderlined") name= *Consultar notas*> Consultar notas <outputtext/>

<graphicalAlignment>

Consultar notas

<graphicalAlignment name=*Consultar notas*> Consultar notas

<outputtext name= *id.curso*> id.curso / >

<outputtext name= *Nombre del curso*> Nombre del curso / >

<outputtext name= *id. evaluacion*> id. evaluacion / >

<outputtext name= *calificacion*> Calificacion / >

<graphicalAlignment>

Los resultados obtenidos se analizan en la siguiente tabla.

Usuario: Jorge				
Tareas	**Permitida**	**Generada**	**Interfaz**	**Elementos concreto generado**
Listado de Cursos	✓	✓	Listado de Cursos	<window name=*Listado de Cursos*>Listado de Cursos</window>
Consultar notas	✓	✓	Consultar notas	• <window name=*Consultar notas*>Consultar notas</window> • <outputtext url=*Consultar notas* ("IsUnderlined") name= *Consultar notas*> Consultar notas </outputtext>
Nueva evaluación	×	×		
Seleccionar fecha	×	×		
Seleccionar tipo	×	×		
Seleccionar cantidad de preguntas	×	×		
Seleccionar tiempo	×	×		
Tomar evaluación	×	×		

La tabla anterior muestra que para el usuario Jorge se generan elementos concretos únicamente para las tareas permitidas. Por lo tanto los resultados obtenidos coinciden con los esperados.

Indicador	**Usuario**	**Valor del indicador**
Roles	Jorge	SI

Usuario: **Pedro**

Interfaces generadas:

Consultar notas

<window name=*Consultar notas*> Consultar notas </window>

<outputtext ("textSize") name= *id.curso*> id.curso </outputtext >

<outputtext ("textSize") name= *Nombre del curso*> Nombre del curso </outputtext >

<outputtext ("textSize") name= *id. evaluacion*> id. evaluacion </outputtext >

<outputtext ("textSize") name= *calificacion*> Calificacion </outputtext >

<window>

Listado de Cursos

<window name=*Ver Listado de Cursos*>Ver Listado de Cursos</window>

<outputtext ("textSize") name= *id.curso*> id.curso </outputtext>

<outputtext ("textSize") name= *Nombre del curso*> Nombre del curso </outputtext>

<outputtext url=*Nueva evaluacion* ("IsUnderlined") name=*Nueva evaluacion*>Nueva evaluacion </outputtext>

<outputtext url=*Consultar notas* ("IsUnderlined") name=*Consultar notas*> Consultar notas </outputtext>

<window>

Nueva evaluación

<window name=*Nueva evaluacion*> Nueva evaluación </window>

<datepicker name= *fecha*> fecha </datepicker>

<outputtext ("textSize") name= *id. evaluacion*> id. Evaluacion </outputtext>

<outputtext ("textSize") name= *Nombre del curso*> Nombre del curso </outputtext>

<listbox name= *Tipo de evaluacion*> Tipo de evaluación </listbox>

<listbox name= *Tiempo de la evaluacion*> Tiempo de la evaluación </listbox>

<listbox name= *Numero de preguntas*> Numero de preguntas </listbox>

<button name=*Crear evaluacion*> Crear evaluación </button>

<window>

En la siguiente tabla son analizados los resultados obtenidos.

Usuario: Pedro				
Tareas	Permitida	Generada	Interfaz	Elementos concreto generado
Ver Listado de Cursos	✓	✓	Ver Listado de Cursos	<window name=*Ver Listado de Cursos*>Ver Listado de Cursos</window>
Consultar notas	✓	✓	Consultar notas	• <window name=*Consultar notas*>Consultar notas</window> • <outputtext url=*Consultar notas* ("textSize") ("IsUnderlined") name= *Consultar notas*> Consultar notas </outputtext>
Nueva evaluación	✓	✓	Nueva evaluación	• <window name=Nueva evaluacion>Nueva evaluación</window> • <outputtext url=*Nueva evaluacion* ("textSize") ("IsUnderlined") name= *Nueva evaluacion*>Nueva evaluación </outputtext>
Seleccionar fecha	✓	✓		• <datepicker name= *fecha*> fecha </datepicker >
Seleccionar tipo	✓	✓		<listbox name= *Tipo de evaluacion* >Tipo de evaluación </ listbox >
Seleccionar cantidad de preguntas	✓	✓		<listbox name= *Numero de preguntas*> Número de preguntas </listbox>
Seleccionar tiempo de la evaluación	✓	✓		<listbox name= *tiempo de la evaluacion*> tiempo de la evaluación </listbox>
Tomar evaluación	×	×		

La tabla anterior muestra que para el usuario Pedro se generan elementos concretos únicamente para las tareas permitidas. Por lo tanto los resultados son los esperados.

Indicador	**Usuario**	**Valor del indicador**
Roles	Pedro	SI

Indicadores: *Características físicas, cognitivas y demográficas.*

Como se comentó anteriormente, para probar la universalidad del diseño se debe probar que las interfaces generadas satisfagan las particularidades de cada uno de los usuarios. Por ello en el modelo de usuario existe información sobre las características de cada uno.

Pero aun así, esto no es suficiente para realizar esta prueba. También se deben poder traducir estas características en elementos de interfaz particulares, para observar que las interfaces finales se adaptan a cada uno de los usuarios.

Es por esto que se crean las guías de diseño que indican los elementos concretos a utilizar para una condición dada.

Entonces, para este indicador observaremos si las interfaces finales aceptan cada una de las características que los usuarios poseen.

Se observa en la siguiente tabla las clases de GUMO utilizadas para instanciar las características de los usuarios.

Tabla IV.11. Relación de Usuarios con sus características.

Clases de GUMO	Instancias de las clases			
Person	Manuel	Julieta	Jorge	Pedro
Age	12	12	38	56
CognitiveLoad	bajo			
LeftHanded			si	
AbilityToSee		bajo		bajo

Usuario: **Manuel**

Se observa en la siguiente tabla los elementos de interfaz que le corresponden al usuario Manuel de acuerdo a la guías de diseño y a sus características.

Usuario: Manuel		
Guías del usuario	Elemento a reemplazar	Elemento nuevo
1	**button**	**imageComponent**
2	• **Outputtext** • **timesensor**	• **vocalOutput** • **timesensor time=+20**
5	**outputtext url**	**outputtext url isUnderlined**

✱ Listado de Cursos

Se observa la interfaz *Listado de Cursos* original, sin considerar las guías de diseño. Se remarca con un cuadro los elementos a reemplazar.

```
<window name=Listado de Cusrsos>Listado de Cursos
<outputtext name= id.curso> id.curso <outputtext/>
<outputtext name= Nombre del curso> Nombre del curso <outputtext/>
<outputtext name= Curso> Curso <outputtext/>
<outputtext url=Tomar evaluacion name= Tomar evaluacion> Tomar evaluación <outputtext/>
<outputtext url=Consultar notas name= Consultar notas> Consultar notas <outputtext/>
<window>
```

Luego de ejecutar las reglas de transformación se obtuvo la siguiente interfaz:

```
<window name=Listado de Cursos>Listado de Cursos
<vocalOutput name= id.curso> id.curso <outputtext/>
<vocalOutput name= Nombre del curso> Nombre del curso <outputtext/>
<vocalOutput name= Curso> Curso <outputtext/>
<outputtext url=Tomar evaluacion ("IsUnderlined") name= Tomar evaluacion>Tomar evaluación
<outputtext/>
<outputtext url=Consultar notas ("IsUnderlined") name=Consultar notas>Consultar notas
<outputtext/>
<window>
```

Ya que las características de los usuarios están asociadas a guías de diseño. Se mide la aceptación de las mismas mediante la aceptación de las que le corresponde al usuario *Manuel* para la interfaz *Listado de cursos*.

Interfaz: Listado de cursos			**Usuario: Manuel**	
Guías del usuario	**Ejecución**	**Elementos a reemplazar**	**Elementos reemplazados**	**Aceptación**
1	No			--
2	Si	Curso, id.curso, Nombre del curso	Curso, id.curso, Nombre del curso	Si
5	Si	Tomar evaluacion, Consultar notas	Tomar evaluacion, Consultar notas	Si

Se observa que fueron reemplazados todos los elementos que debieron serlo.

✱ Consultar notas

Se analiza ahora la interfaz Consultar notas. La interfaz original es la siguiente.

```
<window name=Consultar notas>Consultar notas
<outputtext name= id.curso> id.curso <outputtext/>
<outputtext name= Nombre del curso> Nombre del curso <outputtext/>
<outputtext name= id. evaluacion> id. evaluacion <outputtext/>
<outputtext name= calificacion> calificación <outputtext/>
</window>
```

Luego de ejecutar las reglas de transformación se obtuvo la siguiente interfaz:

```
<window name=Consultar notas> Consultar notas
<vocalOutput name= id.curso> id.curso / >
<vocalOutput name= Nombre del curso> Nombre del curso / >
<vocalOutput name= id. evaluacion> id. evaluacion / >
<vocalOutput name= calificacion> Calificación / >
<window>
```

Se observa que la única guía que corresponde ejecutar es la número 2 y que se ejecuta correctamente.

Interfaz: Consultar notas		**Usuario: Manuel**		
Guías del usuario	Ejecución	Elementos a reemplazar	Elementos reemplazados	Aceptación
1	No			--
2	Si	id.curso, Nombre del curso, id.evaluacion, Calificacion	id.curso, Nombre del curso, id.evaluacion, Calificacion	Si
5	No			--

- **Tomar evaluación**

La interfaz original es la siguiente.

```
<window name=Tomar evaluacion>Tomar evaluación
<outputtext name= fecha> fecha <outputtext/>
<outputtext name= id. evaluacion>id. evaluacion<outputtext/>
<outputtext name= Nombre del curso> Nombre del curso <outputtext/>
<outputtext name= Tipo de evaluacion>Tipo de evaluación <outputtext />
<outputtext name=nro preguntas> Numero de preguntas<outputtext/>
<outputtext name=pregunta>Pregunta<outputtext/>
<timesensor name= tiempo de la evaluacion> tiempo de la evaluación <timesensor/>
<button name= Enviar resultados> Enviar resultados />
<window>
```

Luego de ejecutar las reglas de transformación se obtuvo la siguiente interfaz.

```
<window name=Tomar evaluacion>Tomar evaluacion
<vocalOutput name= fecha> Fecha />
<vocalOutput name= id. evaluacion> id. evaluacion />
<vocalOutput name= Nombre del curso> Nombre del curso />
<vocalOutput name= Tipo de evaluacion>Tipo de evaluación />
<vocalOutput name=nro preguntas>Numero de preguntas/>
<vocalOutput name=pregunta> Pregunta/>
<timesensor name= tiempo de la evaluación ("time+20")> Tiempo de la evaluación / >
<imageComponent name=Enviar
resultados("Id")("hyperLinkTarget")("help")("nombre")>Enviar resultados/>
<window>
```

Se examinan ahora, cada uno de los elementos de la interfaz anterior.

Interfaz: Tomar evaluación		**Usuario: Manuel**		
Guías	Ejecución	Elementos a reemplazar	Elementos reemplazados	Aceptación
1	Si	Enviar resultados	Enviar resultados	Si
2	Si	Fecha, id.evaluacion, Nombre del curso, Tipo de evaluación, Numero de preguntas, Pregunta, Tiempo de la evaluacion	Fecha, id.evaluacion, Nombre del curso, Tipo de evaluación, Numero de preguntas, Pregunta, Tiempo de la evaluacion	Si
5	No			--

Se observa que efectivamente se reemplazan todos los elementos que corresponde reemplazar.

Usuario: **Julieta**

Las guías que se disparan para el usuario Julieta son las siguientes.

Usuario: Julieta		
Guías del usuario	Elemento a reemplazar	Elemento nuevo
1	**button**	**imageComponent**
4	**Outputtext**	**Outputtext textSize=20**
5	**outputtext url**	**outputtext url isUnderlined**

✱ Listado de Cursos

Interfaz original

```
<window name=Listado de Cursos>Listado de Cursos
<outputtext name= id.curso> id.curso<outputtext/>
<outputtext name= Nombre del curso>Nombre del curso<outputtext/>
<outputtext url=Tomar evaluacion name= Tomar evaluacion> Tomar evaluación<outputtext/>
<outputtext url=Consultar notas name= Consultar notas> Consultar notas<outputtext/>
<window>
```

Luego de ejecutar las reglas de transformación se obtuvo la siguiente interfaz:

```
<window name=Listado de Cursos>Listado de Cursos
<outputtext ("textSize") name= id.curso> id.curso / >
<outputtext ("textSize")  name= Nombre del curso> Nombre del curso / >
<outputtext("textSize") url=Tomar evaluacion("IsUnderlined")name=Tomar evaluacion>Tomar
evaluación/>
<outputtext("textSize")url=Consultar notas("IsUnderlined")name=Consultar notas>Consultar
notas/>
</window>
```

Se examinan ahora, cada uno de los elementos de la interfaz anterior. Se puede observar que fueron remplazados correctamente todos los elementos que se esperaban.

Interfaz: Listado de Cursos		**Usuario: Julieta**		
Guías del usuario	Ejecución	Elementos a reemplazar	Elementos reemplazados	Aceptación
1	No			--
4	Si	id.curso, Nombre del curso	id.curso, Nombre del curso	Si
5	Si	Tomar evaluacion, Consultar notas	Tomar evaluacion, Consultar notas	Si

✱ Consultar notas

Interfaz original

```
<window name=Consultar notas>Consultar notas
<outputtext name= id.curso> id.curso / >
<outputtext name= Nombre del curso> Nombre del curso / >
<outputtext name= id. evaluacion> id. evaluacion / >
<outputtext name= calificacion> calificación / >
</window>
```

Luego de ejecutar las reglas de transformación se obtuvo la siguiente interfaz

```
<window name=Consultar notas> Consultar notas
<outputtext ("textSize") name= id.curso> id.curso />
<outputtext ("textSize") name= Nombre del curso> Nombre del curso />
<outputtext ("textSize") name= id. evaluacion> id. evaluacion />
<outputtext ("textSize") name= calificacion> Calificación />
<window>
```

A continuación se revisan los elementos de la interfaz anterior. Fueron modificados los elementos esperados.

Interfaz: Consultar notas		**Usuario: Julieta**		
Guías del usuario	Ejecución	Elementos a reemplazar	Elementos reemplazados	Aceptación
1	No			--
4	Si	id.curso, Nombre del curso, id.evaluacion, Calificacion	id.curso, Nombre del curso, id.evaluacion, Calificacion	Si
5	No			--

✱ Tomar evaluación

Interfaz original

```
<window name=Tomar evaluacion>Tomar evaluación
<outputtext name= fecha> fecha />
<outputtext name= id. evaluacion> id. evaluacion />
<outputtext name= Nombre del curso> Nombre del curso />
<outputtext name= Tipo de evaluacion>Tipo de evaluación />
<outputtext name= nro preguntas>Número de preguntas />
<outputtext name= pregunta>Pregunta />
<timesensor name= tiempo de la 133valuación> tiempo de la evaluación />
<button name= Enviar resultados> Enviar resultados />
<window>
```

Luego de ejecutar las reglas de transformación se obtuvo la siguiente interfaz

```
<window name=Tomar 134valuación>Tomar evaluación
<outputtext ("textSize") name= fecha> Fecha />
<outputtext ("textSize") name= id. Evaluacion> id. Evaluacion />
<outputtext ("textSize") name= Nombre del curso> Nombre del curso />
<outputtext ("textSize") name= Tipo de 134valuación>Tipo de evaluación />
<outputtext ("textSize") name= Numero de preguntas>Número de preguntas />
<outputtext ("textSize") name= pregunta>Pregunta />
<timesensor name= tiempo de la 134valuación> Tiempo de la evaluación />
<imageComponent name=Enviar resultados ("Id") ("hyperLinkTarget") ("help")
("nombre")>Enviar resultados/>
<window>
```

A continuación se examina, cada uno de los elementos de la interfaz anterior.

Interfaz: Tomar evaluacion		**Usuario: Julieta**		
Guías del usuario	**Ejecución**	**Elementos a reemplazar**	**Elementos reemplazados**	**Aceptación**
1	Si	Enviar resultados	Enviar resultados	Si
4	Si	Fecha, id.evaluacion, Nombre del curso, Tipo de evaluación, Numero de preguntas, Pregunta	Fecha, id.evaluacion, Nombre del curso, Tipo de evaluación, Numero de preguntas, Pregunta	Si
5	No			--

Se observa en la tabla anterior que los resultados son los esperados.

Usuario: **Jorge**

Las guías que se ejecutan para el usuario Jorge son las siguientes.

Usuario: Jorge		
Guías del usuario	Elemento a reemplazar	Elemento nuevo
3	**window**	graphicalAlignment
5	**outputtext url**	**outputtext url isUnderlined**

- **Listado de Cursos**

Interfaz original

```
<window name=Listado de Cursos>Listado de Cursos
<outputtext name= id.curso> id.curso / >
<outputtext name= Nombre del curso> Nombre del curso / >
<outputtext url=Consultar notas name= Consultar notas> Consultar notas / >
<window>
```

Luego de ejecutar las reglas de transformación se obtuvo la siguiente interfaz

```
<graphicalAlignment name=Listado de Cursos>Listado de Cursos
<outputtext name= id.curso> id.curso />
<outputtext name= Nombre del curso> Nombre del curso />
<outputtext url=Consultar notas ("IsUnderlined") name= Consultar notas> Consultar notas />
<graphicalAlignment>
```

En la siguiente tabla se analizan, los elementos de la interfaz generada. Se observa que se logra la aceptación de las características del usuario al reemplazarse los elementos esperados.

Interfaz: Listado de Cursos		**Usuario: Jorge**		
Guías	Ejecución	Elementos a reemplazar	Elementos reemplazados	Aceptación
3	Si	Listado de cursos	Listado de cursos	Si
5	Si	Consultar notas	Consultar notas	Si

- **Consultar notas**

Interfaz original

```
<window name=Consultar notas>Consultar notas
<outputtext name= id.curso> id.curso / >
<outputtext name= Nombre del curso> Nombre del curso / >
<outputtext name= id. evaluacion> id. evaluacion / >
<outputtext name= calificacion> calificación / >
</window>
```

Luego de ejecutar las reglas de transformación se obtuvo la siguiente interfaz

```
<graphicalAlignment name=Consultar notas> Consultar notas
<outputtext name= id.curso> id.curso />
<outputtext name= Nombre del curso> Nombre del curso />
<outputtext name= id. evaluacion> id. evaluacion />
<outputtext name= calificacion> Calificación />
<graphicalAlignment>
```

Los elementos de la interfaz generada son analizados a continuación

Interfaz: Consultar notas		**Usuario: Jorge**		
Guías	Ejecución	Elementos a reemplazar	Elementos reemplazados	Aceptación
3	Si	Consultar notas	Consultar notas	Si
5	No			--

Usuario: **Pedro**

Las guías que se ejecutan para el usuario Pedro son las siguientes.

Interfaz: Listado de Cursos		**Usuario: Pedro**
Guías del usuario	Elemento a reemplazar	Elemento nuevo
4	**Outputtext**	**Outputtext textSize=20**
5	**outputtext url**	**outputtext url isUnderlined**

✱ Listado de cursos

Interfaz original

```
<window name=Ver Listado de Cursos>Ver Listado de Cursos
<outputtext name= id.curso> id.curso/>
<outputtext name= Nombre del curso> Nombre del curso/>
<outputtext url=Nueva evaluacion name=Nueva evaluacion> Nueva evaluación/>
<outputtext url=Consultar notas name= Consultar notas> Consultar notas />
<window>
```

Resultado obtenido luego de ejecutar las reglas:

```
<window name=Listado de Cursos>Listado de Cursos
<outputtext ("textSize") name= id.curso> id.curso/>
<outputtext ("textSize") name= Nombre del curso> Nombre del curso />
<outputtext ("textSize") url=Nueva evaluacion ("IsUnderlined") name= Nueva evaluacion>Nueva evaluación />
<outputtext ("textSize") url=Consultar notas ("IsUnderlined") name=Consultar notas>Consultar notas/>
<window>
```

Examinamos ahora, cada uno de los elementos de la interfaz anterior.

Interfaz: Listado de Cursos		**Usuario: Pedro**		
Guías del usuario	Ejecución	Elementos a reemplazar	Elementos reemplazados	Aceptación
4	Si	id.curso, Nombre del curso	id.curso, Nombre del curso	Si
5	Si	Nueva evaluacion, Consultar notas	Nueva evaluacion, Consultar notas	Si

✱ Consultar notas

Interfaz original

```
<window name=Consultar notas>Consultar notas
<outputtext name= id.curso> id.curso / >
<outputtext name= Nombre del curso> Nombre del curso / >
<outputtext name= id. evaluacion> id. evaluacion / >
<outputtext name= calificacion> calificación / >
</window>
```

Resultado obtenido luego de ejecutar las reglas:

```
<window name=Consultar notas> Consultar notas
<outputtext ("textSize") name= id.curso> id.curso />
<outputtext ("textSize")  name= Nombre del curso> Nombre del curso />
<outputtext ("textSize") name= id. evaluacion> id. evaluacion />
<outputtext ("textSize") name= calificacion> Calificacion />
<window>
```

Se examinan los elementos generados en la siguiente tabla.

Interfaz: Nueva evaluación	**Usuario: Pedro**			
Guías del usuario	Ejecución	Elementos a reemplazar	Elementos reemplazados	Aceptación
4	Si	id.curso, Nombre del curso, id.evaluacion	id.curso, Nombre del curso, id.evaluacion	Si
5	No			No

- **Nueva evaluación**

Interfaz original

```
<window name=Nueva evaluacion>Nueva evaluación
<datepicker name= fecha> fecha / >
<outputtext name= id. evaluacion> id. evaluacion />
<outputtext name= Nombre del curso> Nombre del curso />
<listbox name= Tipo de evaluacion >Tipo de evaluación/>
<listbox name= tiempo de la evaluacion> Tiempo de la evaluación />
<listbox name= Numero de preguntas> Número de preguntas />
<button name=Crear evaluacion> Crear evaluación />
<window>
```

Resultado obtenido luego de ejecutar las reglas:

```
<window name=Nueva evaluacion>Nueva evaluación
<datepicker name= Seleccionar fecha> Seleccionar fecha />
<outputtext ("textSize") name= id.evaluacion> id.evaluacion />
<outputtext ("textSize") name= Nombre del curso> Nombre del curso />
<listbox name= tipo de evaluacion >Tipo de evaluación />
<listbox name= tiempo de la evaluacion> Tiempo de la evaluación/>
<listbox name=cantidad de preguntas> Cantidad de preguntas/>
<button name=Crear evaluacion> Crear evaluación />
<window>
```

Se examinan ahora, cada uno de los elementos de la interfaz anterior. Los resultados observados son los que se esperaban.

Interfaz: Nueva evaluacion	**Usuario: Pedro**			
Guías	Ejecución	Elementos a reemplazar	Elementos reemplazados	Aceptación
4	Si	id.evaluacion, Nombre del curso	id.evaluacion, Nombre del curso	Si
5	No			No

IV.5.1.2. Dimensión de tareas

Indicador: *Aceptación de las tareas.*

Descripción: Para determinar la completitud se generaron todas las interfaces posibles para cada usuario y se analiza la correspondencia de los elementos o interfaces generados con las tareas instanciadas en el modelo de tareas.

Tareas	Elementos concretos generado	Aceptación
Listado de Cursos	<window name=*Ver Listado de Cursos*>Ver Listado de Cursos<window>	Si
Consultar notas	• <window name=*Consultar notas*>Consultar notas<window> • <outputtext url=*Consultar notas* ("IsUnderlined") name= *Consultar notas*> Consultar notas />	Si
Nueva evaluación	• <window name=*Nueva evaluacion*>Nueva evaluación<window> • <outputtext url=*Nueva evaluacion* ("IsUnderlined") name= *Nueva evaluacion*> Nueva evaluación />	Si
Seleccionar fecha	<datepicker name=*Seleccionar fecha*> Seleccionar fecha />	Si
Seleccionar tipo	<listbox name=*Seleccionar tipo de evaluacion*>Seleccionar tipo de evaluación />	Si
Seleccionar cantidad de preguntas	<listbox name=*Seleccionar cantidad de preguntas*> Seleccionar cantidad de preguntas />	Si
Seleccionar tiempo	<listbox name= *Seleccionar tiempo de la evaluacion*>Seleccionar tiempo de la evaluación />	Si
Tomar evaluación	<window name=*Tomar evaluacion*>Tomar evaluación<window>	Si

Como se observa en la tabla anterior cada una de las tareas del modelo de tareas son representadas una o más veces en las interfaces generadas. Por lo tanto se logra la aceptación de las tareas.

IV.5.1.3. Dimensión de dominio

Indicador: *Aceptación de los elementos del dominio.*

Descripción: Para determinar la completitud se generaron todas las interfaces posibles para cada usuario y se analiza la correspondencia de los elementos generados en ellas con los elementos de dominio instanciados en el modelo de dominio.

Elementos del dominio	Elementos concretos generados	Aceptación
Nombre del curso	<outputtext name= *Nombre del curso*> Nombre del curso </outputtext >	Si
id.curso	<outputtext name= *id.curso*> id.curso </outputtext >	Si
id.evaluación	<outputtext name= *id.evaluacion*> id.evaluacion</outputtext>	Si
Fecha	<outputtext ("textSize") name= *fecha*> Fecha </outputtext>	Si
Tiempo de la evaluación	<timesensor name= *tiempo de la evaluacion*> Tiempo de la evaluación </timesensor>	Si
Número de preguntas	<outputtext ("textSize") name= *Numero de preguntas*>Número de preguntas </outputtext>	Si
Nombre del curso	<outputtext name= *Nombre del curso*> Nombre del curso </outputtext>	Si
Tipo de evaluación	<outputtext ("textSize") name= *Tipo de evaluacion*>Tipo de evaluación </outputtext>	Si
Calificación	<outputtext name= *calificacion*> Calificación </outputtext>	Si
Crear evaluación	<outputtext url=*Nueva evaluacion* ("IsUnderlined") name= *Nueva evaluacion*>Nueva evaluación </outputtext>	Si
Ver notas	<outputtext url=*Consultar notas* ("IsUnderlined") name= *Consultar notas*> Consultar notas </outputtext>	Si

Como se observa en la tabla anterior cada uno de los elementos del modelo de dominio son representados una o más veces en las interfaces generadas.

IV.5.2. CORRECTITUD

Indicador: Grado de concordancia del modelo concreto con los modelos de usuario, de tareas y de dominio.

Descripción: Para determinar la correctitud se analizan las interfaces concretas generadas por el prototipo para las tareas y elementos instanciados en los modelos y se analiza si concuerdan con los resultados esperados.

Tarea y elemento del dominio	Salida Esperada	Salida Obtenida
Ver listado de Cursos	<window name=*Listado de Cursos*>Mis cursos</window>	<window name=*Listado de Cursos*>Mis cursos</window>
Nueva evaluación	<window name=*Nueva evaluacion*>Agregando cuestionario</window>	<window name=*Nueva evaluacion*>Agregando cuestionario</window>
Seleccionar fecha	<datepicker name=*fecha*>fecha de evaluación/>	<datepicker name= *fecha*> fecha de evaluación / >
Seleccionar tipo	<listbox name=*Tipo de evaluacion* >Tipo de evaluación / >	<listbox name= *Tipo de evaluacion* >Tipo de evaluación / >
Seleccionar cantidad de preguntas	<listbox name= *Numero de preguntas*> Número de preguntas/>	<listbox name= *Numero de preguntas*> Número de preguntas/>
Consultar notas	<window name=*Consultar notas*>Calificaciones </window>	<window name=*Consultar notas*>Calificaciones </window>
Tomar evaluación	<window name=*Tomar evaluacion*>Nombre de evaluación</window>	<window name=*Tomar evaluacion*>Nombre de evaluación</window>
Nombre del curso	<outputtext name= *Nombre del curso*> Nombre del curso />	<outputtext name= *Nombre del curso*> Nombre del curso />
id.curso	<outputtext name= *id.curso*> id.curso />	<outputtext name= *id.curso*> id.curso />
id.evaluacion	<outputtext name= *id. evaluacion*> id. evaluacion />	<outputtext name= *id. evaluacion*> id. evaluacion />
Fecha	<outputtext name= *fecha*> fecha />	<outputtext name= *fecha*>fecha/>
Tiempo de la evaluación	<timesensor name= *tiempo de la evaluacion*> tiempo de la evaluación/>	<timesensor name= *tiempo de la evaluacion*> tiempo de la evaluación / >
Número de preguntas	<listbox name= *Numero de preguntas*> Número de preguntas />	<listbox name= *Numero de preguntas*> Número de preguntas/>
Tipo de evaluación	<outputtext name= *Tipo de evaluacion*>Tipo de evaluación />	<outputtext name= *Tipo de evaluacion*>Tipo de evaluación />
Pregunta	<outputtext name= Pregunta >Pregunta/>	<outputtext name= Pregunta > Pregunta />
Nombre del curso	<outputtext name= *Nombre del curso*> Nombre del curso />	<outputtext name= *Nombre del curso*> Nombre del curso />
Calificación	<outputtext name= *calificacion*> calificación />	<outputtext name= *calificacion*> calificación />

Como se observa en la tabla anterior en el 100% de los casos se obtuvieron los resultados esperados.

IV.5.3. USABILIDAD

A continuación se detallan los métodos necesarios para probar el siguiente objetivo:
***Usabilidad**: que posea herramientas que faciliten el aprendizaje y modificación de las funcionalidades del prototipo.*

IV.5.3.1. Aprendibilidad

Para corroborar que se cumple la facilidad de aprendizaje (*aprendibilidad*) se tomó como base una lista de comprobación desarrollada en el trabajo *Características Blandas De La Calidad De Los SI* [GAL06] la cual fue adaptada a las necesidades de este trabajo.
Para recolectar información acerca de la aprendibilidad se desarrollaron encuestas con el formato que se observa en la siguiente página.
Los formularios de la encuesta se crearon en forma digital y se los envió por mail a los encuestados. Además se les entregó el manual de usuario y la herramienta para que realicen las pruebas correspondientes.
Los resultados se almacenaron automáticamente en una hoja de cálculo, de la que luego se pudieron obtener las gráficas y los porcentajes.
A continuación se observa el cuestionario resultante con los indicadores correspondientes a la característica *aprendibilidad*.

Cuestionario de Medición de Aprendibilidad.

Para cada elemento identificado a continuación, seleccione el número que considere más acorde con su criterio. La escala a tener en cuenta es 1: completamente en desacuerdo, 2: algo en desacuerdo, 3: ni en desacuerdo ni acuerdo, 4: algo en acuerdo, 5: totalmente de acuerdo. **Obligatorio*

Datos Generales

Nombre Completo: _____

E1. Profesión/Especialización *: _____

Conocimientos Previos

E2. ¿Tiene conocimientos previos de diseño de interfaces de usuario o sobre UsiXML?*

Si		No	

E3. ¿Tiene conocimientos sobre ontologías? *

Si		No	

E4. ¿Posee conocimientos sobre diseño universal? *

Si		No	

Evaluación del Prototipo

E5. Es el sistema fácil de usar sin conocimientos especiales de diseño de interfaces de usuario. *

1	2	3	4	5

E6. Es el sistema fácil de usar sin conocimientos especiales de diseño de interfaces de usuario con UsiXML. *

1	2	3	4	5

E7. Los usuarios con conocimientos intermedios de diseño de interfaces de usuario requieren escaso tiempo (menos de diez días) para aprender a usar el sistema. *

1	2	3	4	5

E8. La documentación disponible esta expresada de una manera consistente con la terminología del usuario. *

1	2	3	4	5

E9. Las ayudas están expresadas de modo que el usuario puede solucionar sus problemas después de consultar las mismas (parcial o totalmente). *

1	2	3	4	5

E10. El prototipo proporciona algún método de entrenamiento que soporte el proceso de aprendizaje en el manejo del mismo. *

1	2	3	4	5

E11. Se ha previsto en el prototipo la recepción de consultas de usuarios. *

1	2	3	4	5

E12. Se ha implementado en el prototipo algún mecanismo de rastreo, monitoreo y registro de los errores más frecuentes cometidos por usuarios. *

1	2	3	4	5

Resultados obtenidos en las encuestas

Los resultados enviados por los encuestados se pueden observar en la tabla. Luego se analizan en base a las estadísticas y porcentajes obtenidos.

E1	LSI	Estudiante	Ing. en computación	LSI	LSI	Profesora de informática	Informática
E2	Si	No	Si	No	No	No	Si
E3	Si	Si	No	No	No	No	No
E4	Si	No	No	Si	Si	Si	No
E5	5	5	5	4	3	4	4
E6	5	3	5	5	3	4	3
E7	4	5	5	5	5	4	5
E8	5	5	5	5	4	3	4
E9	5	2	3	5	4	4	4
E10	5	1	5	4	3	5	4
E11	5	1	3	5	5	4	3
E12	5	2	3	5	3	4	3

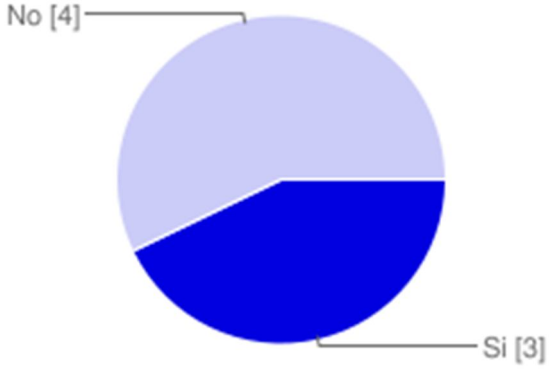
LSI: Licenciado en Sistemas de Información.

RESUMEN DE RESULTADOS

A continuación se presentan en gráficos los resultados obtenidos en las encuestas realizadas. Indicando el porcentaje y la cantidad de cada opción seleccionada por los encuestados

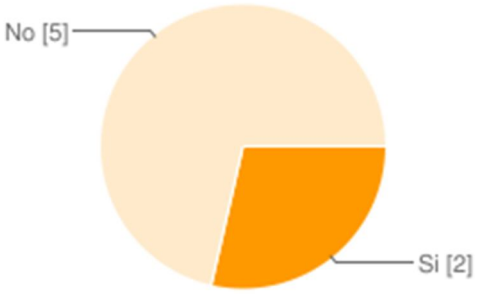
CONOCIMIENTOS PREVIOS

¿Tiene conocimientos previos de diseño de interfaces de usuario o sobre UsiXML?



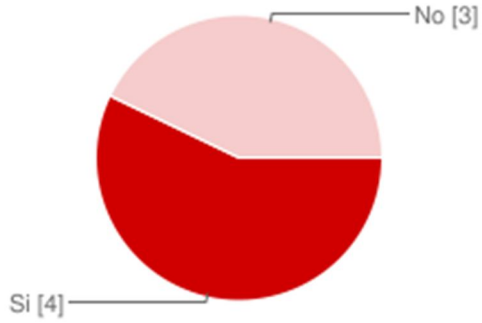
Si	**3**	**43%**
No	4	57%

¿Tiene conocimientos sobre ontologías?



Si	2	29%
No	5	71%

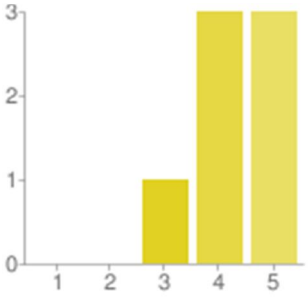
¿Posee conocimientos sobre diseño universal?



Si	4	57%
No	3	43%

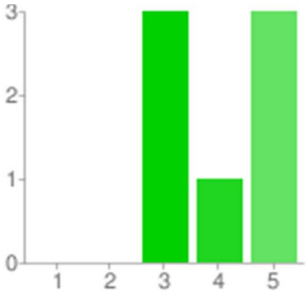
EVALUACIÓN DEL PROTOTIPO

Es el sistema fácil de usar sin conocimientos especiales de diseño de interfaces de usuario.



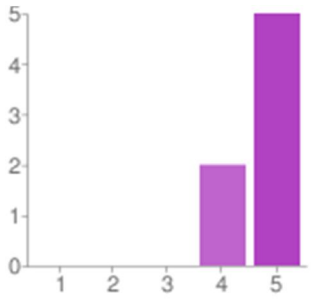
1: Completamente en desacuerdo	0	0%
2: Algo en desacuerdo	0	0%
3: Ni en desacuerdo ni acuerdo	1	14%
4: Algo en acuerdo	3	43%
5: Totalmente de acuerdo	3	43%

Es el sistema fácil de usar sin conocimientos especiales de diseño de interfaces de usuario con UsiXML.



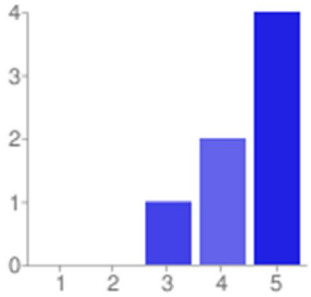
1: Completamente en desacuerdo	0	0%
2: Algo en desacuerdo	0	0%
3: Ni en desacuerdo ni acuerdo	3	43%
4: Algo en acuerdo	1	14%
5: Totalmente de acuerdo	3	43%

Los usuarios con conocimientos intermedios de diseño de interfaces de usuario requieren escaso tiempo (menos de diez días) para aprender a usar el sistema.



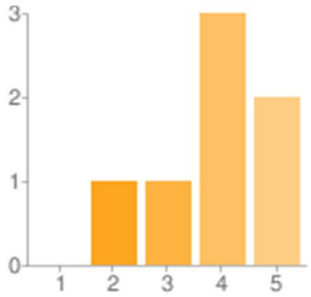
1: Completamente en desacuerdo	0	0%
2: Algo en desacuerdo	0	0%
3: Ni en desacuerdo ni acuerdo	0	0%
4: Algo en acuerdo	2	29%
5: Totalmente de acuerdo	5	71%

La documentación disponible esta expresada de una manera consistente con la terminología del usuario.



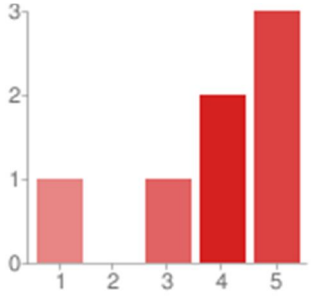
1: Completamente en desacuerdo	0	0%
2: Algo en desacuerdo	0	0%
3: Ni en desacuerdo ni acuerdo	1	14%
4: Algo en acuerdo	2	29%
5: Totalmente de acuerdo	4	57%

Las ayudas están expresadas de modo que el usuario puede solucionar sus problemas después de consultar las mismas (parcial o totalmente).



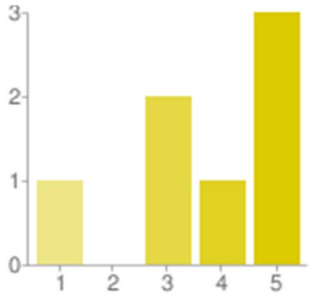
1: Completamente en desacuerdo	0	0%
2: Algo en desacuerdo	1	14%
3: Ni en desacuerdo ni acuerdo	1	14%
4: Algo en acuerdo	3	43%
5: Totalmente de acuerdo	2	29%

El prototipo proporciona algún método de entrenamiento que soporte el proceso de aprendizaje en el manejo del mismo.



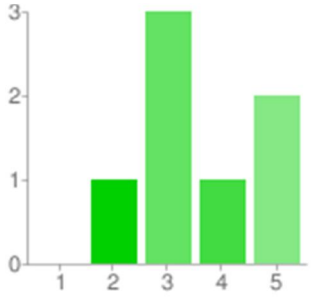
1: Completamente en desacuerdo	1	14%
2: Algo en desacuerdo	0	0%
3: Ni en desacuerdo ni acuerdo	1	14%
4: Algo en acuerdo	2	29%
5: Totalmente de acuerdo	3	43%

Se ha previsto en el prototipo la recepción de consultas de usuarios.



1: Completamente en desacuerdo	**1**	**14%**
2: Algo en desacuerdo	0	0%
3: Ni en desacuerdo ni acuerdo	2	29%
4: Algo en acuerdo	1	14%
5: Totalmente de acuerdo	3	43%

Se ha implementado en el prototipo algún mecanismo de rastreo, monitoreo y registro de los errores más frecuentes cometidos por usuarios.



1: Completamente en desacuerdo	**0**	**0%**
2: Algo en desacuerdo	1	14%
3: Ni en desacuerdo ni acuerdo	3	43%
4: Algo en acuerdo	1	14%
5: Totalmente de acuerdo	2	29%

ANÁLISIS DE RESULTADOS

Un 57% de los encuestados contestó que tiene *conocimientos sobre usiXML*, un 71% tiene *conocimientos sobre ontologías*. Mientras que sólo el 43% *conoce sobre diseño universal*.

A pesar de esto un 86% admitió que *se puede aprender a usar el prototipo sin tener conocimientos especiales sobre interfaces de usuario* (diseño universal, modelado, etc.)

El 100% de los encuestados opinó que de *poseer conocimientos intermedios se podría aprender a utilizar la herramienta en menos de 10 días*.

El 86% de opinó que *la documentación entregada (manual de usuario) estaba escrita de manera clara y al nivel de los conocimientos de los usuarios*. El 72% declaró que *se pueden solucionar problemas después de consultar la documentación*.

El 72% creyó que *el prototipo posee un método de entrenamiento que soporte el proceso de aprendizaje en el manejo del mismo*. En este trabajo se produjo como método para soportar el proceso de aprendizaje, el manual de usuario. La producción de otros métodos que soporten el proceso de aprendizaje se puede implementar en trabajos futuros.

El 57% dijo que *se ha previsto en el prototipo la recepción de consultas de usuarios.* Se considera en este caso el envío de consultas a través de correo electrónico a los desarrolladores del prototipo. Se prevén para futuros trabajos otros métodos de recepción de consultas.

Finalmente sólo el 43% estuvo de acuerdo con que se ha *implementado en el prototipo algún mecanismo de rastreo, monitoreo y registro de los errores más frecuentes cometidos por usuario.* En este caso el prototipo no posee ningún mecanismo incorporado. Queda para trabajos futuros implementar funciones que permitan plasmar los errores más frecuentes de los usuarios en el uso del prototipo.

IV.5.3.2. Extensibilidad

Para determinar esta dimensión se utilizó una segunda lista, esta se basa en el trabajo del proyecto *OPSOA* [MON09] y fue modificada para adaptarse a la línea de este trabajo. Esta lista permitió analizar la característica mencionada en el objetivo como *modificación de las funcionalidades del prototipo* (extensibilidad).

Preguntas
¿La estructuración de la ontología permite modificarla con facilidad?
¿El código de las reglas es correcto, está comentado y permite modificaciones con facilidad?
¿Existe documentación para desarrolladores que ayude a entender el código y los mecanismos de extensión?

RESPUESTAS

¿La estructuración de la ontología permite modificarla con facilidad?

Como se explicó anteriormente, una de las características principales de una ontología es que define un vocabulario común para personas que necesitan compartir información de un dominio en particular. La motivación en la creación de ontologías es la de poder compartir y reutilizar el conocimiento de un dominio.

En este caso la ontología reúne terminología sobre el dominio del modelado de interfaces, la cual utiliza la terminología propuesta por un estándar como UsiXML. Dicha ontología como el estándar pueden ser descargados gratuitamente y pueden ser modificados libremente.

Además del requisito fundamental de ser de libre acceso, es necesario contar con herramientas que faciliten la manipulación de estos archivos. Para esto se trabajó con Protégé, ya que gracias a la interfaz gráfica del este gestor de ontologías, fue posible realizar, de forma muy sencilla, las tareas de consulta, modificación y entrada de conceptos en la ontología.

¿El código de las reglas es correcto, está comentado y permite modificaciones con facilidad?

Para la creación de las reglas se recurrió a un motor de reglas de transformación como es Jess. La empresa desarrolladora también creó un plug-in para Protégé llamado JessTab el cual permite utilizar Jess y Protégé juntos. Este plug-in brinda una consola en donde se puede interactuar con Jess dentro de Protégé. JessTab extiende Jess con funciones adicionales que permiten manipular las bases de conocimiento.

Para expresar la lógica en forma de reglas se utilizó el lenguaje de reglas provisto por Jess. También existe la posibilidad de expresar en otros formatos, pero en este trabajo se utilizó el primero mencionado.

El código de las reglas se lo dividió en dos archivos, uno en donde se encuentran las reglas que generan el modelo abstracto y el segundo en donde se encuentran las que generan el modelo concreto. Los archivos poseen extensión "clp" y existe una función mediante la cual se puede cargar y ejecutar los archivos desde la consola de JessTab. Estos archivos pueden editarse mediante cualquier editor de texto. El archivo que genera el modelo abstracto posee unas 60 líneas contando los comentarios. Para poder actualizar el código es necesario poseer un conocimiento sobre el modelado de interfaces con UsiXML, conocimiento sobre ontologías y sobre el lenguaje de reglas que brinda Jess. A pesar de la poca cantidad de líneas de código, el lenguaje es de muy bajo nivel, lo cual dificulta la tarea de programación.

¿Existe documentación para desarrolladores que ayude a entender el código y los mecanismos de extensión?

Se ha documentado en el Anexo A toda la información que detalla el código de cada una de las reglas explicando el significado de cada una de las variables así como también el porqué de su estructuración.

En cuanto a las funciones y métodos de Jess, existe una documentación online que detalla cada una de ellas y su significado.

CONCLUSIONES

Actualmente cada vez son más los usuarios que usan aplicaciones informáticas para el trabajo, estudio, compras, etc. Estos usuarios presentan una variedad de características, requisitos, roles, etc. que la interfaz de usuario debe satisfacer para facilitar la comunicación entre la persona y el sistema de información.

Este crecimiento, tanto en cantidad como en variedad, ha provocado que se intensifique el interés y estudio en el diseño de Interfaces de Usuario y, en particular, por el Diseño Universal de Interfaces de Usuario, cuyo objetivo es lograr múltiples IU, en lugar de una IU común para todos.

Sin embargo, llevar a cabo estas interfaces requiere de un esfuerzo considerable por parte del diseñador ya que se debe tomar en cuenta una gran cantidad de parámetros y son numerosas las relaciones a considerar.

En este trabajo, se presentó una solución para el diseño universal de IU mediante el desarrollo de un prototipo que toma como base los "Entornos de Desarrollo de Interfaces de Usuario basados en Modelos" (MB-UIDE) y el uso de ontologías.

A lo largo del proceso de construcción del prototipo, se intentó cumplir con los objetivos propuestos al comienzo del trabajo, y finalmente creemos que contribuiremos con una herramienta que:

- Permite implementar IU de manera profesional, de forma sistemática en menor tiempo y costo.
- Facilita la manipulación de una amplia cantidad de usuarios con múltiples características.
- Produce múltiples modelos de IU concreta para la satisfacción de las preferencias y características de cada uno de los usuarios, en lugar de de producir una IU promedio para todos los usuarios. Lo que permite lograr IU de alta calidad, accesibles y usables admitiendo una personalización de las mismas.

Todo esto en virtud de que en este prototipo:

- Se trató de incluir la mayor cantidad de características, roles, preferencias, etc. de los usuarios reusando la ontología GUMO.

- Se desarrolló un conjunto de reglas de transformación que permiten automatizar las tareas de generación de los modelos necesarios (de interfaz de usuario abstracta y concreta) para el diseño una IU.
- Se logró simplicidad en el lenguaje usado para describir los modelos, lo que facilita su aprendizaje y uso.

O sea, que en todo momento, se intentó no sólo responder a los criterios de un buen diseño, sino, sobre todo, responder de manera clara y precisa a los requerimientos de los diseñadores de interfaces de usuarios.

Con respecto a la verificación se implementaron pruebas concienzudas para lograr, por un lado, la correctitud, o sea, la representación correcta de las especificaciones de los modelos en las interfaces finales. Por otro lado, las características de completitud y usabilidad.

Finalmente las conclusiones arribadas en este trabajo son:

- ✱ El prototipo generó interfaces manipulando múltiples parámetros de los diferentes modelos de entrada (modelo de tareas, dominio y usuario) considerando las vinculaciones entre ellos.
  Para cada uno de los usuarios, las interfaces obtenidas fueron aquellas a las cuales les estaba permitido acceder.
  Reflejaron las características físicas, cognitivas y demográficas de los usuarios.
- ✱ Los elementos y el contenido de cada una de las interfaces obtenidas se correspondieron en un 100% con la información de los modelos.
  Los conocimientos del usuario del prototipo sobre la especificación de usiXML volcados en los modelos producen elementos concretos particulares, los cuales forman las interfaces concretas.
  Dichas interfaces se generaron con todos los elementos concretos esperados.
  También se abstrajo correctamente la información contenida en el modelo de dominio.
- ✱ La reutilización de diferentes componentes que forman el prototipo como editores de ontologías, plugins y ontologías reusables permitiría acelerar el proceso de aprendizaje del mismo.

Por último, pensamos que sería interesante que en otros trabajos:

1) Se mejore la aprendibilidad para el usuario del prototipo. Aumentando y mejorando la documentación de la misma. Proporcionando un método de entrenamiento que soporte el proceso de aprendizaje en el manejo del mismo.
2) Se aumente la cantidad de características de los usuarios y guías de diseño. Al aumentar la cantidad de características de los usuarios se incrementaría la complejidad de las interfaces a generar. Además, se debería aumentar la cantidad de guías de diseño que consideren las nuevas características de los usuarios.

REFERENCIAS BIBLIOGRÁFICAS

[ACM09] ACM SIGCHI. Curricula for Human-Computer Interaction. old.sigchi.org/cdg/cdg2.html. Fecha de acceso: 20 de Mayo de 2009.

[BAR07] Barchini, G; Alvarez, M; Herrera, S; Trejo, M. El rol de las ontologías en los SI. Revista Ingeniería Informática, edición 14, 2007.

[BOR97] Borst, W. N., 1997. "Construction of Engineering Ontologies". PhD thesis, University of Tewnte, Enschede, 1997.

[BUI97] Buie, E., A., & Vallone, A. Integrating HCI engineering with software engineering: A call to a larger vision. In Smith, M. J., Salvendy, G., & Koubek, R. J. (Eds.), Design of Computing Systems: Social and Ergonomic Considerations (Proceedings of the Seventh International Conference on Human-Computer Interaction), Volume 2. Amsterdam, the Netherlands: Elsevier Science Publishers, 1997, pp. 525-530. http://www.luminanze.com/writings/larger_vision.html. Fecha de acceso: 1 de Junio de 2009.

[CON96] Constantine, L. (1996) "Abstract Objects," Object Magazine, 6, (10) December. Rprinted in L. Constantine, The Peopleware Papers (Prentince Hall, 2001).

[DEL07] Delgado González, Antonio Luis. "Generación de Interfaces Web basada en Modelos". Escuela Técnica Superior de Ingeniería Informática-Universidad de Sevilla Departamento de Lenguajes y Sistemas Informáticos. 2007.

[FUR07] Furtado E., Furtado V., Silva, W., Rodrigues, D., Taddeo, L., Limbourg Q., and Vanderdonckt J. "An Ontology-Based Method for Universal Design of User Interfaces. Proceedings of Workshop on Multiple User Interfaces over the Internet: Engineering and applications Trends". Disponible en: http://www.cs.concordia.ca/%7Efaculty/seffah/ihm2001/program.html. Fecha de acceso: 27 de Diciembre de 2007.

[FUR04] Furtado, E., Furtado, V., Soares Sousa, K., Vanderdonckt, J., Limbourg, Q., KnowiXML: A Knowledge-Based System Generating Multiple Abstract User Interfaces in UsiXML, Proc. of 3rd Int. Workshop on Task Models and Diagrams for user interface design TAMODIA'2004 (Prague, November 15-16, 2004), Ph. Palanque, P. Slavik, M. Winckler (eds.), ACM Press, New York, 2004, pp. 121-128.

[GOM99] Gómez-Pérez, A. "Ontological Engineering: A State Of The Art". Expert Update. British Computer Society. Autumn. Vol. 2. Nº 3. 1999.

[GRU95] Gruber, T. "Towards Principles for the Design of Ontologies used for Knowledge Sharing". International Journal of Human-Computer Studies, 43(5/6), pp. 907-928. 1995.

[GRU96] Grüninger, M. "Designing and Evaluating Generic Ontologies". Proceedings of the 12thEuropean Conference of Artificial Intelligence (pp. 53-65). 1996.

[GRU94a] Grüninger, M., Fox, M. S. "The Design and Evaluation of Ontologies for Enterprise Engineering". Proceedings of the Workshop on Implemented Ontologies, European Conference on Artificial Intelligence. 1994.

[GRU94] Grüninger, M., Fox, M. S. "The Role of Competency Questions in Enterprise Engineering". Proceedings of the IFIP WG5.7 Workshop on Benchmarking - Theory and Practice. 1994.

[HEC05] Heckmann, Dominik; Schwartz, Tim; Brandherm, Boris; Schmitz, Michael; von Wilamowitz-Moellendorff, Margeritta: GUMO – "the General User Model Ontology". 2005

[HEN06] Henrik Eriksson, Dept. of Computer and Information Science, Linköping University. 2006. http://www.ida.liu.se/~her/JessTab/ Fecha de acceso 26 de junio de 2009

[INT06] International Journal of Web Engineering and Technology. Raising the Level of Abstraction of User Interface Specification with XML User Interface Description Languages. 2006. www2.edm.uhasselt.be/xmluidl2006. Accedido: 26 de agosto de 2009.

[KOY04] Koyani , Sanjay J.; Bailey, Robert W.; Nall, Janice R. Research-Based Web Design & Usability Guidelines. 2004.

[LEI04] Lei Y., Motta E., and Domingue J., Modelling Data-Intensive Web Sites with OntoWeaver, in proceedings of the International Workshop on Web Information Systems Modelling (WISM 2004), Riga, Latvia, 2004.

[LIM04] Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Víctor López Jaquero, UsiXML: a Language Supporting Multi-Path Development of User Interfaces, Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems, EHCI-DSVIS'2004 (Hamburg, July 11-13, 2004), Lecture Notes in Computer Science, Vol. 3425, Springer-Verlag, Berlin, 2005, pp. 200-220.

[LOR02] Lorés, Jesús; Granollers, Toni; Lana, Sergi. Introducción a la Interacción Persona-Ordenador. Universitat de Lleida. 2002.

[MEN03] Mendoza, Jorge A. e-Learning, el futuro de la educación a distancia. 2003. http://www.informaticamilenium.com.mx/paginas/mn/articulo78.htm. Fecha de acceso: 06 de noviembre de 2009.

[MIZ95] Mizoguchi, R. Vanwelkenhuysen, J. and Ikeda M. "Task ontology for reuse of problem solving knowledge". In 2nd International Conference on Very Large-Scale Knowledge Bases., pages 46–57. IOS Press, Amsterdam, NL. 1995.

[MOD09] Model-based User Interfaces Incubator Group. Abstraction Levels in Interactive Systems. 2009.
http://www.w3.org/2005/Incubator/model-based-ui/wiki/Layered_Architecture. Fecha de accesso: 29 de Junio de 2009.

[MOR81] Moran T. P. "The command language grammar: a representation for the user interface of interactive systems" en International Journal of man machine studies, núm. 15, 1981

[MYE92] Myers, B.A and Rosson, M.B. "Survey on User Interface Programming". Human Factors in Computing Systems. Proc. SIGCHI'92, pp. 195-202, 1992.

[NIE01] Nielsen, Jakob; Pernice, Kara. Beyond ALT Text: Making the Web Easy to Use for Users with Disabilities. Octubre 2001 http://www.nngroup.com/reports/

[NOY05] Noy, Natalya F.; McGuinness, Deborah L. "Desarrollo de Ontologías-101: Guía Para Crear Tu Primera Ontología". Stanford University, Stanford. 2005

[PAR83] Partsch, H., Steinbruggen, R.: Program Transformation Systems. ACM Computing Surveys 15,3 (September 1983), 199–236

[PLA09] Plataforma de e-Learning Moodle. http://docs.moodle.org/es/Portada. Fecha de acceso: 19 de Mayo de 2009.

[PRO09] Protégé. What is Protégé. http://protege.stanford.edu/overview/. Fecha de acceso: 24 de Mayo de 2009.

[PUE97] Puerta, Angel R. "A Model-Based Interface Development Environment. Stanford University". 1997.

[QUE04] Quentin Limbourg and Jean Vanderdonckt. UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence. Articles of Workshop on Device Independent Web, 2004.

[ROZ97] Rozenberg, G. (ed.). Handbook of Graph Grammars and Computing by Graph Transformation. World Scientific, Singapore (1997)

[SAN08] Sandia National Laboratories. http://www.jessrules.com/ 2008. Fecha de acceso: 29 de Mayo de 2009.

[SAN07] Sandoval Cantor, Ana Efigenia. Uso de ontologías y web semántica para apoyar la gestión del conocimiento. Ciencia e Ingeniería Neogranadina, Vol. 17-2, pp. 111-129. Bogotá. ISSN 0124-8170. 2007.

[SAV01] Savidis, A., Akoumianakis, D., Stephanidis, C. The Unified User Interface Design Method, Chapter 21, in "User Interfaces for All: Concepts, Methods, and Tools " C. Stephanidis (ed.), Lawrence Erlbaum Associates, Pub. pp. 417-440. Mahwah. 2001.

[SOW00] Sowa, J F. "Knowledge Representation: Logical, Philosophical, and Computational Foundations", Brooks Cole Publishing Co., Pacific Grove, CA 2000. Disponible parcialmente en: http://www.jfsowa.com/ontology/. Fecha de acceso: 10 de Diciembre de 2008.

[STU98] Studer, R.; Benjamins V. R. and Fensel D. "Knowledge engineering: principles and methods". Data and Knowledge Engineering, 25 (1-2): 161-197, 1998.

[THE09] The usability professionals' association. What is User-centered Design? http://www.upassoc.org/usability_resources/about_usability/what_is_ucd.html. Fecha de acceso: 12 de Mayo de 2009.

[USI07] UsiXML Consortium. UsiXML V1.8. UsiXML, USer Interface eXtensible Markup Language. 2007.

[VAN08] Vanderdonckt, Jean. Model-Driven Engineering of User Interfaces: Promises, Successes, Failures, and Challenges. 2008.

[WIK09] Wikipedia. Interacción persona-computador. http://es.wikipedia.org/wiki/Interacción_persona-computador. Fecha de acceso: 12 de Julio de 2009.

Anexo A
Reglas de Transformación

ANEXO A

REGLAS DE TRANSFORMACIÓN

A.1. PROGRAMACIÓN

Las reglas que se utilizaron para procesar los modelos de tareas, dominio y sus relaciones fueron definidas usando el Plug-in de Reglas Jess, las reglas permitirán al diseñador derivar hacia el modelo de interfaz de usuario abstracta, imagen A.1.

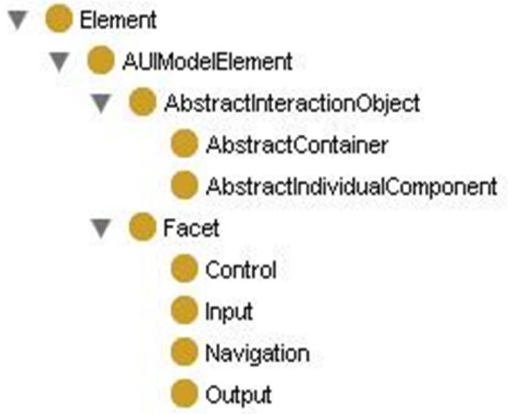


Imagen A.1. Modelo de interfaz de usuario abstracta

Primero se debe crear el conjunto de reglas necesarias para producir el modelo de interfaz de usuario abstracta. Comenzando con los elementos *AbstractContainer y AbstractIndividualComponent* para seguir con los *Facets* que incluyen los elementos *Output*, *Input*, *Navigation* y *Control*.

AbstractContainer y AbstractIndividualComponent

Para crear los AbstractContainer se aplica la siguiente regla tomada de UsiXML: *Si dos tareas (T1 y T2 pertenecientes a TaskModel) están relacionadas mediante un tipo de relación binaria de habilitación (Enabling) o de selección (Choice), y la relación unaria de una tarea (T1) es "Iteración finita" (FiniteIteration) y la relación unaria de la otra tarea relacionada (T2) es "opcional" (Optional) entonces crear un "Contenedor Abstracto" (AbstractContainer).*

$\mathcal{R}$ = relación

Si T1 $\mathcal{R}$ T2 ^ ($\mathcal{R}$ = Enabling **v** $\mathcal{R}$ = Choice) ^ T1 $\mathcal{R}$ T1^ ($\mathcal{R}$ = finiteIteration) ^ T2 $\mathcal{R}$

T2 ^ ($\mathcal{R}$ = Optional) => instanciar clase AbstractContainer

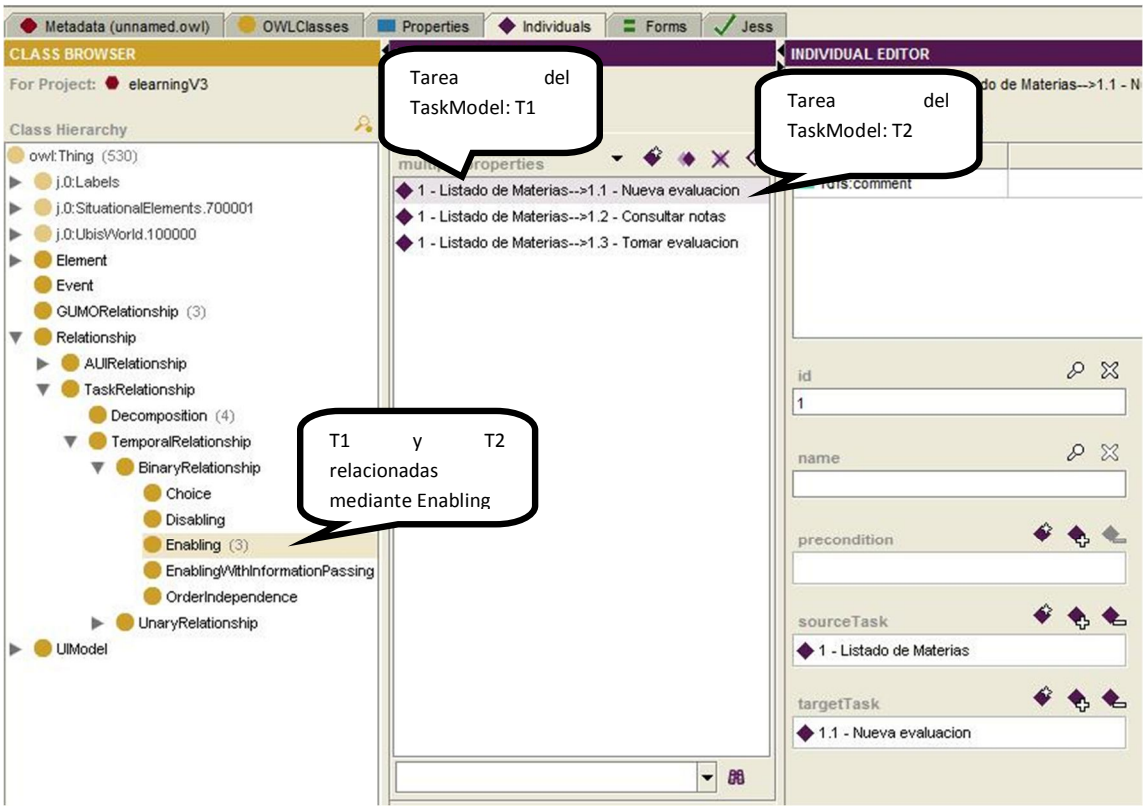


Imagen A.2. Instancias de las relaciones binarias *Enabling*.

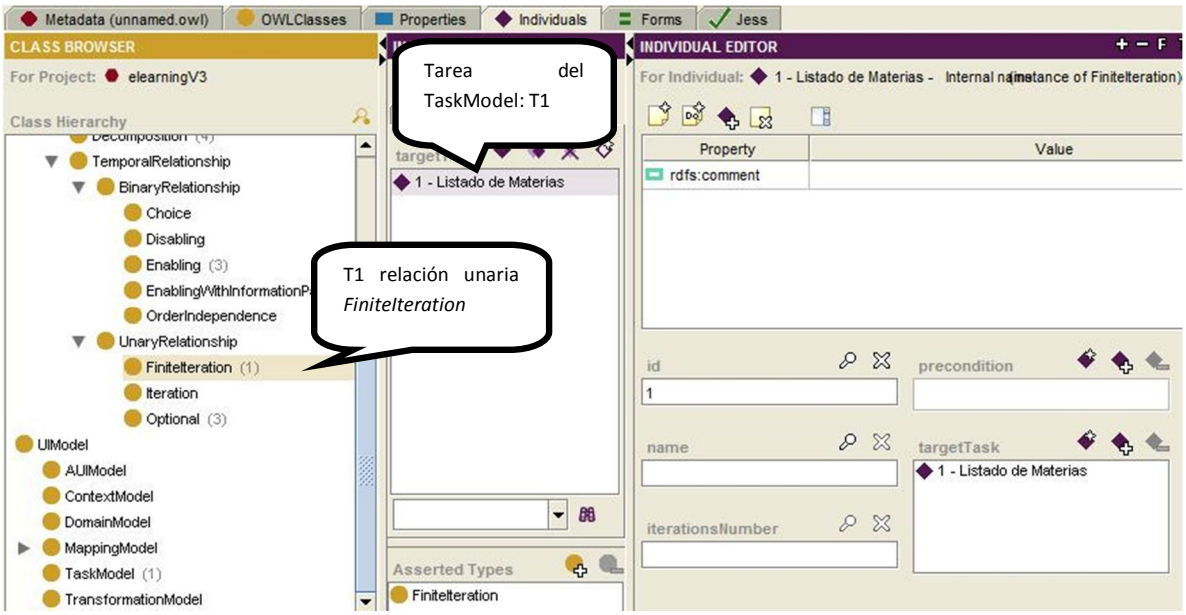


Imagen A.3. Instancias de las relaciones unarias *FiniteIteration*.

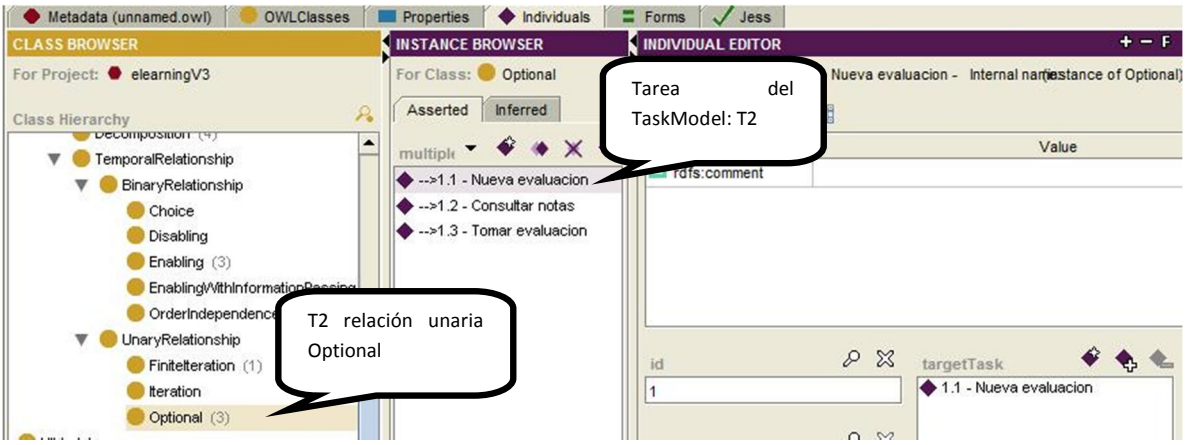


Imagen A.4. Instancias de las relaciones unarias *Optional*.

En primer lugar se recorren las relaciones binarias de tipo *Enabling* y *Choice*. De cada una de ellas se toman los atributos *id*, *sourceTask* y *targetTask* (observar imagen A.2). Los últimos dos atributos son referencias a instancias del modelo de tareas. De las tareas a las cuales hacen referencia se obtiene el nombre y su relación unaria en un hecho (fact) que equivaldría a un array de información.

```
(defrule RelacionesBinarias ;Nombre de la regla
 (object
   (is-a Enabling|Choice);Relaciones de tipo Enabling o Choice
   (sourceTask ?sourct)
   (targetTask ?targt)
   (id ?id))
         =>
(assert (RelacionBinaria ;Se crea fact con nombre RelacionBinaria
 (slot-get ?sourct name);Nombre tarea fuente
 (slot-get ?sourct unaryRelationship);Relacion unaria de tarea fuente
 (slot-get ?targt name) ;Nombre tarea objetivo
(slot-get ?targt unaryRelationship);Relación unaria de tarea objetivo
  ?id ))) ;Id de relación binaria
```

Se obtiene el nombre de la clase a la cual pertenece la relación unaria de las tareas del fact anterior.

```
(defrule NombreRelacionUnaria
 (RelacionBinaria ;Nombre del fact creado en la regla anterior
   ?nombrest
   ?runariast
   ?nombretargt
   ?runariatargt
```

```
   ?id)
      =>
 (assert (RelacionesUnarias ;Nombre del fact a crear
 ?nombrest
 (class ?runariast);Clase de la relación unaria de la tarea fuente
  ?nombretargt
(class ?runariatargt);Clase de la relación unaria de la tarea objetivo
  ?id)))
```

En esta regla se comparan los nombres de las clases obtenidas en el paso anterior, si la relación unaria de la tarea fuente es de la clase FiniteIteration y la tarea objetivo de la clase Optional se debe generar un AbstractContainer.

```
(defrule crearContenedorAbstracto
  (RelacionesUnarias
   ?nombrest
   ?runariast&:(= ?runariast FiniteIteration);Pregunta
   ?nombretargt
   ?runariatargt&:(= ?runariatargt Optional);Pregunta
   ?id)
   =>
;Se crea instancia de clase AbstractContainer
 (make-instance of AbstractContainer
   (name ?nombretargt)
   (id ?id)))
(make-instance of AbstractContainer
   (name ?nombrest)
   (id ?id)))
(defrule  eliminarContenedorRepetido      (object           (is-a
AbstractContainer)   (OBJECT  ?objAbs1)     (name  ?nomA))  (object
(is-a  AbstractContainer)  (OBJECT  ?objAbs2&~?  objAbs1)      (name
?nomA))  =>  (unmake-instance ?objAbs2))
```

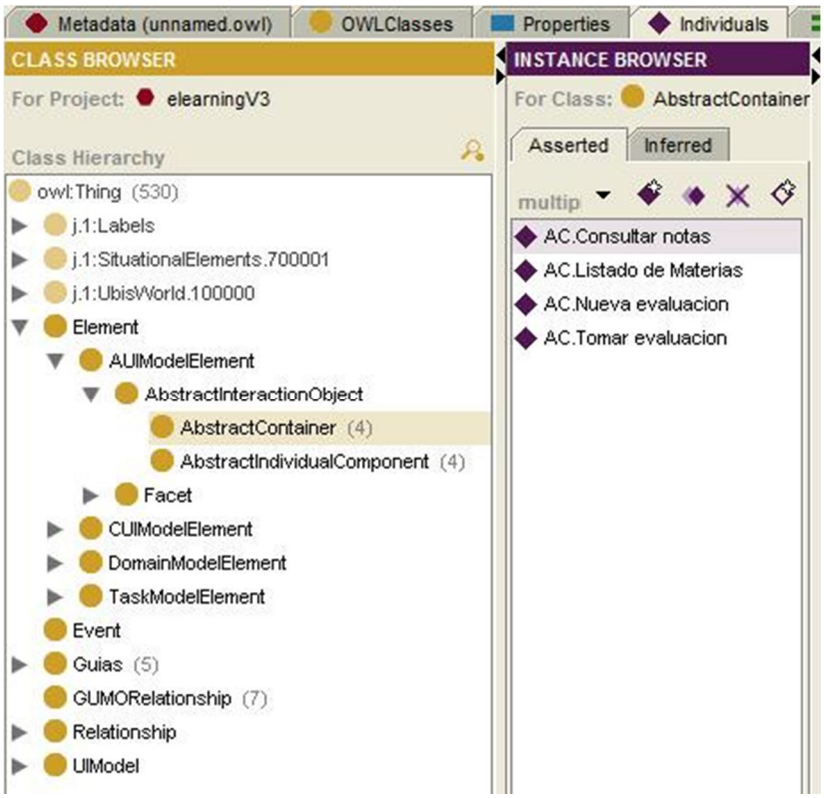


Imagen A.5. Instancias de la clase *AbstractContainer*.

Ahora se generan cada uno de los *AbstractIndividualComponent*. En este caso, para cada una de las instancias de *AbstractContainer* y se crea una instancia de *AbstractIndividualComponent* con la propiedad *name* y *parentContainer* como se observa en la imagen A.6.

```
(defrule crearComponenteIndividualAbstracto
  (object
    (is-a AbstractContainer)
    (OBJECT ?obj)
    (name ?nombreAC))
     =>
 (make-instance of  AbstractIndividualComponent
  (name ?nombreAC)
  (parentContainer ?obj)))
```

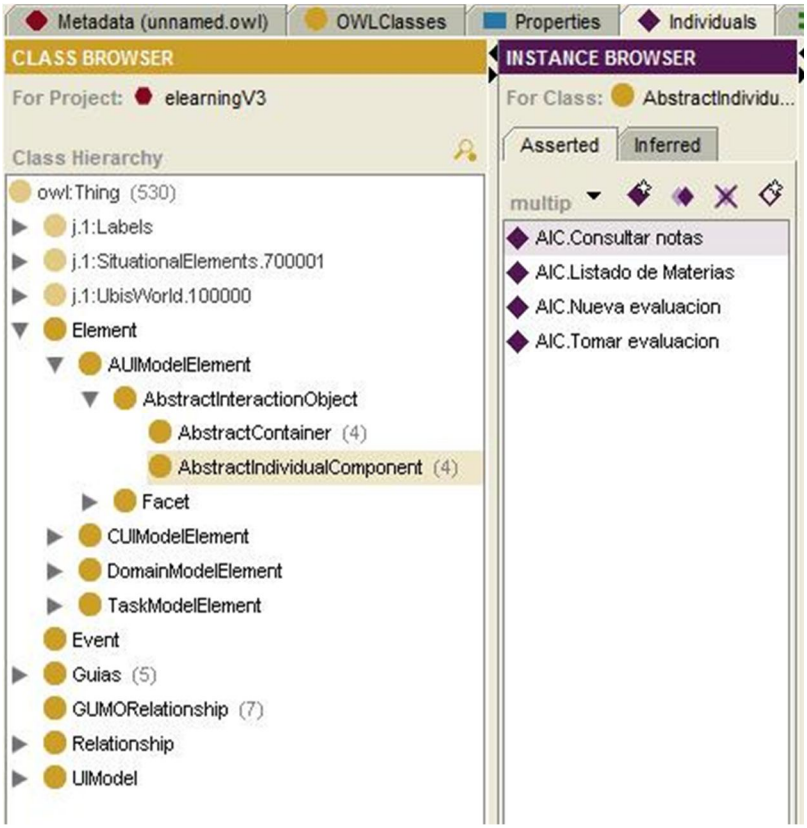


Imagen A.6. Instancias de la clase *AbstractIndividualComponent*.

A continuación se muestra el proceso de desarrollo utilizado para crear las instancias de cada una de los Facets del modelo AUI.

Output

Se crean todos los Output. Se recorre la clase *Manipulates* en busca de las relaciones entre las tareas y los elementos del dominio como se observa en imagen A.7.

Para crear los *Output* se aplica la siguiente regla: *Si una tarea (T1 perteneciente a TaskModel) está relacionada con una clase del modelo de dominio (C1) entonces crear un "Elemento de salida" (Output) para todos los atributos de dicha clase*.

R = relación

Si T1 R C1 ^ C1 = Clase => instanciar clase *Output*

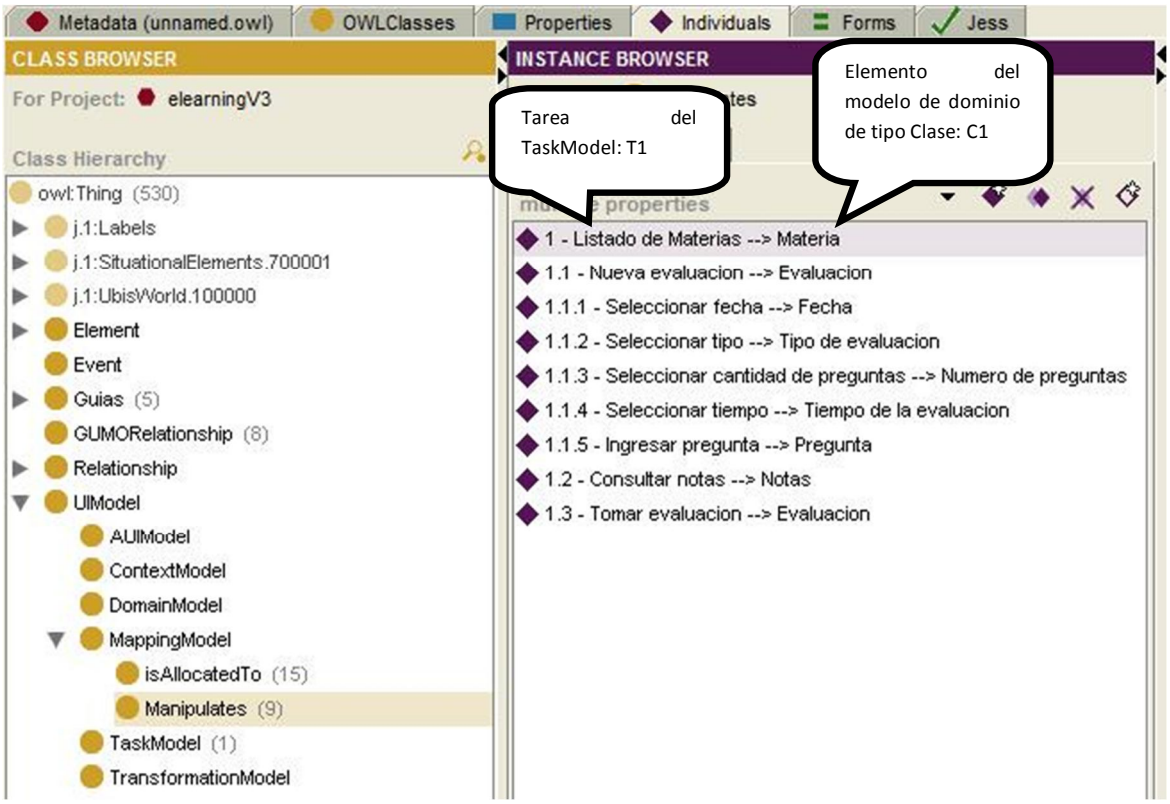


Imagen A.7. Instancias de la clase *Manipulates*.

Se obtienen de cada una las relaciones los atributos *sourceManipulate* que toma una instancia de la clase *Tareas* y *targetManipulates* que toma una instancia del modelo de dominio, específicamente a la clase *Clase* o *Attributes* que están incluidas en el mismo.

```
 (defrule AUIFacetTareaDominio
  (object
   (is-a Manipulates)
   (sourceManipulate ?smanip)
   (targetManipulate ?tmanip))
     =>
;Se guarda en el fact el nombre de la tarea y el elemento del modelo de
dominio
(assert (TareDominio
  (slot-get ?smanip name)
   ?tmanip)))
```

Ahora se determina a que clase del modelo de dominio pertenece el elemento *targetManipulate* que se encuentra en el fact TareaDominio y en la variable ?tmanip.

```
(defrule Atributos
 (TareDominio
   ?smanip                ;El nombre de la tarea
   ?tmanip&:(= (class ?tmanip) Clase));Si es de tipo "Clase"
    =>
;Obtenemos los atributos de la clase
(assert
  (TareAtrb
   ?smanip
   (slot-get ?tmanip attributes))))
```

Finalmente se crean los *Output* recorriendo el fact creado anteriormente. Para cada uno de los elementos del mismo se obtienen instancias de la clase *AbstractIndividualComponent* y la clase *Task* que estén relacionadas con la variable ?smanip.

```
(defrule CrearOutput
 (TareAtrb
   ?smanip         ;El nombre de la tarea
   $?attrb)       ;El simbolo $ hace referencia a un array
;Se obtiene la instancia de la clase AbstractIndividualComponent y Task
que posea el mismo nombre en ?smanip, obtenido del fact TareAtrb
(object
   (is-a AbstractIndividualComponent)
   (OBJECT ?obj)
   (name ?smanip))
(object
    (is-a Task)
    (name ?smanip)
    (Taskitem ?titem)
    (Tasktype ?ttype))
        =>     ;Para cada uno de los atributos en $?tmanip se crea una
              ;instancia de Output
(foreach ?variable $?attrb
 (make-instance of Output
  (name (slot-get ?variable  name))
  (abstractComponent ?obj)
  (actionItem ?titem)     ;?titem obtenida de recorrer la clase Task
  (actionType ?ttype))))  ;?ttype obtenida de recorrer la clase Task
  (outputContent ?variable))));?variable: Contenido a mostrar
```

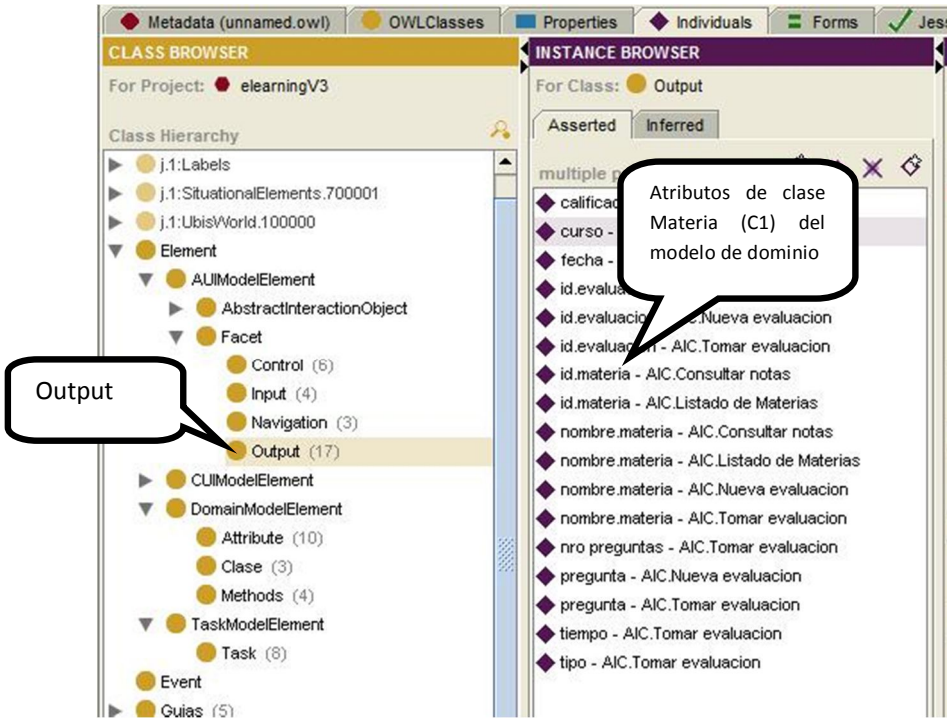


Imagen A.8. Instancias de la clase Output.

Input

Si una tarea (T1 perteneciente a TaskModel) está relacionada con un aributo del modelo de dominio (A1) entonces crear un "Elemento de entrada" (Input) para dicho atributos.

R = relación

Si T1 R A1 ^ A1 = Attribute => instanciar clase Input

Similarmente a la creación de Output, para crear los Input se recorre en primer lugar la clase *Manipulates*.

```
(defrule AUIFacetTareaDominio
  (object
   (is-a Manipulates)
   (sourceManipulate ?smanip)
   (targetManipulate ?tmanip))
     =>
;Se guarda en el fact el nombre de la tarea y el elemento del modelo de dominio
(assert (TareDominio
  (slot-get ?smanip name)
   ?tmanip)))
```

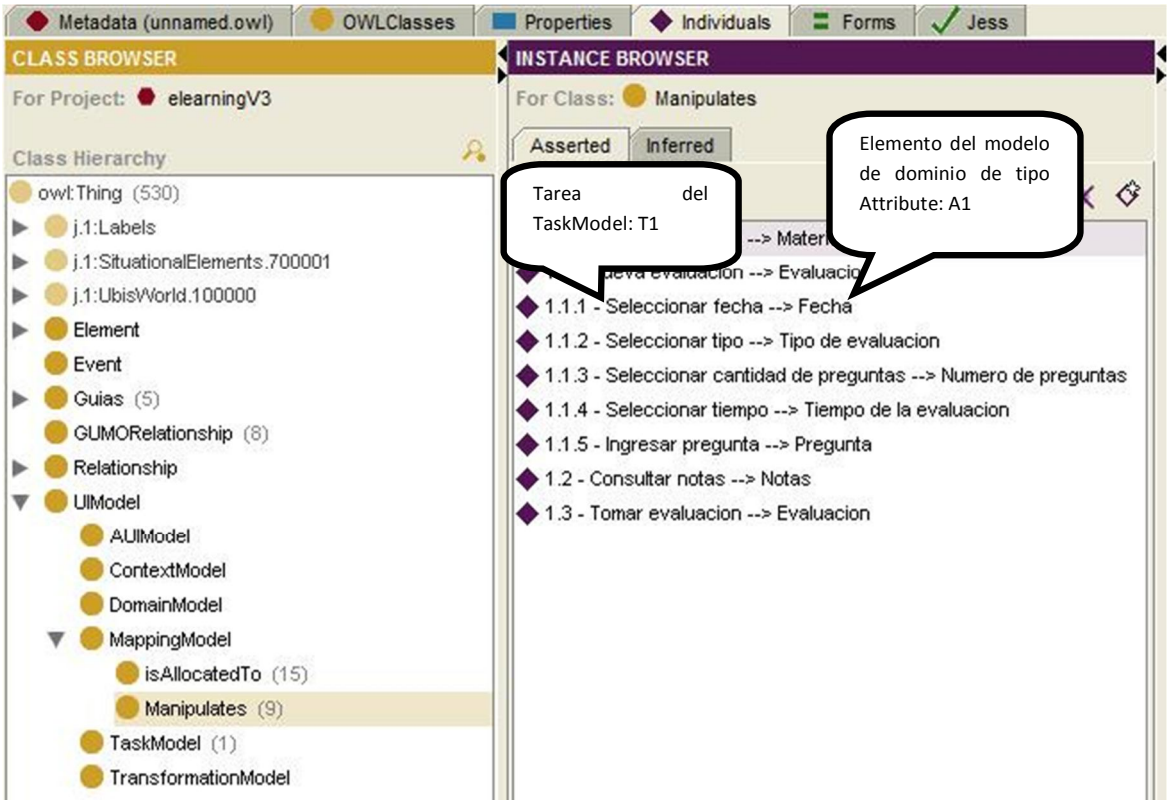


Imagen A.9. Instancias de la clase *Manipulates*.

Se guarda en un fact solamente las tareas que tengan relación *Manipulate* con un atributo.

```
(defrule Atributos
 (TareDominio
  ?smanip
  ?tmanip&:(= (class ?tmanip) Attribute)) ;Si es de tipo "Clase"
   =>
(assert (Atrb
  ?smanip
  (slot-get ?tmanip name)))) ;El tipo de dato del atributo
```

Del fact anterior se toman las tareas que posean las propiedades *Tasktype* "*select*" y *Taskitem* "*element*". Esto nos indica que se trata de una tarea que solicita el ingreso de un elemento del modelo de dominio. Se obtiene también la segunda tarea con la que posee la relación *decomposition*, es decir, hay una relación de inclusión. De esta relación se podrá saber dentro de cual *AUIIndividualElement* se ubica la tarea.

```
(defrule AttrTareaTipo
 (Atrb
  ?smanip      ;Nombre de la tarea
  ?nomAt)       ;Nombre del Atributo
  (object
   (is-a Task)
   (name ?smanip)
   (decomposition ?decom)
   (Tasktype ?sel&:(= ?sel select|create));de tipo select o create
   (Taskitem ?tit&:(= ?tit  element))) ;si es de tipo element
    =>
(assert
  (AttrTarea
   ?nomAt                ;nombre de tarea
   (slot-get ?decom targetTask))))     ;si pertenece a otra tarea
```

Se necesita ejecutar una regla más para obtener el nombre de la tarea mencionada anteriormente.

```
(defrule Aux1
 (AttrTarea
  ?nomAt
  ?tdecomp)
   =>
(assert
  (AttrNomtarea
   ?nomAt
   (slot-get ?tdecomp name))))
```

Se crean finalmente los Input.

```
(defrule CrearInput
  (AttrNomtarea
   ?nomAt
   ?nomTar)
  (object
   (is-a AbstractIndividualComponent)
   (OBJECT ?obj)
   (name ?nomTar))
   (object
```

```
    (is-a Output)
     (abstractComponent ?obj)
     (name ?nomAt))
     =>
   (make-instance of Input
    (name ?nomAt)
    (abstractComponent ?obj)
    (dataType ?datatype)
    (actionType ?taskitem)))
```

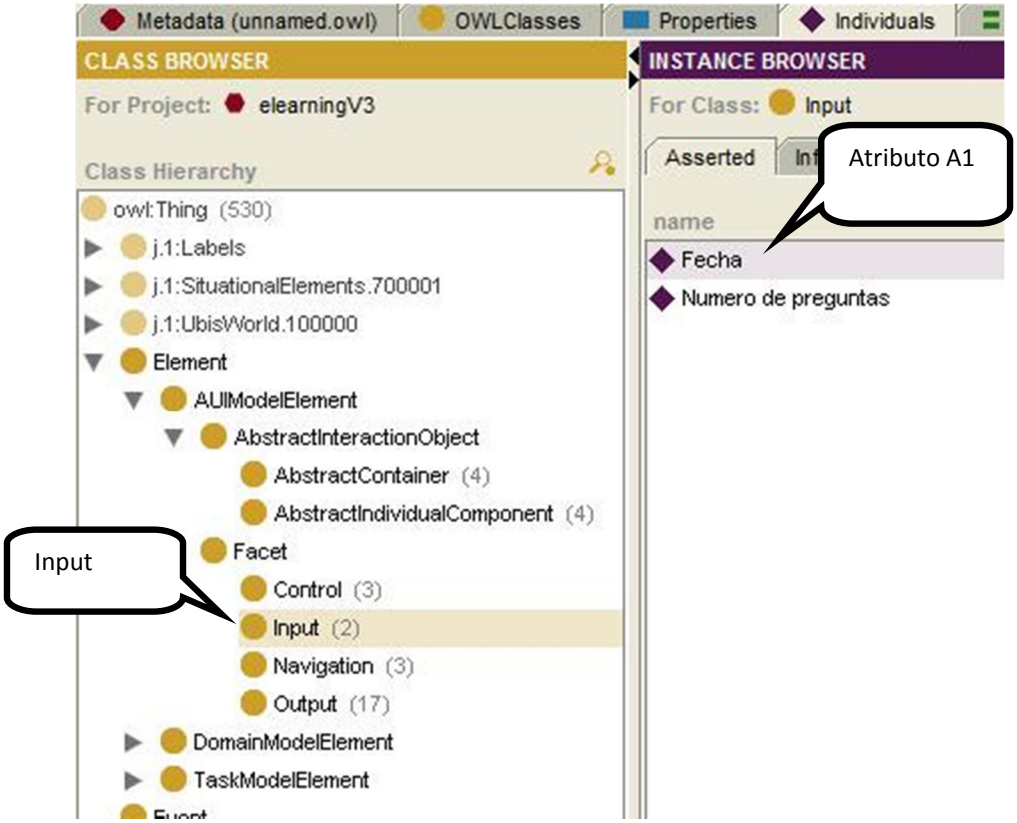


Imagen A.10. Instancias de la clase Input.

En el momento de crear los *Output* se lo hizo para todos los atributos. Algunos de ellos en realidad deberían ser *Input*, por eso ahora se los elimina. Esto nos lo indica el modelo de tareas, ya que hay tareas que se relacionan con los atributos de manera diferente, en nuestro caso, se toman las tareas con *Tasktype=select* y *Taskitem=item*.

```
(defrule EliminarOutput
 (AttrNomtarea
  ?nomAt
  ?nomTar)
  (object
```

```
    (is-a AbstractIndividualComponent)
    (OBJECT ?obj)
    (name  ?nomTar))
    (object (is-a Output)
     (name ?nomAt)
     (abstractComponent ?obj)
     (OBJECT ?o))
    =>
(unmake-instance ?o)) ;Se elimina la instancia
```

Navigation

Si dos tareas (T1 y T2 pertenecientes a TaskModel) están relacionadas mediante un tipo de relación binaria de habilitación (Enabling) entonces crear un "Elemento de Navegación" (Navigation).

R = relación

Si T1 R T2 ^ (R = Enabling) => instanciar clase Navigation

Si existe una relación binaria de tipo Enabling, se crea una instancia de tipo Navigation. Esta relación indica que hay tareas que posibilitan la ejecución de otras tareas.

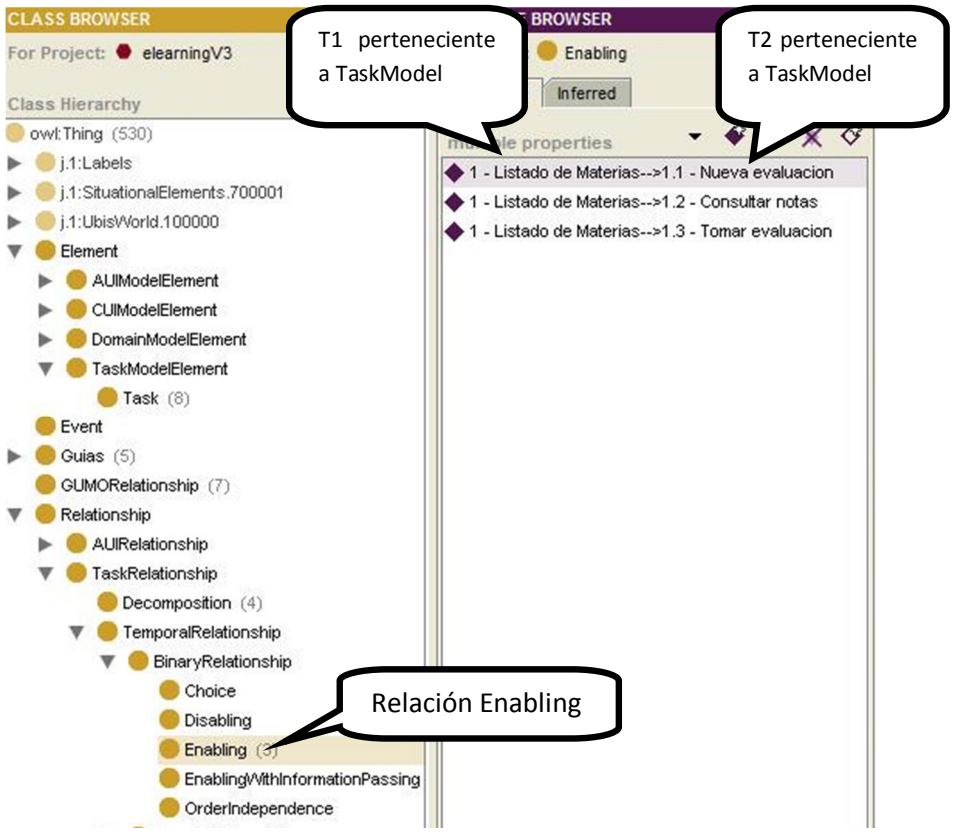


Imagen A.11. Instancias de la clase *Enabling*.

En primer lugar se crean dos reglas, la primera para obtener las tareas de la relación *Enabling* y la segunda para obtener los nombres de dichas tareas.

```
 (defrule TareasEnabling
  (object
  (is-a Enabling)
  (OBJECT ?obj))
   =>
  (assert
   (SourTTargeT
   (slot-get ?obj sourceTask)
   (slot-get ?obj  targetTask))))
(defrule NombreTareasEnabling
  (SourTTargeT
   ?sourcet
   ?targett)
   =>
 (assert
  (SourTNTargeTN
   (slot-get ?sourcet name)     ;Nombre de tarea fuente
   (slot-get ?targett name))))  ;Nombre de tarea objetivo
```

Con la siguiente regla se crean las instancias de la clase *Navigation*.

```
(defrule CrearNavigation
  (SourTNTargeTN
   ?nombresourcet
   ?nombretargett)
   (object
    (is-a AbstractIndividualComponent)
    (OBJECT ?obj)
    (name  ?nombretargett))
   =>
  (make-instance of Navigation
   (name ?nombresourcet)
   (abstractComponent ?obj)))

(defrule eliminarNavigationRepetido
  (object
```

```
    (is-a Navigation)
    (OBJECT ?objAbs1)
   (name ?nombre)
   (abstractComponent ?abscomp))
(object
  (is-a Navigation)
  (OBJECT ?objAbs2&~?objAbs1)
    (name ?nombre)
    (abstractComponent ?abscomp))
  =>
   (unmake-instance ?objAbs2))
```

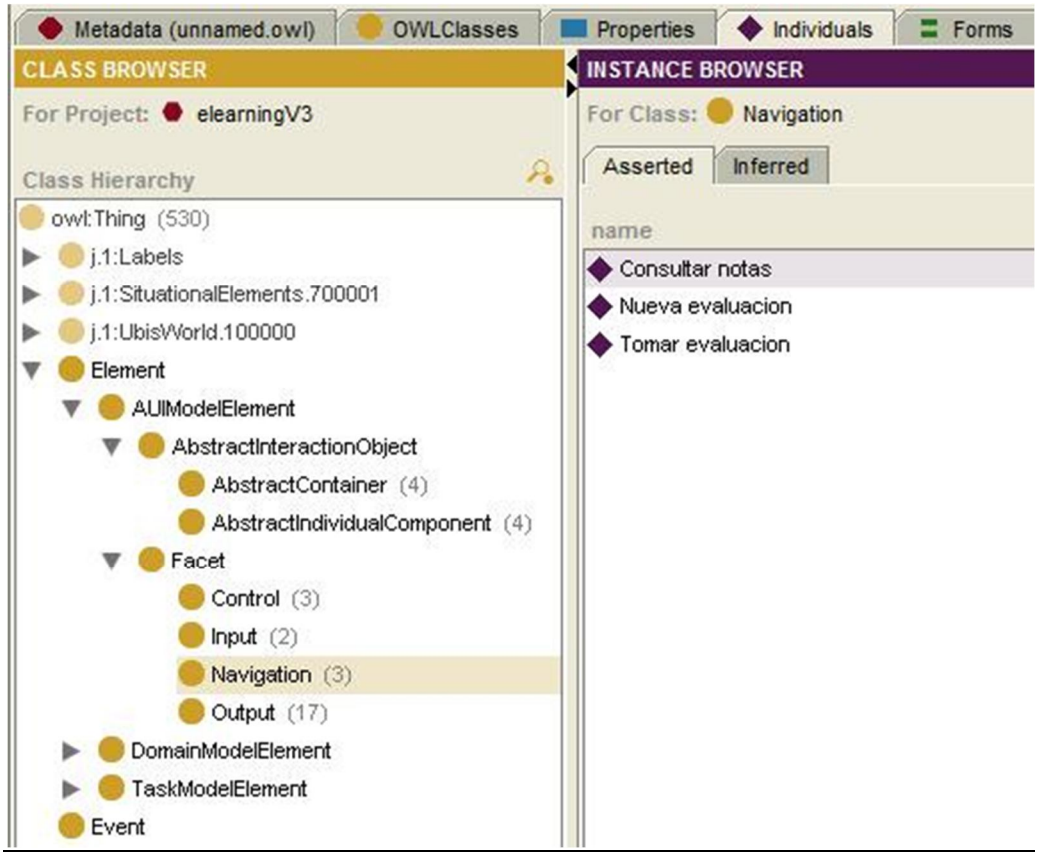


Imagen A.12. Instancias de la clase *Navigation.*

.Control

Si una tarea (T1 perteneciente a TaskModel) está relacionada con una clase del modelo de dominio (C1) y dicha clase posee algún método asociado (M1) entonces crear un "Elemento de control" (Control) para dichos métodos.

R = relación

∋ = contiene

Si T1 R C1 ^ C1 = Clase ^ C1 ∋ M1 => instanciar clase Control

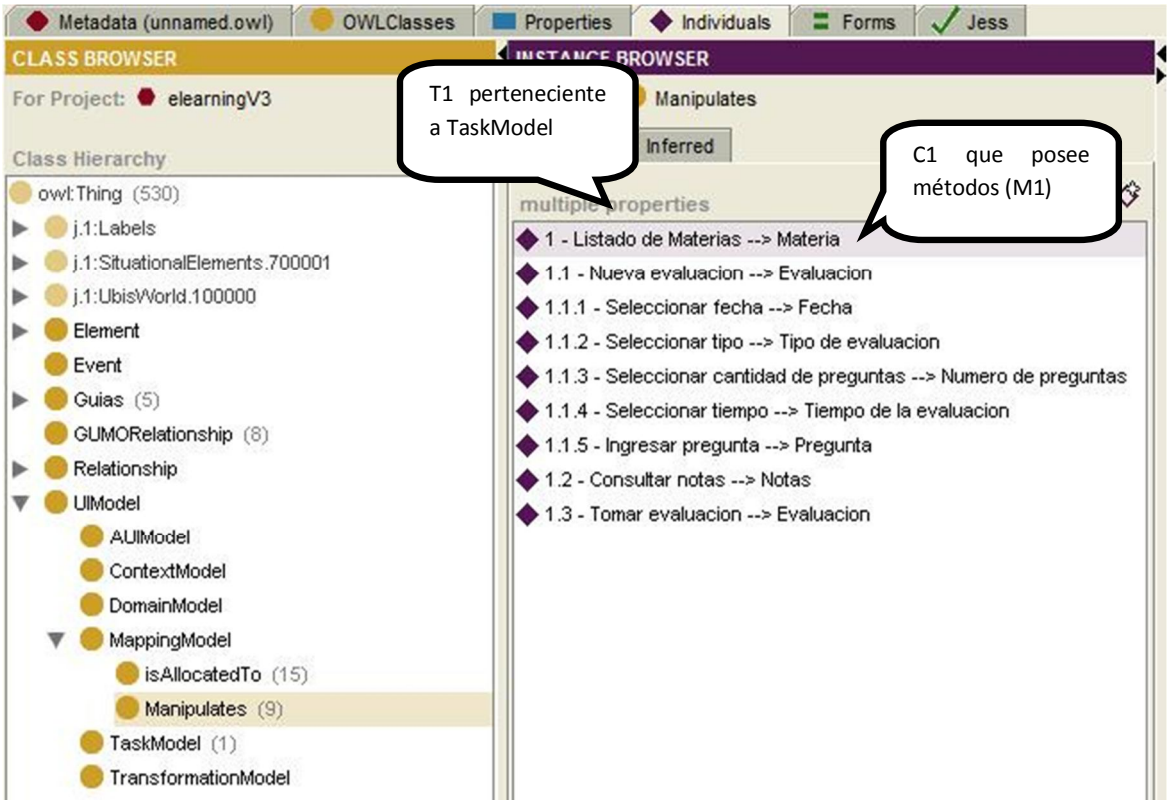


Imagen A.13. Instancias de la clase *Manipulates*.

Para crear los elementos de tipo Control se recorren *Manipulates* y se obtienen los *Methods* del modelo de dominio. Luego con esta información se pueden crear los elementos abstractos de tipo *Control*.

```
(defrule AUIFacetTareaDominio
(object
 (is-a Manipulates)
 (sourceManipulate ?smanip)
 (targetManipulate ?tmanip))
  =>
(assert
```

```
  (TareDominio
   (slot-get ?smanip name)
    ?tmanip)))
```

De la tarea objetivo, la cual debe ser de la clase *Clase* se obtiene el atributo *methods*.

```
 (defrule Metodos
 (TareDominio
  ?smanip
  ?tmanip &:(= (class ?tmanip) Clase))
   =>
 (assert (Met
   ?smanip
   (slot-get ?tmanip methods))))
```

Con al fact anterior se pueden crear las instancias de la clase *Control*.

```
 (defrule CrearControl
  (Met
   ?smanip
   $?methods)
 (object
  (is-a AbstractIndividualComponent)
  (OBJECT ?obj)
  (name ?smanip))
   =>
(foreach  ?variable $?methods
 (make-instance of Control
 (name (slot-get ?variable name)) ;Nombre del metodo
 (abstractComponent ?obj) )));El AbstractIndividualComponent al cual
pertenece
```

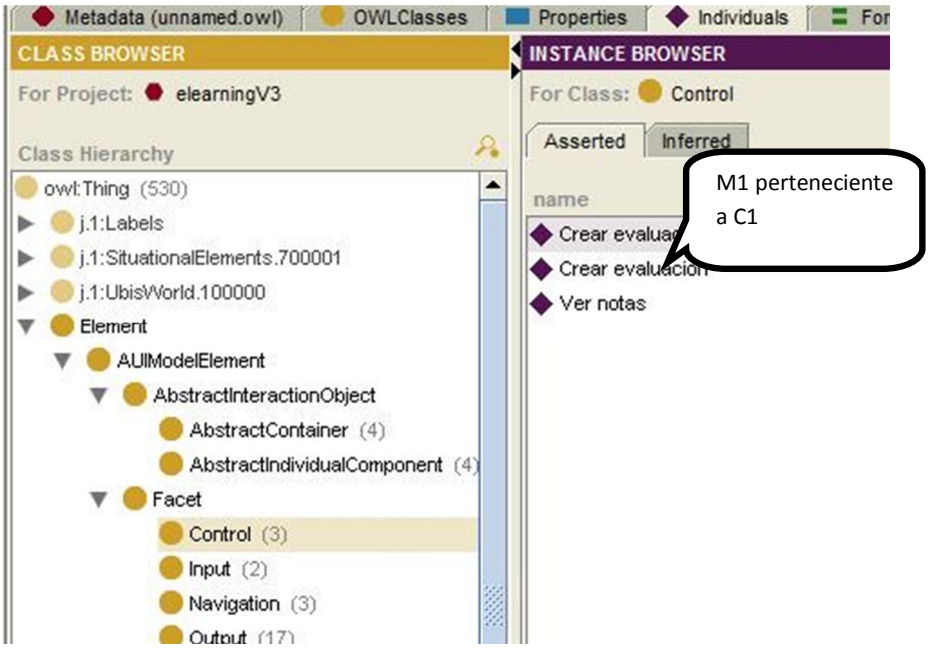


Imagen A.14. Instancias de la clase *Control*.

REGLAS PARA EL MODELO CONCRETO

Se obtienen los elementos concretos que indican las guías de usabilidad para cada uno de los usuarios. Es decir, los elementos concretos que se deben reemplazar, por cuales reemplazar y de acuerdo a cuales características de los usuarios.

Primero los elementos de guías que se aplican a ciertos usuarios, por último las guías que se aplican a todos los usuarios por igual.

```
(defrule UsuariosGuias
  (object
    (is-a j.1:Person.110003)
    (OBJECT ?u)
    (rdfs:label ?nomusuario)) ;Nombre de la persona
  (object
    (is-a GUMORelationship) ;Relacion de los usuarios y sus
    (subject ?u)            ;características
    (auxiliary ?auxi)
  (object ?objlabel))
    (object (is-a Guias)
    (auxiliary ?auxi)
    (object ?objlabel)
    (tiene_elementos $?elementos))
  =>
```

```
(assert
  (ElementosGuias
    ?nomusuario
    $?elementos) ))
```

Elementos de *guias* que se aplican a todos los usuarios por igual

```
(defrule UsuariosGuiasGenerales
  (object
    (is-a j.1:Person.110003)
    (OBJECT ?u)
    (rdfs:label ?nomusuario)) ;Nombre de la persona
  (object (is-a Guias)
    (auxiliary nil)
    (object nil)
    (tiene_elementos $?elementos))
  =>
(assert
  (ElementosGuias
    ?nomusuario
    $?elementos) ))
```

Ahora es necesario recorrer más profundamente las clases de la ontología para obtener los detalles de los elementos concretos, sus nombres, propiedades, etc.

```
(defrule UsuariosElementosGuias_1
  (ElementosGuias ?nomusuario $?elementos)
   =>
  (foreach ?variable $?elementos
   (assert
    (DetallesElementosGuias_Aux1 ?nomusuario
    (slot-get ?variable nombre)
    Propiedades
    (slot-get ?variable propiedades)
     ReempPor
    (slot-get ?variable reemp_a))) ))
```

```
(defrule UsuariosElementosGuias_2
  (DetallesElementosGuias_Aux1
    ?nomusuario
    ?el
    Propiedades $?propiedades
    ReempPor $?reempPor)
  =>
  (foreach ?variable $?reempPor
    (assert
     (DetallesElementosGuias_Aux2
       ?nomusuario
       ?el
       Propiedades $?propiedades
       ReempPor
       (slot-get ?variable nombre)
       Propiedades
      (slot-get ?variable propiedades) )) ))
```

```
(defrule UsuariosElementosGuias_3
  (DetallesElementosGuias_Aux2
   ?nomusuario
   ?el
   Propiedades $?propiedades
   ReempPor ?reempPor
   Propiedades $?propiedades_reemp )
 =>
(foreach ?variable $?propiedades_reemp
  (assert
   (DetallesElementosGuias
    ?nomusuario
    ?el
    Propiedades $?propiedades
    ReempPor ?reempPor
    Propiedades (slot-get ?variable nombre) ))))
```

Finalmente en el fact DetallesElementosGuias se poseen todos los elementos y sus detalles para cada usuario.

```
(defrule UsuariosElementosGuias_4
  (DetallesElementosGuias_Aux2
   ?nomusuario
   ?el
   Propiedades $?propiedades
   ReempPor ?reempPor
   Propiedades)
=>
 (assert
  (DetallesElementosGuias
   ?nomusuario
   ?el
   Propiedades $?propiedades
   ReempPor ?reempPor Propiedades nil )))
```

La regla siguiente se ocupa de procesar la clase *isAllocatedTo* que sirve para establecer una relación entre el modelo de tareas y el modelo de usuario, en particular entre la clase *Task* y la clase *Person* de GUMO.

Imagen A.15. Instancias de la clase *isAllocatedTo*.

Se crea el formato del fact que guardará todos los elementos para cada uno de los usuarios.

```
(deftemplate UsuariosTareasComponent
  (slot NU) ;nombre de usuario
  (slot NT) ;nombre de tarea
  (slot ABSC) ;contenedor al cual pertenece
```

```
 (defrule UsuariosTareas
  (object
    (is-a j.1:Person.110003)
    (OBJECT ?usuario)
    (rdfs:label ?nombreUsu));Nombre de la persona
(object
  (is-a isAllocatedTo)
  (target ?usuario) ;Variable ?usuario, extraída de la clase anterior
  (source ?tarea))
   =>
(assert
  (UsuariosTareasComponent
   (NU ?nombreUsu)
   (NT (slot-get ?tarea name))
   (ABSC 0))) ;Nombre de la tarea
```

Cada uno de los *AbstracContainer* posee uno o más *AbstractIndividualComponent* y cada uno de estos posee los *Input*, *Control*, *Output* y *Navigation*. Ahora se recorre la clase *AbstractIndividualComponent* para guardarlo en un fact y luego poder obtener estos elementos abstractos.

```
(defrule ComponentesAbstractos
  ?UsTC<-(UsuariosTareasComponent
          (NU ?nomusuario)
          (ABSC)
          (NT ?nomtarea))
 (object
   (is-a AbstractIndividualComponent)
   (OBJECT ?obj)
   (name ?nomtarea))
     =>
(modify ?UsTC (ABSC ?obj) ))
```

Elemento *Window*

Se crean las reglas para generar el elemento concreto de tipo *window*. Cada ventana se la asocia con el elemento abstracto *AbstractContainer* y con un usuario. Se recorre el fact creado anteriormente y para cada tarea y usuario, se guarda en un fact llamado Elementos el nombre del usuario, el elemento window y el nombre de la tarea.

Si una tarea (T1 perteneciente a TaskModel) está relacionada con una clase del modelo de dominio (C1) y dicha clase posee algún método asociado (M1) entonces crear un "Elemento de control" (Control) para dichos métodos.

R = relación

∋ = contiene

Si T1 R C1 ^ C1 = Clase ^ C1 ∋ M1 => instanciar clase Control

```
(defrule Ventanas_x_guias
 (UsuariosTareasComponent
   (NU ?nomusuario&"Manuel")
   (NT ?nomtarea)
   (ABSC ?abscomp))
(object
 (is-a AbstractIndividualComponent)
 (OBJECT ?abscomp))
(DetallesElementosGuias
  ?nomusuario
  ?el
  Propiedades $?propiedades
  ReempPor ?reempPor&"window"
  Propiedades ?propiedades_reemp)
  =>
(printout t ?nomusuario " <" ?el)
(foreach ?var $?propiedades
  (printout t " "(slot-get ?var nombre)))
  (printout t "> " ?nomtarea " </" ?el "> -> Tarea:" ?nomtarea crlf) )
```

```
(defrule resto_Ventanas
 (UsuariosTareasComponent
  (NU ?nomusuario&"Manuel")
  (NT ?nomtarea)
  (ABSC ?abscomp))
 (object
```

```
   (is-a AbstractIndividualComponent)
   (OBJECT ?abscomp))
   (not
    (DetallesElementosGuias
     ?nomusuario&"Manuel"
     ?el
     Propiedades    $?propiedades
     ReempPor ?reempPor&"window"
     Propiedades ?propiedades_reemp))
 =>
(printout t ?nomusuario " <window name=" ?nomtarea ">" ?nomtarea
"</window> -> Tarea:" ?nomtarea crlf) )
```

Elemento *Button*

Para crear los elementos de este tipo se recorre la clase *Control* y para todas las instancias de esta se crean los *button*.

```
(defrule Button_x_guias
 (UsuariosTareasComponent
  (NU ?nomusuario&"Manuel")
  (NT ?nomtarea)
  (ABSC ?abscomp))
 (object
  (is-a Control)
   (abstractComponent ?abscomp)
   (name ?nomelement)
   (actionType ?at))
 (DetallesElementosGuias
   ?nomusuario
   ?el
   Propiedades $?propiedades
   ReempPor ?reempPor&"button"
   Propiedades ?propiedades_reemp)
 =>
 (printout t ?nomusuario " <" ?el " name=" ?nomelement)
  (foreach ?var $?propiedades
   (printout t " "(slot-get ?var nombre)))
   (printout t "> "?nomelement" </" ?el "> -> Tarea:" ?nomtarea crlf) )
```

```
(defrule resto_Button
 (UsuariosTareasComponent
  (NU ?nomusuario&"Manuel")
  (NT ?nomtarea)
  (ABSC ?abscomp))
 (object
  (is-a Control)
  (abstractComponent ?abscomp)
  (name ?nomelement)
  (actionType ?at))
 (not
  (DetallesElementosGuias
  ?nomusuario
  ?el
  Propiedades $?propiedades
  ReempPor ?reempPor&"button"
  Propiedades ?propiedades_reemp))
 =>
 (printout t ?nomusuario " <button name=" ?nomelement ">" ?nomelement
"</button> -> Tarea:" ?nomtarea crlf) )
```

Elemento de tipo datePicker

Se recorren las instancias de la clase Input y para las mismas con *dataType* de tipo *date*, se crean los *datePicker*.

```
(defrule Datepicker_x_guias
 (UsuariosTareasComponent
  (NU ?nomusuario&"Manuel")
  (NT ?nomtarea)
  (ABSC ?abscomp))
 (object
  (is-a Input)
  (abstractComponent ?abscomp)
  (name ?nomelement)
  (dataType ?at&"date"))
  (DetallesElementosGuias
   ?nomusuario
   ?el
   Propiedades $?propiedades
   ReempPor ?reempPor&"datepicker"
   Propiedades ?propiedades_reemp)
 =>
```

```
 (printout t ?nomusuario " <" ?el " name=" ?nomelement)
  (foreach ?var $?propiedades
    (printout t " "(slot-get ?var nombre)))
  (printout t "> "?nomelement" </" ?el "> -> Tarea:" ?nomtarea crlf) )
```

```
 (defrule resto_Datepicker
 (UsuariosTareasComponent
  (NU ?nomusuario&"Manuel")
  (NT ?nomtarea)
  (ABSC ?abscomp))
 (object
  (is-a Input)
   (abstractComponent ?abscomp)
   (name ?nomelement)
   (dataType ?at&"date"))
  (not
   (DetallesElementosGuias
    ?nomusuario
    ?el
    Propiedades $?propiedades
    ReempPor ?reempPor&"datepicker"
    Propiedades ?propiedades_reemp))
 =>
  (printout t ?nomusuario " <datepicker name=" ?nomelement ">"
?nomelement "</datepicker> -> Tarea:" ?nomtarea crlf) )
```

Elemento de tipo *listBox*

Para las instancias de la clase *Input* con la propiedad *actionItem* igual a *collection* y *dataType* de tipo string o integer se crean los *listBox*.

```
 (defrule Listbox_x_guias
  (UsuariosTareasComponent
   (NU ?nomusuario&"Manuel")
   (NT ?nomtarea)
   (ABSC ?abscomp))
 (object
  (is-a Input)
   (abstractComponent ?abscomp)
   (name ?nomelement)
   (dataType ?at&"integer" | "string" | "time")
```

```
   (actionType ?ai&"collection"))
  (DetallesElementosGuias
   ?nomusuario
   ?el
   Propiedades $?propiedades
   ReempPor ?reempPor&"listbox"
   Propiedades ?propiedades_reemp)
 =>
(printout t ?nomusuario " <" ?el " name=" ?nomelement)
 (foreach ?var $?propiedades
  (printout t " "(slot-get ?var nombre)))
 (printout t "> "?nomelement" </" ?el "> -> Tarea:" ?nomtarea crlf) )
```

```
(defrule resto_Listbox
 (UsuariosTareasComponent
  (NU ?nomusuario&"Manuel")
  (NT ?nomtarea)
  (ABSC ?abscomp))
 (object
  (is-a Input)
   (abstractComponent ?abscomp)
   (name ?nomelement)
   (dataType ?at&"integer" | "string")
   (actionType ?ai&"collection"))
 (not
  (DetallesElementosGuias
  ?nomusuario
  ?el
  Propiedades $?propiedades
  ReempPor ?reempPor&"listbox"
  Propiedades ?propiedades_reemp))
 =>
(printout t ?nomusuario " <listbox name=" ?nomelement ">" ?nomelement
"</listbox> -> Tarea:" ?nomtarea crlf) )
```

Elemento de tipo *inputText*

Similarmente al elemento anterior pero para *actionItem* igual a *element*, se crean los *inputText*.

```
(defrule Inputtext_x_guias
 (UsuariosTareasComponent
  (NU ?nomusuario&"Manuel")
  (NT ?nomtarea)
  (ABSC ?abscomp))
 (object
  (is-a Input)
  (abstractComponent ?abscomp)
  (name ?nomelement)
  (dataType ?at&"integer" | "string")
  (actionType ?ai&"element"))
  (DetallesElementosGuias
   ?nomusuario
   ?el
   Propiedades $?propiedades
   ReempPor ?reempPor&"inputtext"
   Propiedades ?propiedades_reemp)
 =>
 (printout t ?nomusuario " <" ?el " name=" ?nomelement)
  (foreach ?var $?propiedades
   (printout t " "(slot-get ?var nombre)))
  (printout t "> " ?nomelement " </" ?el "> -> Tarea:" ?nomtarea crlf))
```

```
(defrule resto_Inputtext
 (UsuariosTareasComponent
  (NU ?nomusuario&"Manuel")
  (NT ?nomtarea)
  (ABSC ?abscomp))
 (object
  (is-a Input)
  (abstractComponent ?abscomp)
  (name ?nomelement)
  (dataType ?at&"integer" | "string")
  (actionType ?ai&"element"))
(not
 (DetallesElementosGuias
```

```
   ?nomusuario
   ?el
   Propiedades $?propiedades
   ReempPor ?reempPor&"inputtext"
   Propiedades ?propiedades_reemp))
  =>
 (printout t ?nomusuario " <inputtext name=" ?nomelement ">" ?nomelement
"</inputtext> -> Tarea:" ?nomtarea crlf) )
```

Elemento de tipo link

En este caso se recorre la clase Navigation para crear los links

```
 (defrule link_x_guias
  (UsuariosTareasComponent
   (NU ?nomusuario&"Manuel")
   (NT ?nomtarea)
   (ABSC ?abscomp))
  (object
   (is-a Navigation)
   (abstractComponent ?abscomp)
   (name ?nomelement))
   (UsuariosTareasComponent
    (NU ?nomusuario&"Manuel")
    (NT ?nomelement)
    (ABSC ?abscomp2))
  (DetallesElementosGuias
   ?nomusuario
   ?el
   Propiedades $?propiedades
   ReempPor ?reempPor&"outputtext"
   Propiedades ?propiedades_reemp&"url")
  =>
  (printout t ?nomusuario " <" ?el " name=" ?nomelement)
  (foreach ?var $?propiedades
   (printout t " "(slot-get ?var nombre)))
  (printout t "> "?nomelement" </" ?el "> -> Tarea:" ?nomtarea crlf) )
```

```
(defrule resto_link
 (UsuariosTareasComponent
  (NU ?nomusuario&"Manuel")
  (NT ?nomtarea)
  (ABSC ?abscomp))
 (object
  (is-a Navigation)
  (abstractComponent ?abscomp)
  (name ?nomelement))
  (UsuariosTareasComponent
  (NU ?nomusuario&"Manuel")
  (NT ?nomelement)
  (ABSC ?abscomp2))
(not
 (DetallesElementosGuias
  ?nomusuario
  ?el
  Propiedades $?propiedades
  ReempPor ?reempPor&"outputtext"
  Propiedades ?propiedades_reemp&"url"))
 =>
(printout t ?nomusuario " <outputtext name=" ?nomelement "
url="?nomelement">" ?nomelement "</outputtext> -> Tarea:" ?nomtarea
crlf) )
```

Elemento de tipo *outputText* y *timeSensor*

Se recorre la clase *Output* y para todas las instancias excepto las que posean *dataType* igual a *time*, se crean los *outputText*. Para el resto se crea el elemento *timeSensor* como se observa al final de la regla.

```
(defrule Outputtext_x_guias
 (UsuariosTareasComponent
  (NU ?nomusuario&"Manuel")
  (NT ?nomtarea)
  (ABSC ?abscomp))
 (object
  (is-a Output)
  (abstractComponent ?abscomp)
  (name ?nomelement)
  (actionType ?at)
```

```
   (outputContent ?outcont))
   (DetallesElementosGuias
   ?nomusuario
   ?el
   Propiedades $?propiedades
   ReempPor ?reempPor&"outputtext"
   Propiedades ?propiedades_reemp)
 =>
(if (<> (slot-get ?outcont dataType) "time")then
   (printout t ?nomusuario " <" ?el " name=" ?nomelement)
   (foreach ?var $?propiedades
    (printout t " "(slot-get ?var nombre)))
   (printout t "> "?nomelement" </" ?el "> -> Tarea:" ?nomtarea crlf)) )
```

```
(defrule resto_Outputtext
 (UsuariosTareasComponent
   (NU ?nomusuario&"Manuel")
   (NT ?nomtarea)
   (ABSC ?abscomp))
 (object
   (is-a Output)
   (abstractComponent ?abscomp)
   (name ?nomelement)
   (actionType ?at)
   (outputContent ?outcont))
 (not
   (DetallesElementosGuias
    ?nomusuario
    ?el
    Propiedades $?propiedades
    ReempPor ?reempPor&"outputtext"
    Propiedades ?propiedades_reemp))
 =>
(if (<> (slot-get ?outcont dataType) "time")then
 (printout t ?nomusuario " <outputtext name=" ?nomelement ">"
?nomelement "</outputtext> -> Tarea:" ?nomtarea crlf)) )
```

```
(defrule Timesensor_x_guias
 (UsuariosTareasComponent
  (NU ?nomusuario&"Manuel")
  (NT ?nomtarea)
  (ABSC ?abscomp))
 (object
  (is-a Output)
  (abstractComponent ?abscomp)
  (name ?nomelement)
  (actionType ?at)
  (outputContent ?outcont))
  (DetallesElementosGuias
   ?nomusuario
   ?el
   Propiedades
   $?propiedades
   ReempPor ?reempPor&"timesensor"
   Propiedades ?propiedades_reemp)
=>
(if (= (slot-get ?outcont dataType) "time")then
 (printout t ?nomusuario " <" ?el " name=" ?nomelement)
 (foreach ?var $?propiedades
  (printout t " "(slot-get ?var nombre)))
 (printout t "> "?nomelement" </" ?el "> -> Tarea:" ?nomtarea crlf)) )
```

```
(defrule resto_Timesensor
 (UsuariosTareasComponent
  (NU ?nomusuario&"Manuel")
  (NT ?nomtarea)
  (ABSC ?abscomp))
 (object
  (is-a Output)
  (abstractComponent ?abscomp)
  (name ?nomelement)
  (actionType ?at)
  (outputContent ?outcont))
 (not
  (DetallesElementosGuias
   ?nomusuario
   ?el
```

```
    Propiedades $?propiedades
    ReempPor ?reempPor&"timesensor"
    Propiedades ?propiedades_reemp))
=>
(if (= (slot-get ?outcont dataType) "time")then
(printout t ?nomusuario " <timesensor name=" ?nomelement ">"
?nomelement "</timesensor> -> Tarea:" ?nomtarea crlf)) )
```

Todas estas reglas serán de utilidad para el diseñador luego de crear los modelos de tareas y de dominio, las relaciones entre los mismos y con GUMO.

Anexo B
Guía de Instalación

ANEXO B

GUÍA DE INSTALACIÓN

B.1. INTRODUCCIÓN

En este anexo se explicará cómo instalar los programas necesarios para utilizar el prototipo desarrollado en este trabajo. El proceso de instalación es simple porque se debe instalar solo un programa y se podrá acceder a todos los demás. A continuación se muestran los pasos que el usuario debe seguir para poder instalar hasta poner en marcha todo lo que viene en el DVD.

B.2. INSTALACIÓN DE VMWARE

En el DVD adjunto se encuentran un archivo ejecutable y una carpeta. El archivo ejecutable tiene el nombre de VMware-player.exe y es un instalador para la aplicación VMware Player. Se observa a continuación una imagen del archivo.

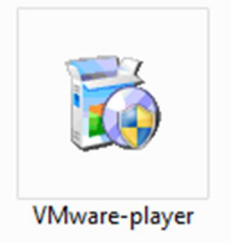


Imagen B.1. Archivo instalador de VMware.

VMware permite la ejecución de un sistema operativo como si fuera un programa más dentro de la computadora.

En primer lugar es necesario ejecutar este archivo para comenzar con la instalación de la aplicación.

Imagen B.2. Advertencia de seguridad.

En caso de mostrarse la ventana anterior, solo es necesario hacer clic en Ejecutar y aparecerá la ventana que se muestra en la imagen B.3, la cual es la ventana de bienvenida a la instalación.

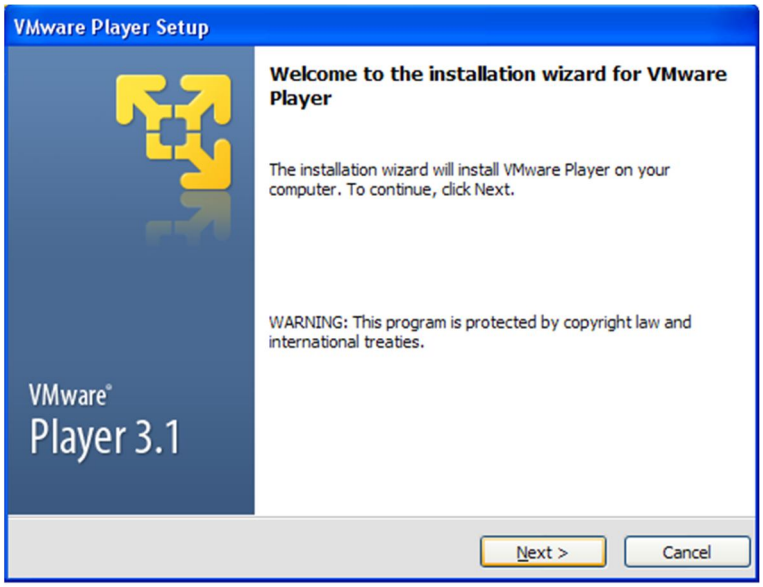


Imagen B.3. Pantalla de bienvenida del instalador de VMware player.

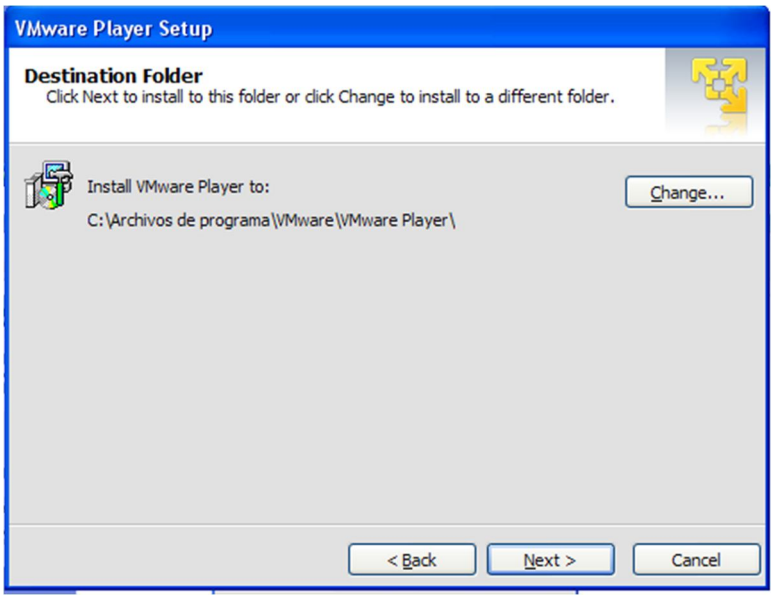


Imagen B.4. Instalación de VMware player, selección de la ubicación.

Al hacer clic en el botón **Next** de la ventana de bienvenida aparece una ventana en la que se puede modificar la carpeta destino de instalación por defecto del programa, imagen B.4.

Nuevamente se debe hacer clic en Next y aparecerá una ventana con un checkbox seleccionado por defecto que sirve para indicarle al programa que verifique actualizaciones cada vez que se ejecuta el mismo.

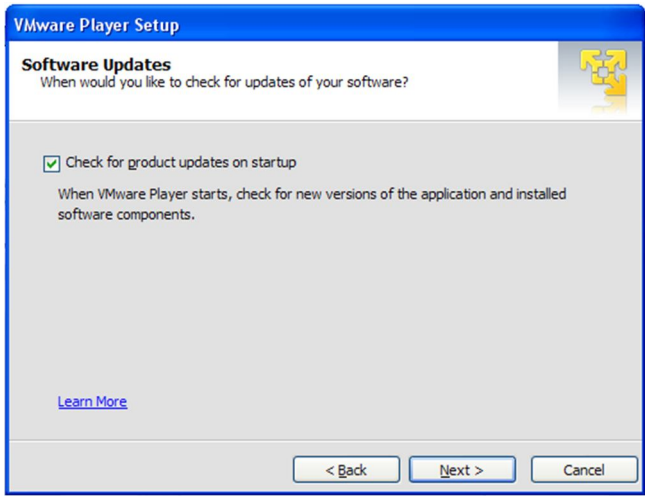


Imagen B.5. Instalación de VMware player, verificar actualizaciones.

La siguiente ventana similar a la anterior, pero esta vez con un checkbox para permitir a la aplicación enviar información anónima sobre del funcionamiento de la misma. Para el funcionamiento del prototipo esta característica no tiene ninguna influencia como tampoco la anterior.

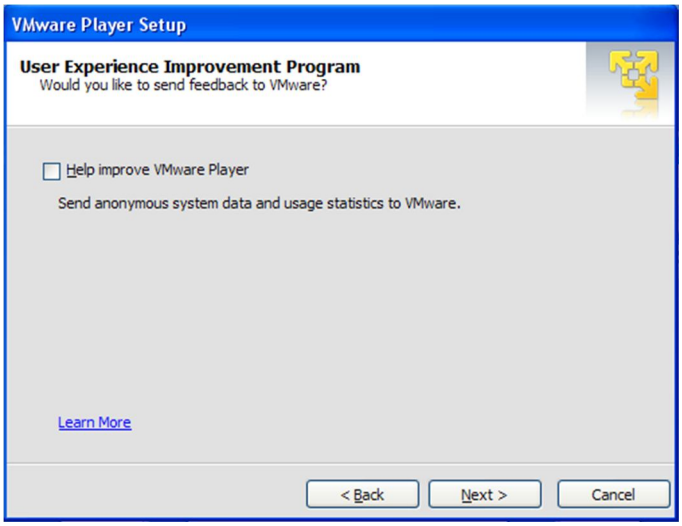


Imagen B.6. Instalación de VMware player, ayudar a mejorar el programa.

Finalmente se muestra la ventana que nos permite seleccionar los lugares en donde se crearán los accesos directos a la aplicación. Luego de esto comenzará la instalación.

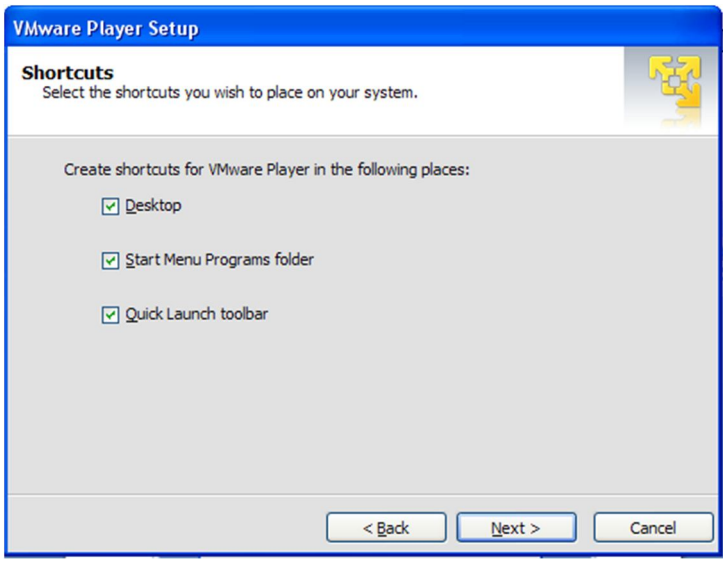


Imagen B.7. Instalación de VMware player, ubicación de los íconos.

Se observa a continuación en la imagen B.8 el progreso de la instalación que el usuario deberá aguardar su finalización para continuar con los siguientes pasos.

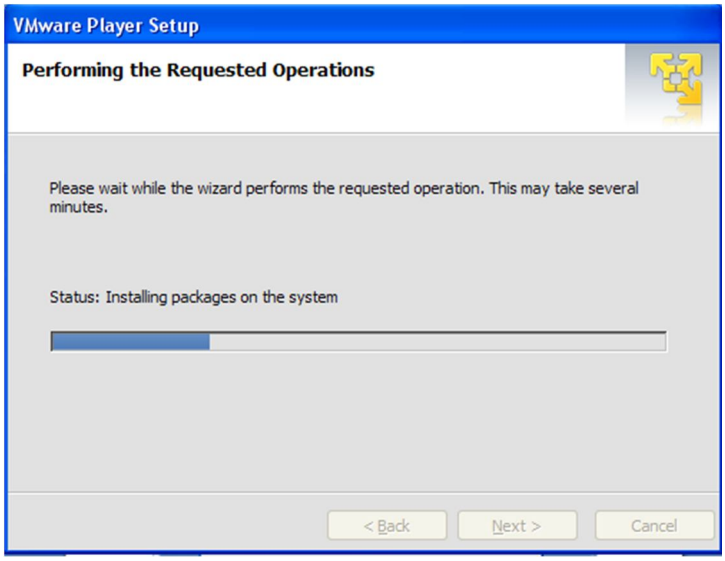


Imagen B.8. Instalación de VMware player, proceso automático de instalación.

Terminada la instalación, es necesario reiniciar la computadora, imagen B.9, para que la aplicación esté lista para usarse.

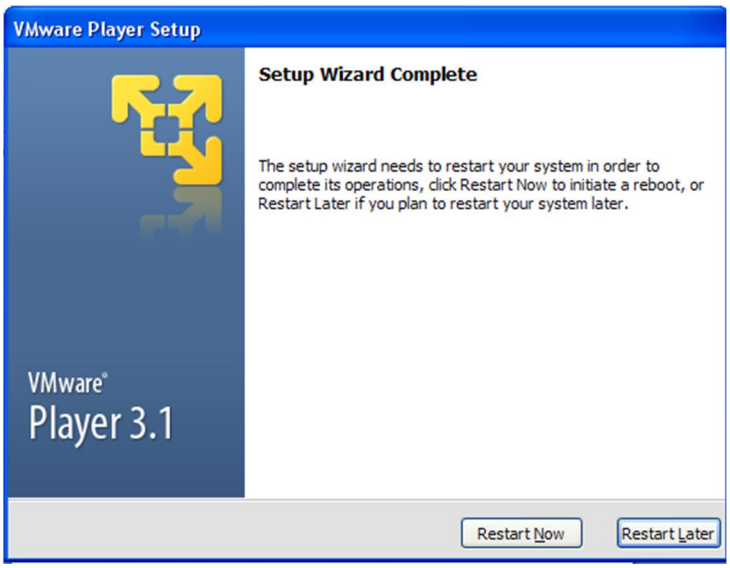


Imagen B.9. Instalación de VMware player, finalización del proceso.

Luego de reiniciar la computadora y al ejecutar la aplicación se muestra la siguiente pantalla en donde nos pide que aceptemos los términos de la licencia.

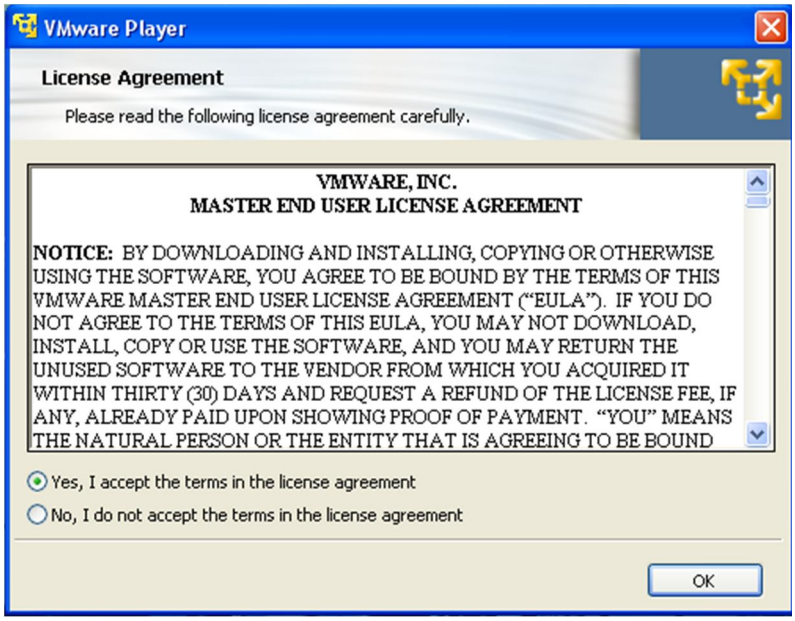


Imagen B.10. Acuerdo de licencia.

Una vez que se aceptan los términos de la licencia (seleccionando la opción *Yes, I accept the terms in the license agreement* y presionando el botón **ok**) aparece la pantalla principal de la aplicación.

Imagen B.11. Pantalla principal del programa.

B.3. EJECUCIÓN DEL SISTEMA OPERATIVO

Se observa en el panel de la derecha que posee las opciones para crear una nueva máquina virtual, para abrir una ya existente, para actualizar la aplicación a una versión diferente y por último la ayuda.

La opción a elegir es la de abrir una máquina virtual existente.

Imagen B.12. Opción para ejecutar el archivo preparado.

Como se mencionó al comienzo en el DVD se encuentra una carpeta. Ésta tiene el nombre XP y es en donde se encuentran los archivos necesarios para cargar el sistema operativo Windows XP luego de instalar la aplicación.

Nos ubicamos en dicha carpeta y abrimos el único archivo que nos permite ver la aplicación como se observa en la imagen.

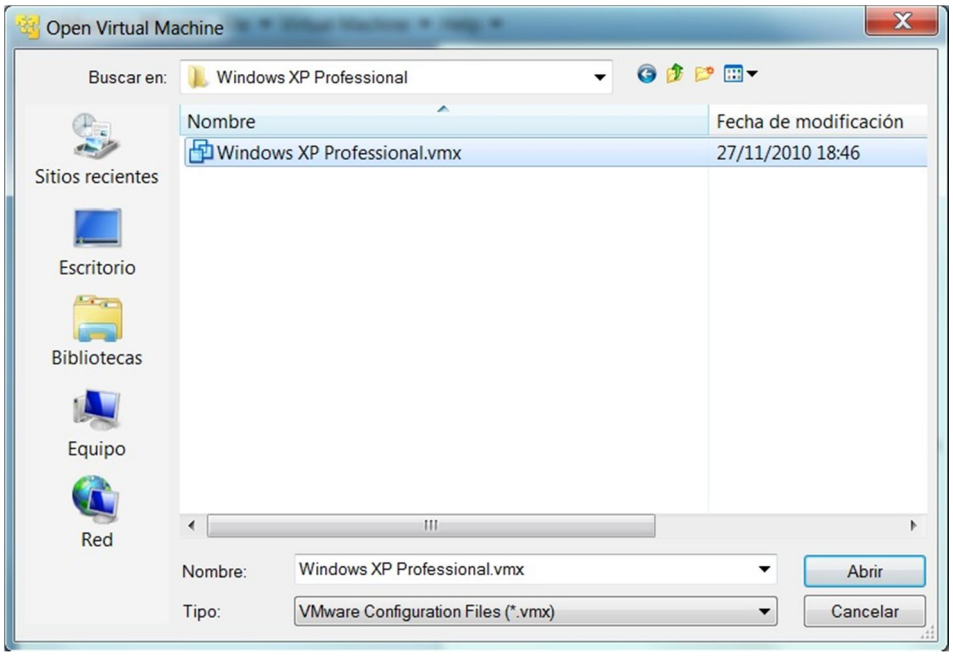


Imagen B.13. Selección del archivo preparado.

Al abrir este archivo se muestra lo siguiente.

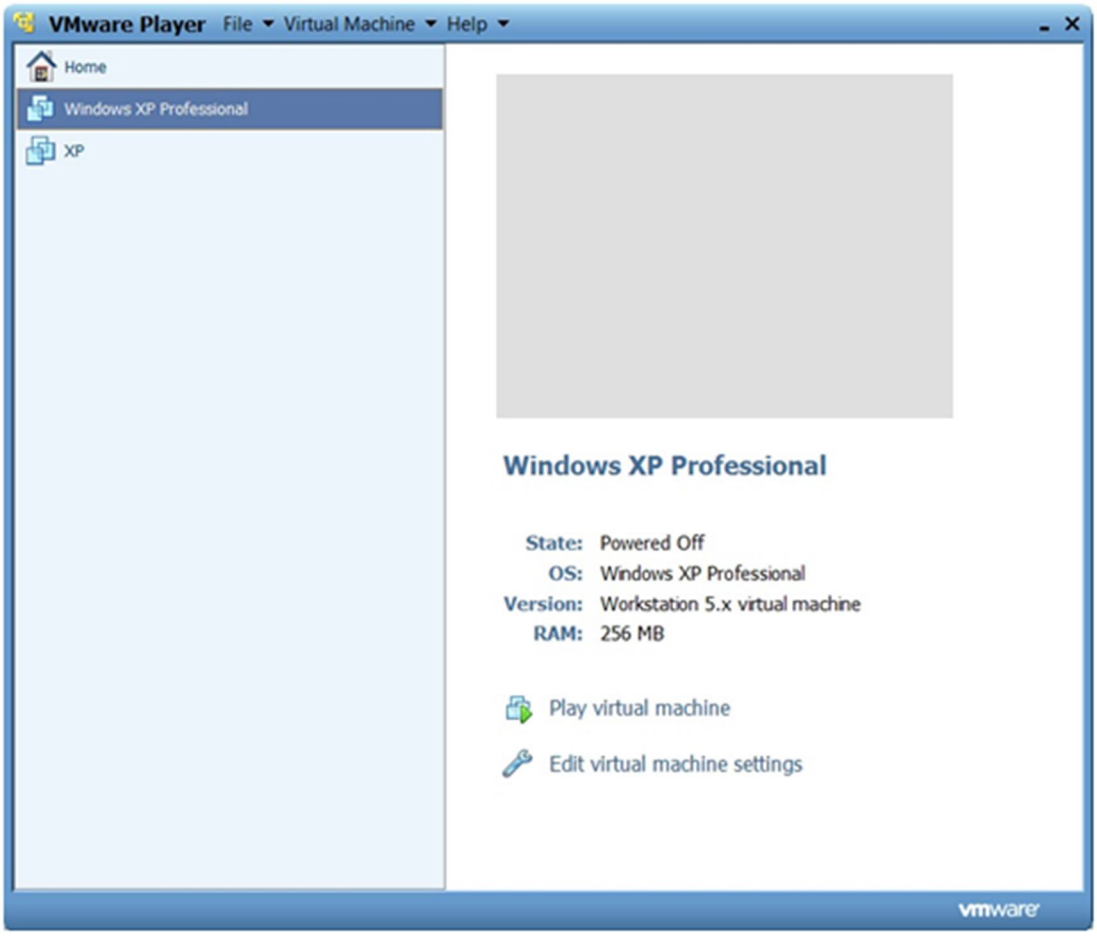


Imagen B.14. Ejecución de la Máquina Virtual.

Se observa en el panel de la derecha una descripción de la máquina virtual que indica entre otras cosas la memoria RAM asignada. Pero lo que interesa es la opción de la imagen siguiente, que permite encender el sistema operativo Windows XP.

Luego de hacer clic en el botón **Play virtual machine** puede que aparezca la siguiente imagen, si es así, solo hay que hacer clic en **ok** y luego comenzará a iniciarse el sistema operativo.

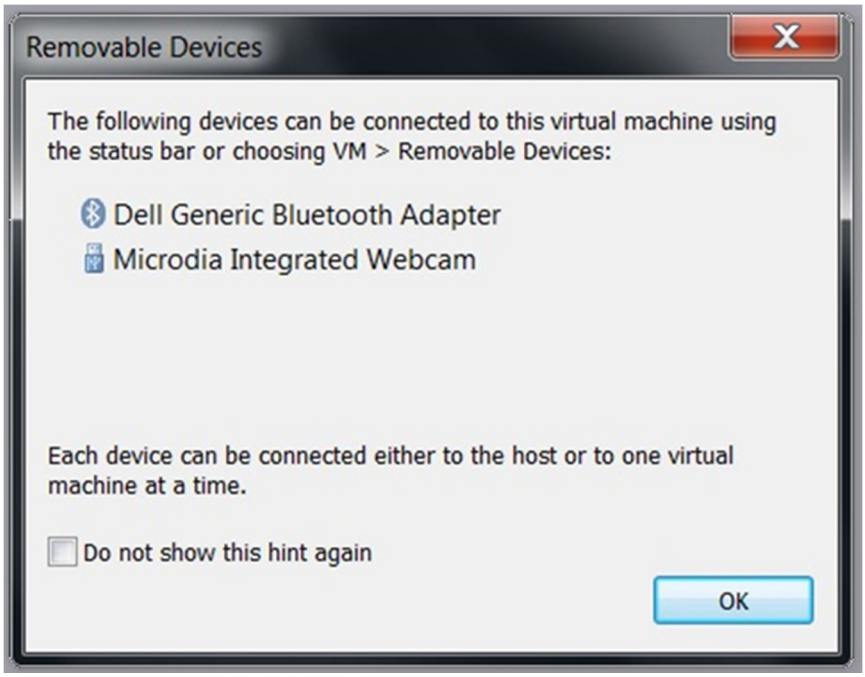


Imagen B.15. Opciones de los dispositivos de la computadora.

Imagen B.16. Ejecución del Sistema Operativo

Se observa en la imagen siguiente el escritorio del sistema operativo.

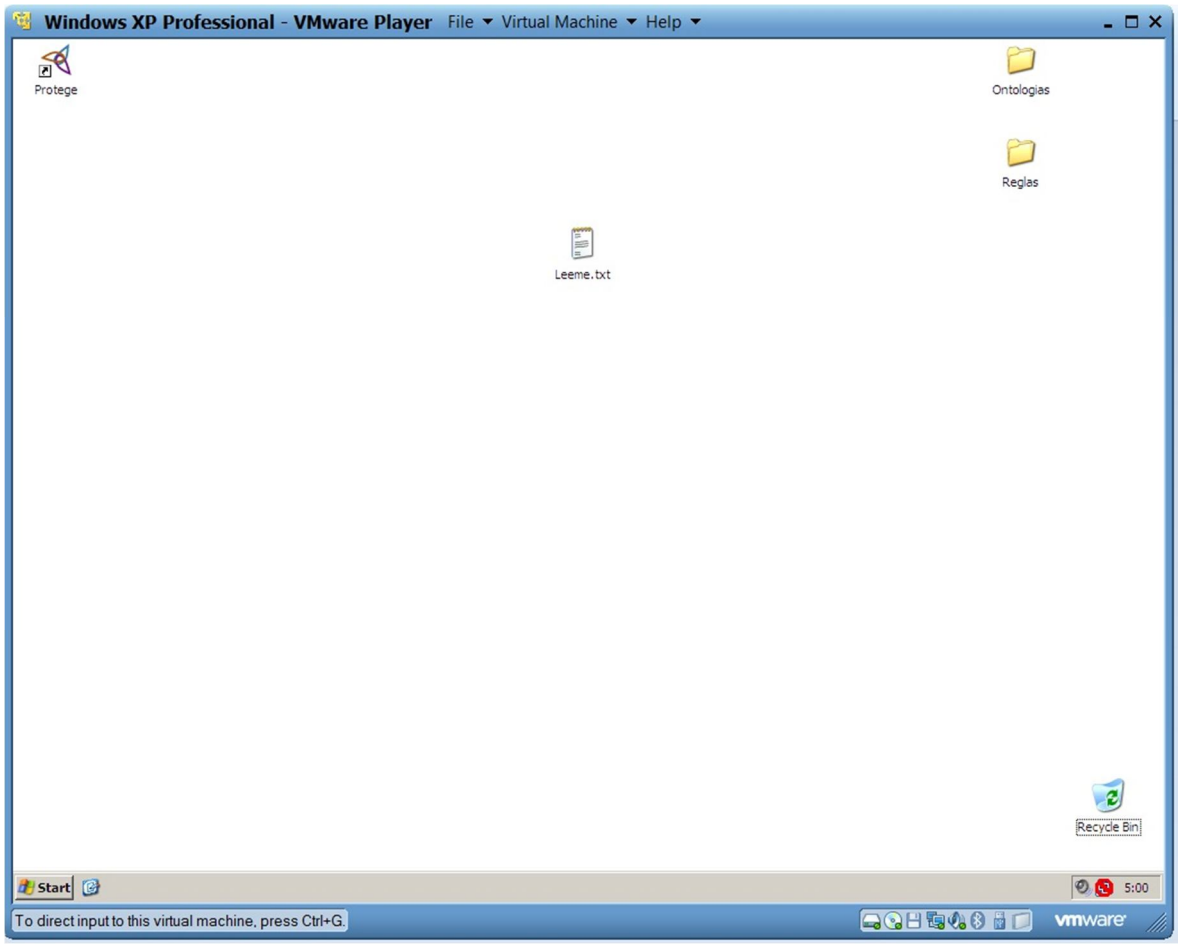
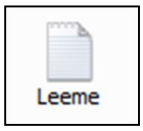


Imagen B.17. Escritorio del Sistema Operativo.

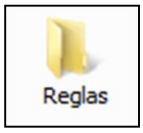
En él se observan 4 íconos:

Ícono para ejecutar Protégé



Archivo de ayuda

Carpeta con las ontologías y archivos del proyecto de Protégé



Carpetas con reglas de transformación

Cuando se llegue a esta instancia ya se podrá ejecutar Protégé y empezar a utilizar el prototipo. También se proporcionó un archivo de ayuda para el usuario aparte del manual de usuario que se adjunta en este trabajo.

Anexo C
Manual de Usuario

ANEXO C

MANUAL DE USUARIO

C.1. INTRODUCCIÓN

A continuación se desarrolla el procedimiento que los usuarios (diseñadores o desarrolladores) del prototipo deberán seguir para obtener los modelos de las interfaces de usuario finales para implementarlas en sus sistemas.

Este proceso cuenta con varias etapas, el relevamiento previo que el usuario deberá realizar, la instanciación de los distintos modelos representados en la ontología con los datos relevados, y la ejecución de las reglas.

Se asume que el usuario realizó la especificación de los requisitos del sistema y las características particulares de los usuarios de dicho sistema.

Además, el usuario deberá tener instalado el programa **Protégé** y el plugin del Jess para Protégé, **JessTab.** Como se explicó en el Anexo B, estas herramientas pueden obtenerse una vez instalada la aplicación que se encuentra en el DVD adjuntado a este trabajo.

C.2. INSTANCIACIÓN DE LA ONTOLOGÍA DE MODELOS

C.2.1. INSTANCIAR EL MODELO DE DOMINIO

En el modelo de dominio se deberá plasmar el diagrama de clases generado para el sistema. En este modelo se ingresarán los datos de las clases, métodos y atributos involucrados en el diagrama mencionado. A continuación en la imagen C.1 se observan los datos que se deben ingresar para instanciar una clase de tipo *Clase.* Para llegar a esta instancia el usuario deberá hacer clic en la pestaña **individuals** y seleccionar del menú de árbol izquierdo la clase que desea instanciar, en este caso *Clase.* Una vez seleccionada la clase deseada se debe hacer clic en el botón ***Create inference***  en la columna central como se observa en la imagen C.1.

El usuario deberá completar los campos *id* y *name* dejando para después *methods* y *atributes.* El campo *id* deberá ser incremental asignándole diferentes números a las diferentes *clases* creadas. El campo *name* deberá contener el nombre de la clase que se está creando, debiendo ser este un nombre representativo.

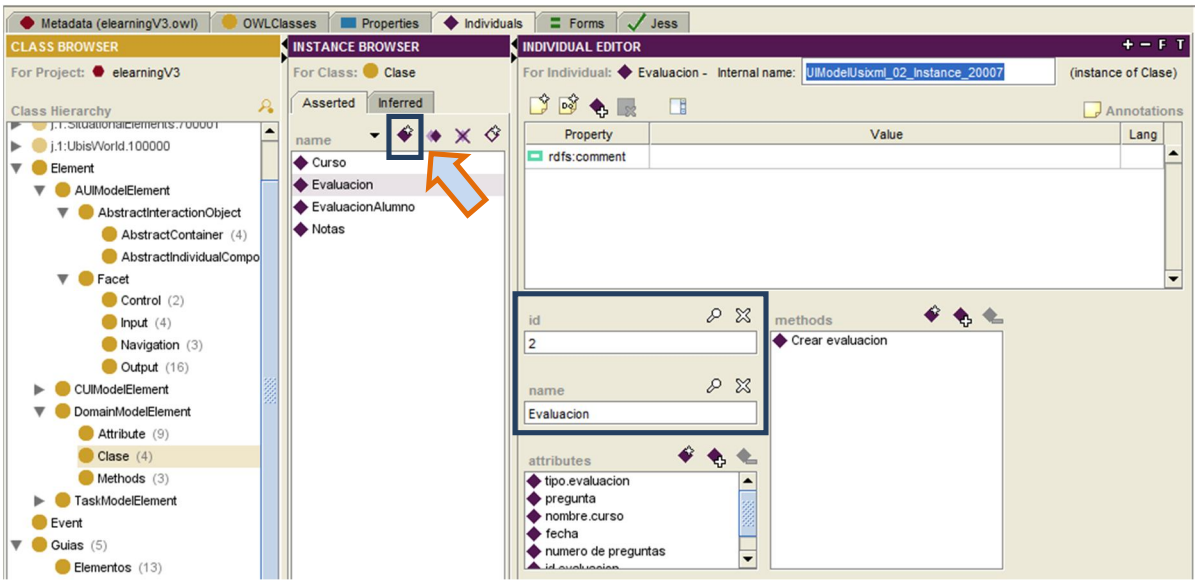


Imagen C.1. Instanciación de la clase *Clase* del modelo de *dominio.*

A continuación en la imagen C.2 se muestra el formulario para agregar una instancia de la clase *Methods,* al igual que la clase anterior se debe ingresar el campo *id*, además del nombre del método en el campo *name* que debe representar la acción que realiza tal método. Y el campo *Tasktype* que indicará el tipo de acción que realiza el método y este puede ser: start/go, stop/exit, select, choose, create, delete, modify, move, duplicate, toogle, view, monitor (definición de USIXML).

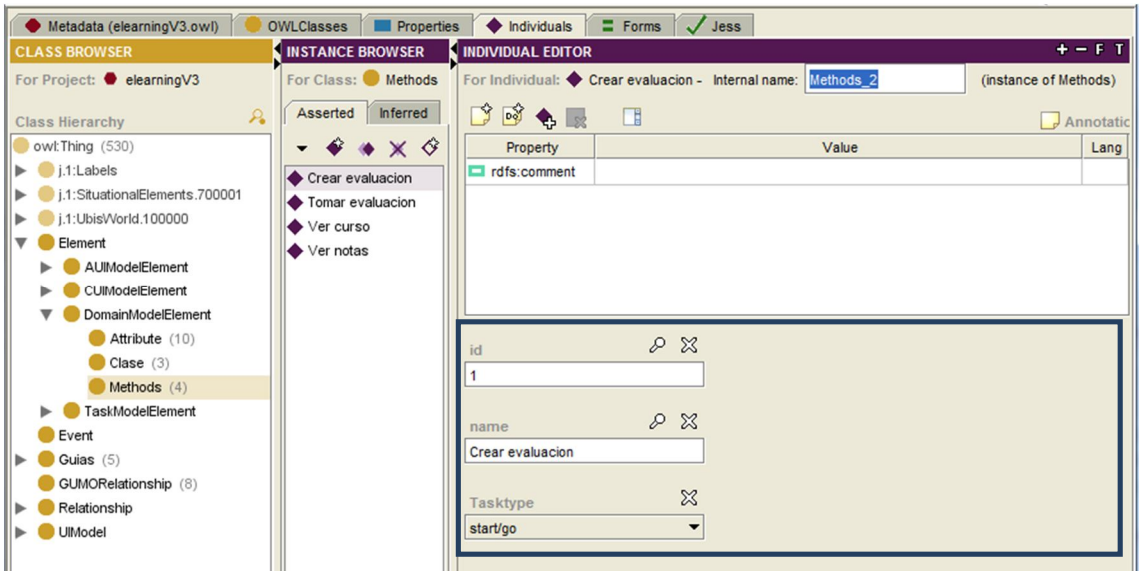


Imagen C.2. Instanciación de la clase *Methods* del modelo de *dominio.*

En la imagen C.3 se observa el formulario para agregar un *Atribute*, estos al igual que los atributos en los diagramas de clases representan la información que van a manipular las clases. Para definirlos se deben ingresar los campos: *id* con un valor numérico secuencial no repetido, el campo *dataType* con el nombre del tipo de dato que va a contener el

atributo y el campo *name* con el nombre del atributo que al igual que en los casos anteriores debe ser representativo.

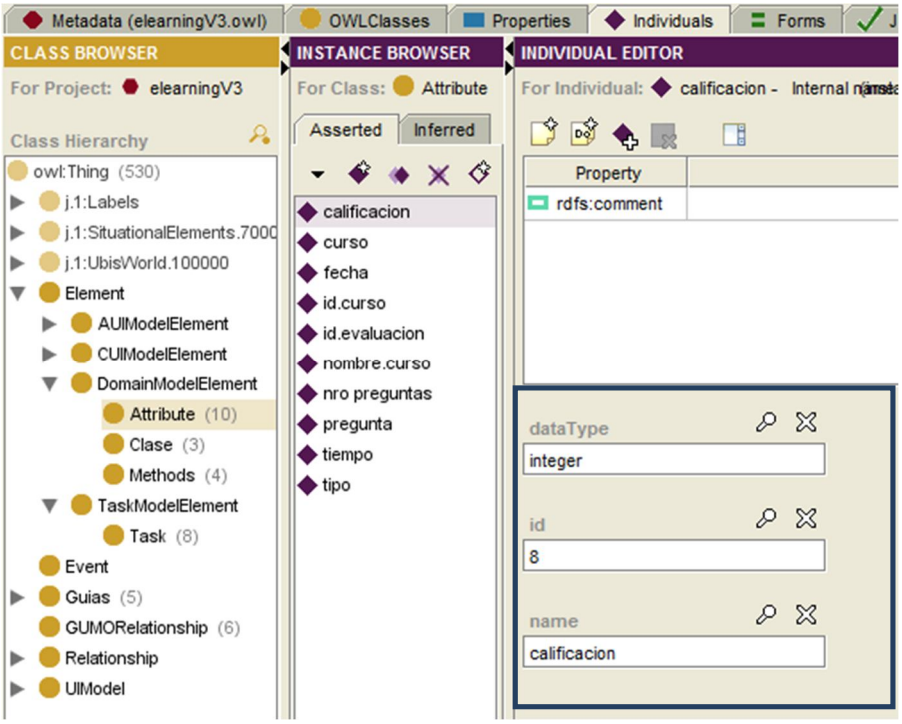


Imagen C.3. Instanciación de la clase *Atribute* del modelo de *dominio.*

Una vez generados todos los atributos y los métodos se debe regresar a la clase *Clase* y seleccionar para cada una de las instancias los métodos y atributos relacionados. Para ello se seleccionará una de las instancias del panel ***INSTANCE BROWSER*** (panel central) y se debe presionar el botón **Select existing resource** (ver imagen C.4) y seleccionar de la ventana emergente las instancias de la clase *atributte* relacionadas con la instancia de *Clase.*

En la imagen C.5 se observa la ventana emergente para seleccionar las instancias de la clase *atributte.* En esta ventana se seleccionan los atributos que corresponden a la instancia de *Clase*.

La imagen C.6 muestra la ventana Select Resources para seleccionar las instancias de la clase *method.* En esta ventana se seleccionan los métodos que corresponden a la instancia de *Clase*.

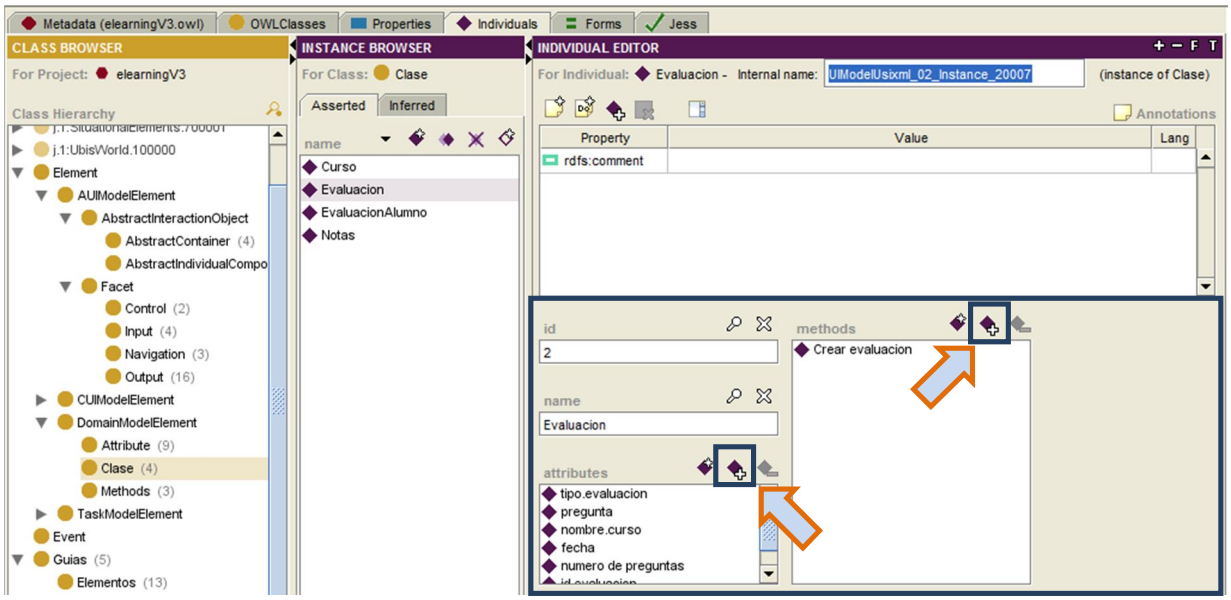


Imagen C.4. Instanciación de la clase *Clase* del modelo de *dominio*.

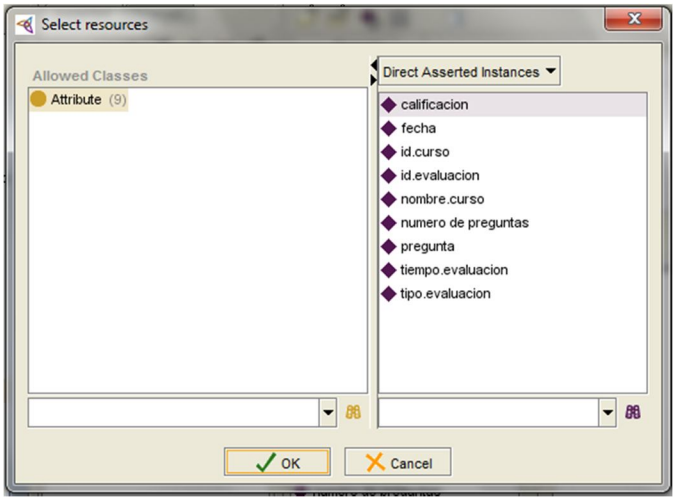


Imagen C.5. Ventana Select Resources para seleccionar instancias de la clase *attribute*.

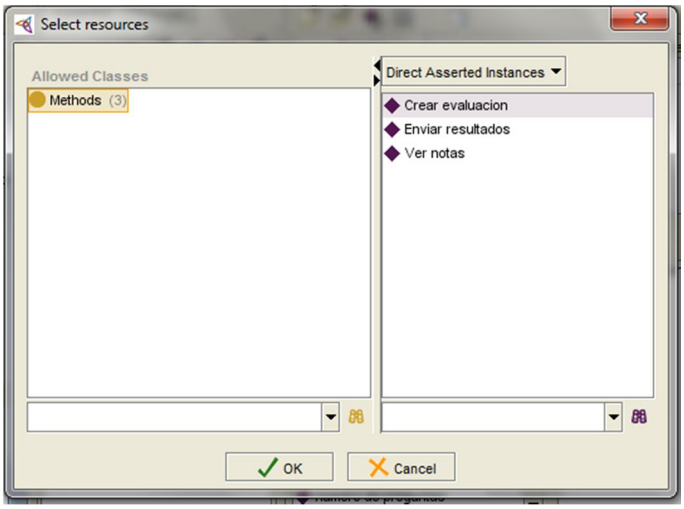


Imagen C.6. Ventana Select Resources para seleccionar instancias de la clase *methods*.

C.2.2. INSTANCIAR EL MODELO DE TAREAS

El modelo de tareas representa como su nombre lo indica las tareas que los usuarios podrán realizar en el sistema. En relación a lo explicado en el modelo de dominio, el modelo de tareas representa los métodos públicos y las acciones que hacen que dichos métodos se ejecuten. Por ejemplo la acción de presionar un botón o link en un sistema permite mostrar un listado que un usuario necesita visualizar, en este ejemplo tanto el botón como el listado son representadas en el modelo de tareas.

Para instanciar una tarea se debe acceder a la pestaña **individuals** en el menú de árbol de la izquierda y se debe seleccionar la clase *Element,* dentro de esta la subclase *TaskModelElement* y dentro de esta la clase Task. Una vez ubicado en esa clase se debe hacer clic en el botón ***Create inference*** y completar los datos del formulario.

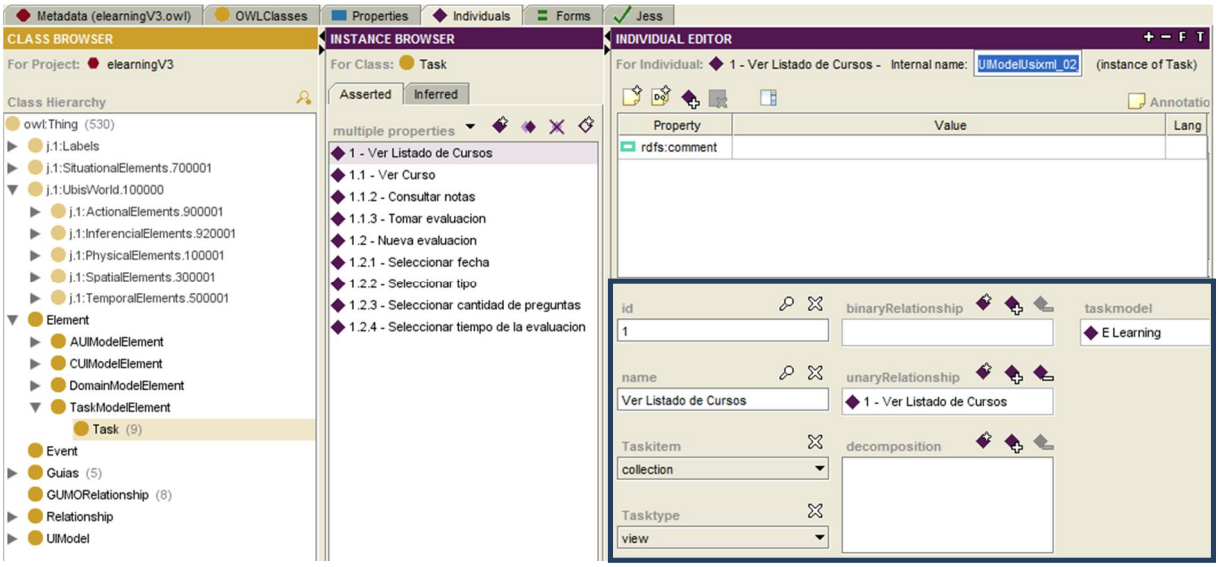


Imagen C.7. Instanciación de la clase *Task* del modelo de *tareas.*

Como las clases anteriores, se deberá completar el campo *id* con un número entero secuencial. El campo *name* debe contener el nombre de la tarea.

En el campo *Taskitem* se indica el contenido de la tarea (*operation*, *container*, *collection*, *element*) en el ejemplo se ve que este campo contiene el valor *collection*, esto quiere decir que la tarea va a tener un listado de algún elemento.

En Tasktype se va a indicar el tipo de tarea y este puede ser: “start/go”, “stop/exit”, “select”, “choose”, “create”, “delete”, “modify”, “move”, “duplicate”, “toogle”, “view”, “monitor”. Como se observa en el ejemplo este campo contiene la opción *view* esto quiere decir que la tarea muestra algo, y como se vio, el campo anterior es de tipo *collection*, entonces la tarea va a mostrar un listado.

El campo *Taskmodel* debe contener el nombre del modelo general que se está construyendo. El modelo general del ejemplo de este trabajo se denomina E-Learning.

En *decomposition* se debe seleccionar la tarea padre, esto indicará la jerarquía de las tareas, la dependencia entre ellas.

Luego se encuentran las clases que establecen las relaciones entre las tareas. La primera de ellas se modela mediante la clase *binaryRelationship* que permite establecer relaciones entre dos tareas diferentes, y la clase *unaryRelationship* que modela la relación de una tarea consigo misma.

Dentro de la clase *binaryRelationship* se observa la instanciación de la clase *Enabling* que indica que tareas habilitan a otras para ser ejecutadas.

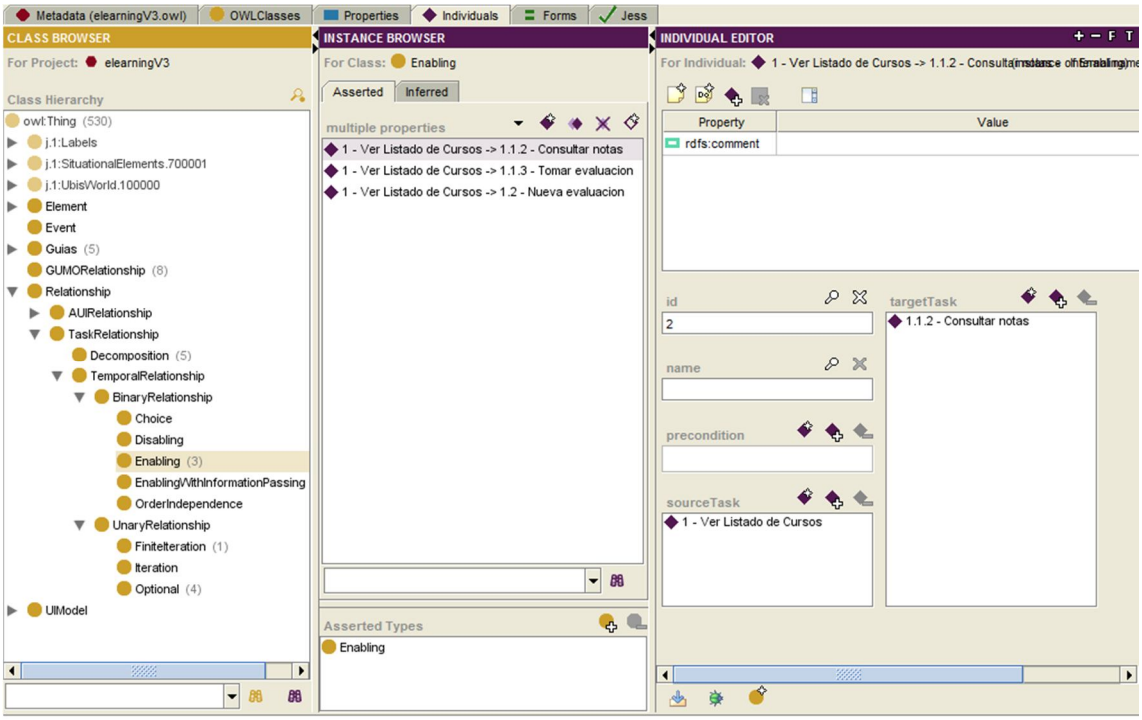


Imagen C.8. Instanciación de la clase *Enabling*.

Cuando una tarea no posee otra que la habilite es necesario establecer una relación *FiniteIteration*.

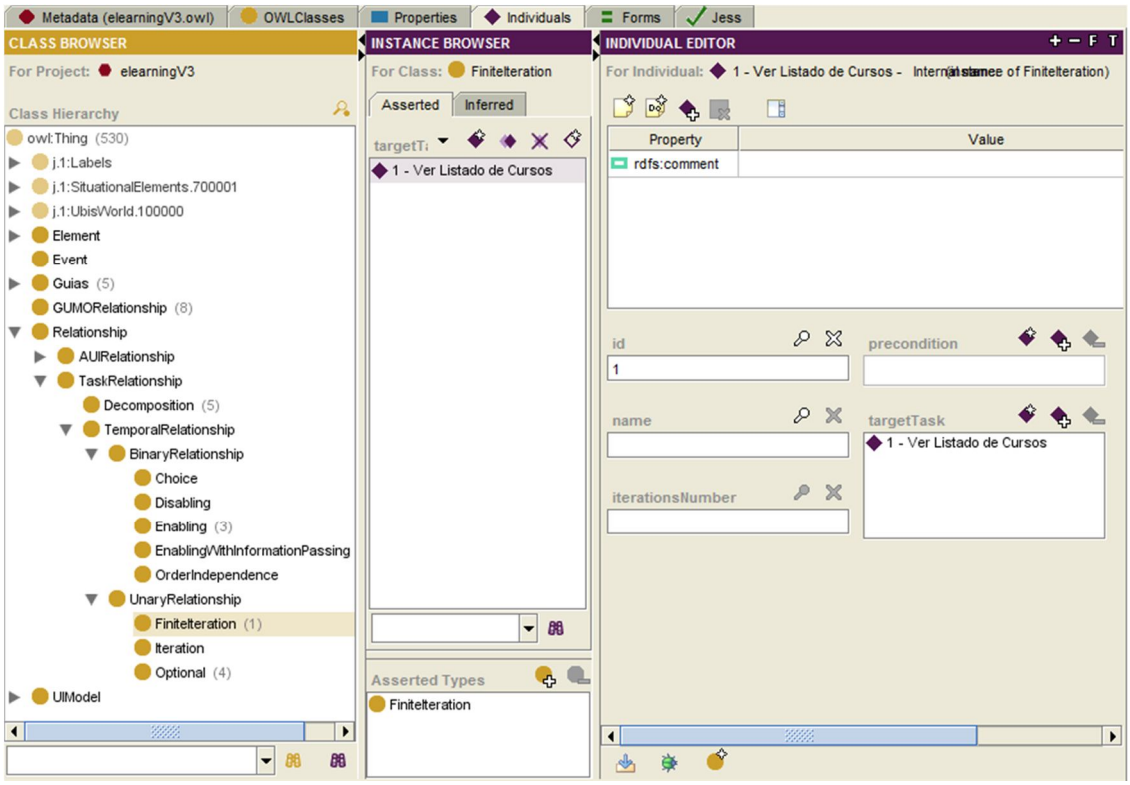


Imagen C.9. Instanciación de la clase *FiniteIteration.*

La última clase utilizada dentro de las *unaryRelationship* es la *Optional* que establece el grupo de tareas que pueden ser ejecutadas luego de realizar determinada tarea.

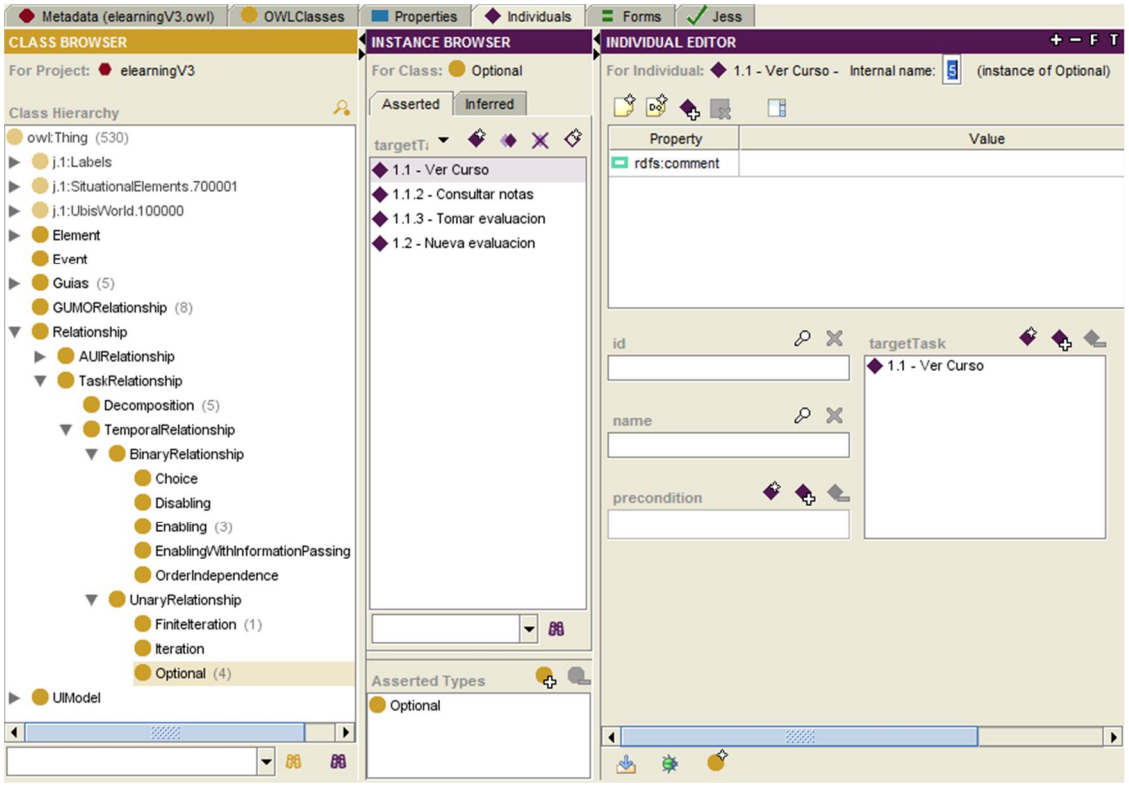


Imagen C.10. Instanciación de la clase *Optional.*

C.2.3. INSTANCIAR EL MODELO DE USUARIO

El modelo de usuarios representa los usuarios del sistema y las características personales de estos.

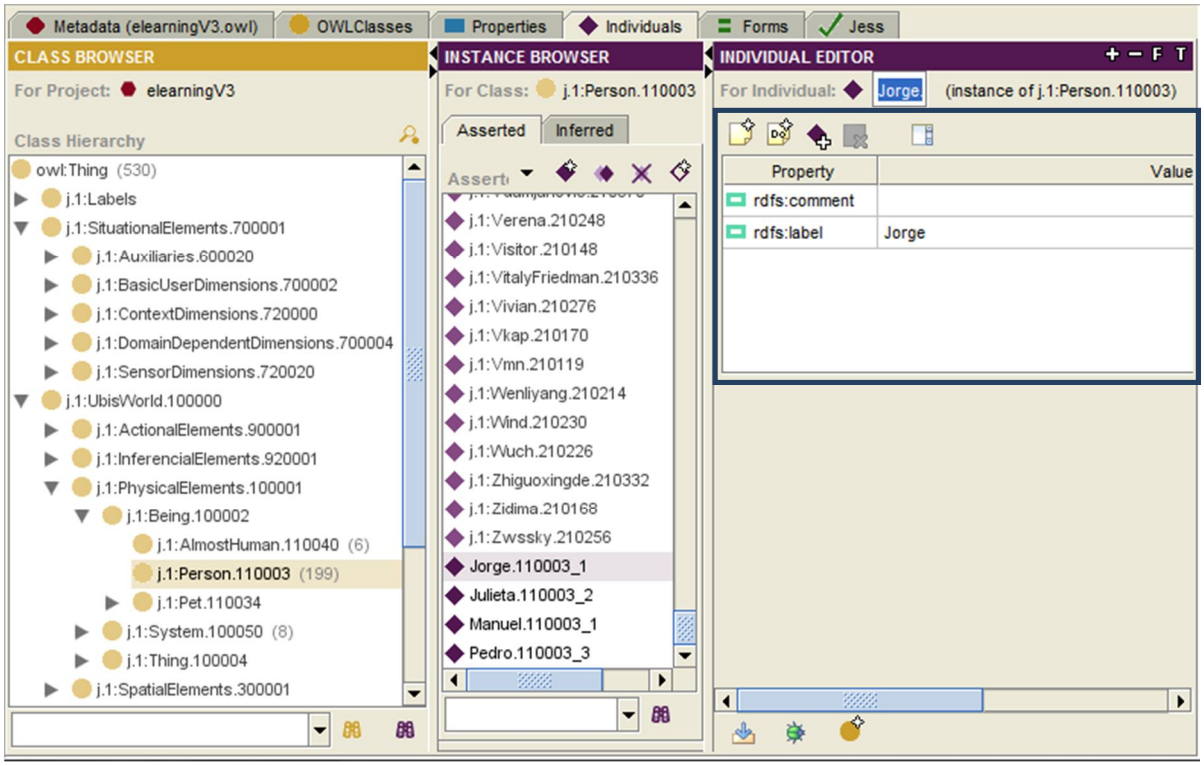


Imagen C.8. Instanciación de la clase *Person* del modelo de *usuarios.*

Primero se deberán crear los usuarios. Para dar de alta instancias de usuarios se debe acceder a *UbisWorld*, luego a *PhysicalElements* dentro de esta a *Being* y por último a *Person*. Al igual que con las clases anteriores se debe presionar el botón ***Create inference*** y completar los campos que se ven en el recuadro de la imagen C.8 Para el ejemplo se creó la instancia del usuario Jorge y se completó la propiedad *label* con el nombre del usuario.

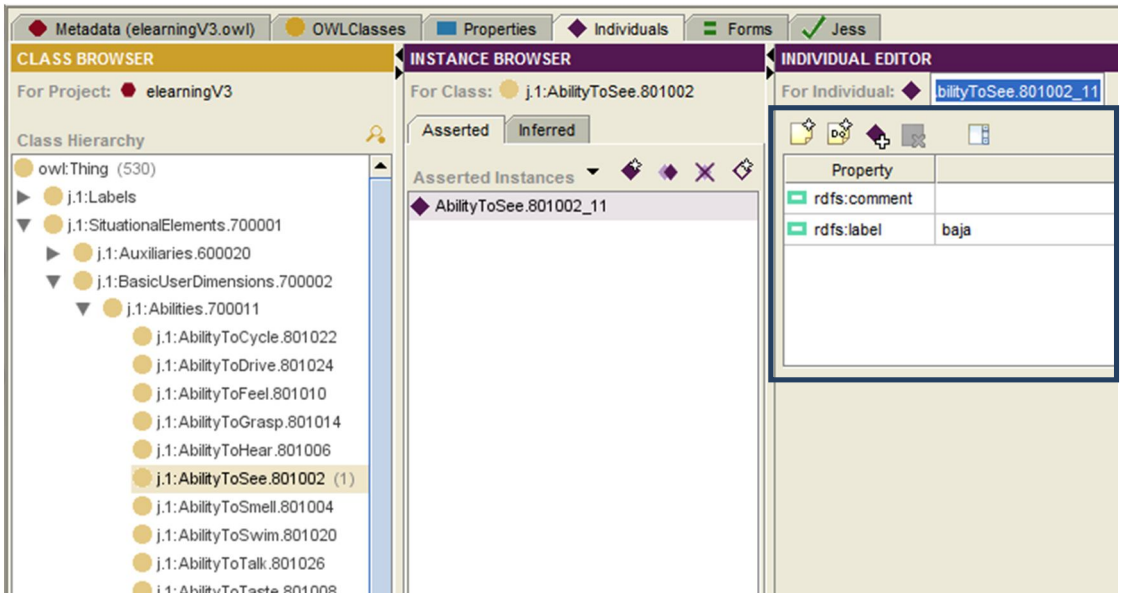


Imagen C.9. Instanciación de características generales del modelo de *usuarios*.

En la imagen C.9 se observa un ejemplo de una característica que posteriormente se podrá asignar a los usuarios, para dar de alta instancias de las características se debe ubicar en el listado la que se desea instanciar, una vez seleccionada la característica se procede como en las clases anteriores, se presiona el botón ***Create inference*** y se completan las propiedades del recuadro seleccionado en la imagen C.9. En el ejemplo la característica instanciada es la capacidad visual con un nivel bajo, esto se indica en la propiedad *label*.

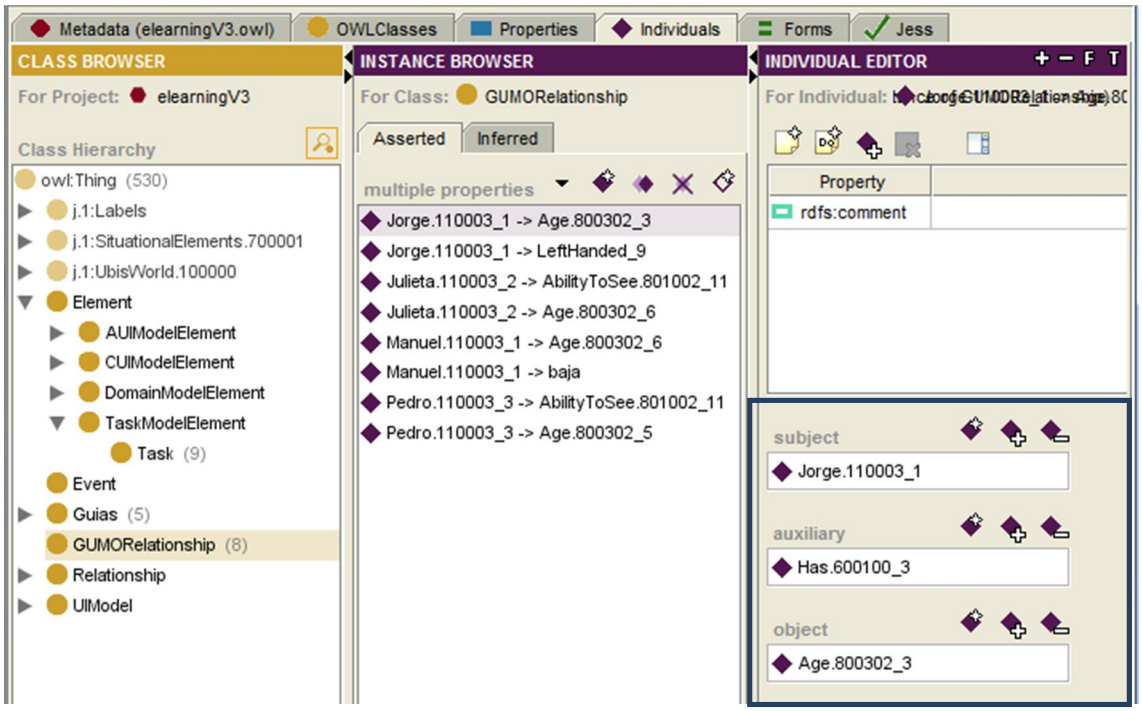


Imagen C.10. Instanciación de la *relación* entre características y usuarios.

Para instanciar la relación entre los usuarios y sus características se debe seleccionar la clase *GumoRelationship* y presionar el botón ***Create inference*** en el formulario que se observa remarcado en la imagen C.10 se deberán seleccionar los datos correspondientes a usuario, relación, característica. En el campo subject se presiona el botón **Select existing resource** y de la ventana Select resources (imagen C.11) se deberá seleccionar el usuario.

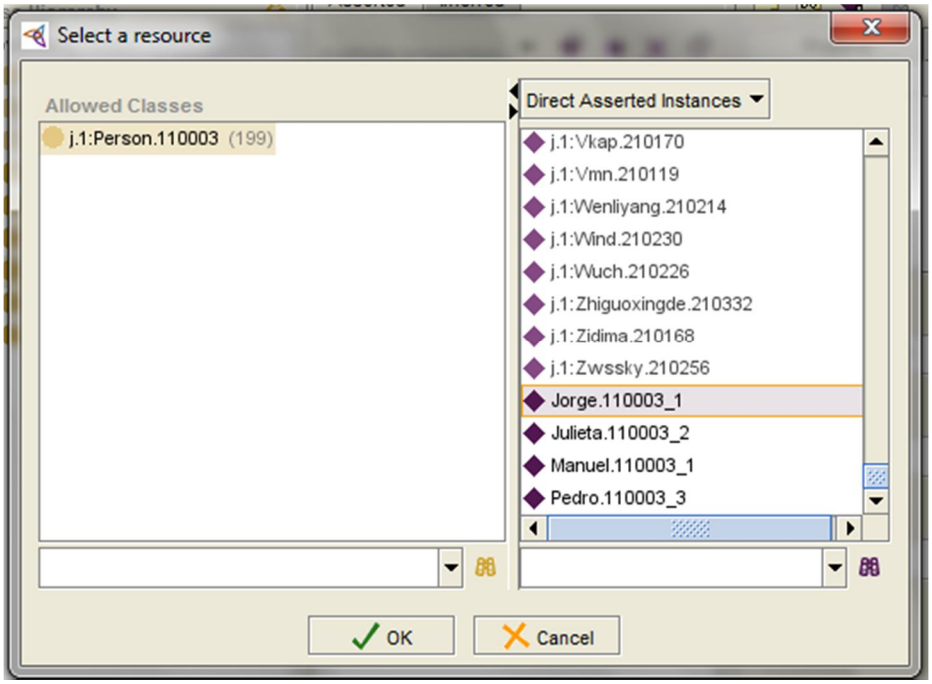


Imagen C.11. Ventana Select Resources para seleccionar instancias de la clase *Person*.

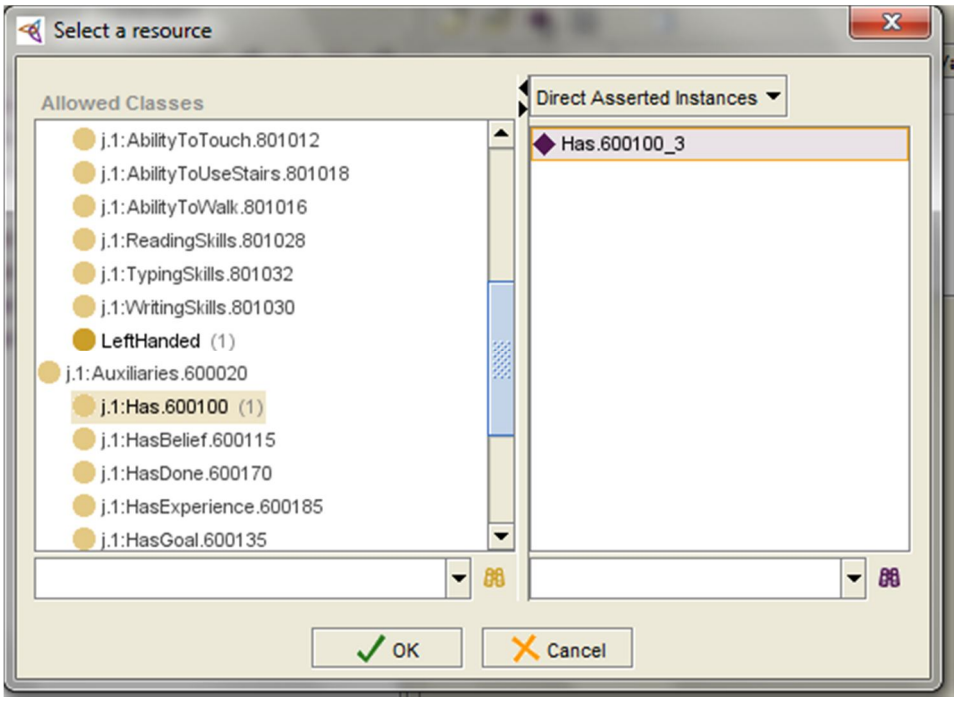


Imagen C.12. Ventana Select Resources con instancias de la clase *Auxiliaries*.

En el campo *auxiliary* se presiona nuevamente el botón **Select existing resource** y se selecciona el conector correspondiente (Imagen C.12). En el ejemplo "Has" (tiene).

Luego en el campo object se deberá presionar el botón y seleccionar la característica deseada para el usuario. En el ejemplo "Ability to see" (capacidad visual) en la imagen C.13**.**

Finalmente la instancia creada (imagen C.10) significa que el usuario "*Jorge tiene capacidad visual baja"*.

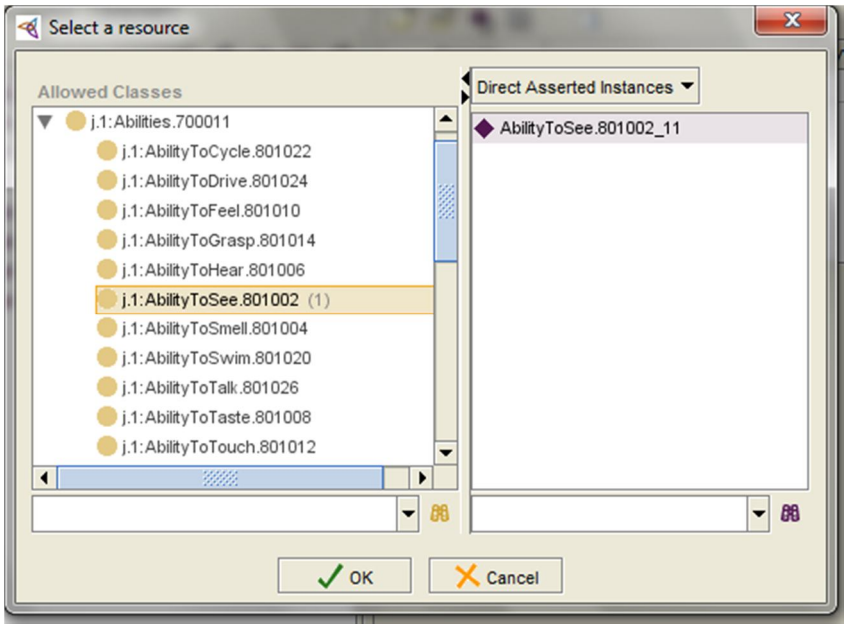


Imagen C.13. Ventana Select Resources con instancias de la clase *Abilities*.

C.2.4. INSTANCIAR EL MODELO DE MAPEO

El modelo de mapeo representa la relación entre el modelo de tareas y los modelos de usuario y de dominio. En este modelo se representa el diagrama de casos de uso, haciendo referencia a UML.

La relación entre el modelo de tareas y el de usuarios se establece mediante la clase isAllocatedTo, como se observa en la imagen C.14. Mediante la instanciación de esta clase es posible establecer que tareas pueden ser ejecutadas por cada usuario.

De la misma manera que en las clases anteriores se utilizará el botón para agregar instancias dadas de alta previamente. En el campo source se debe indicar la instancia de la tarea y en el campo target se debe indicar la instancia de usuario. En el ejemplo se especifica que el usuario *"Jorge puede acceder a la tarea Ver Listado de Cursos"*.

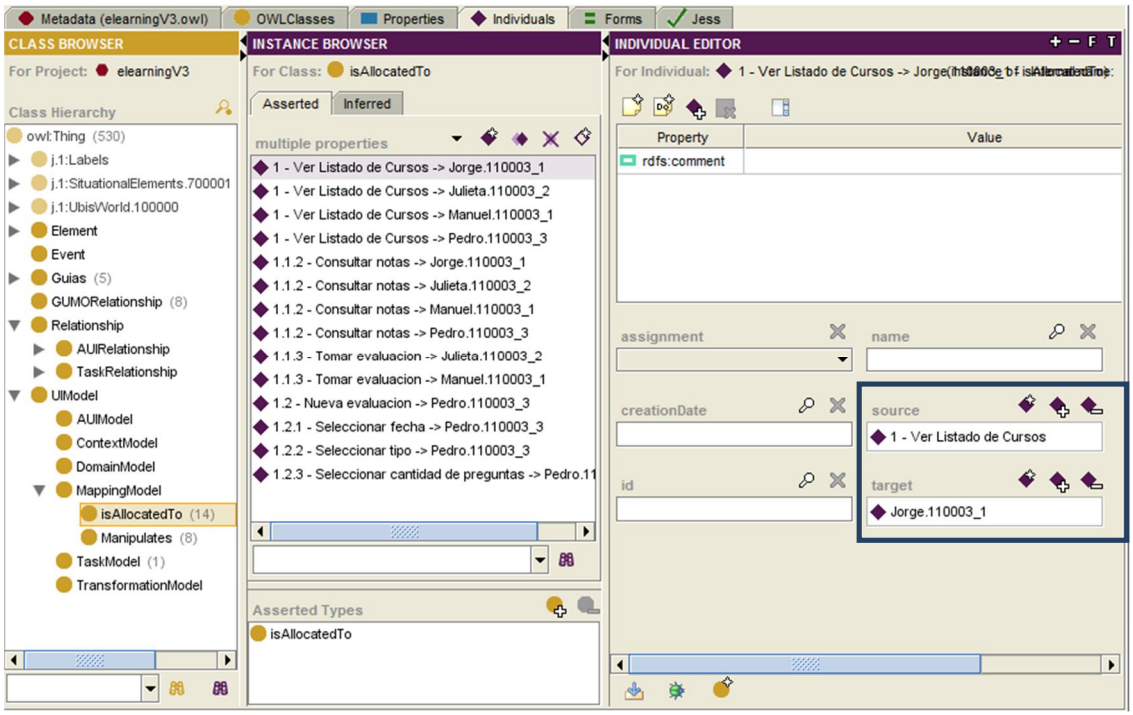


Imagen C.14. Instanciación de la *relación* entre tareas y usuarios. Clase *isAllocatedTo*.

Por último la relación entre el modelo de tareas y el de dominio se establece mediante la clase *Manipulates*. Esta relación permite indicar los elementos que son manipulados por las tareas y estos pueden ser atributos o clases del modelo de dominio.

Al igual que en las clases anteriores se debe acceder hasta la clase mencionada y presionar el botón del panel central para dar de alta una nueva instancia. Una vez que se presiona el botón se debe completar el formulario que se mostrará en el panel de la derecha y se deben completar los campos resaltados con recuadro en la imagen C.15.

Para completar el campo sourceManipulate se debe seleccionar una instancia de *Task* previamente generada presionando el botón. Lo mismo para el campo targetManipulate pero en este caso se debe seleccionar una instancia de *Attribute* o *Clase*. En el ejemplo se especifica que la tarea "*Ver Listado de Cursos"* manipula objetos de tipo *Curso*.

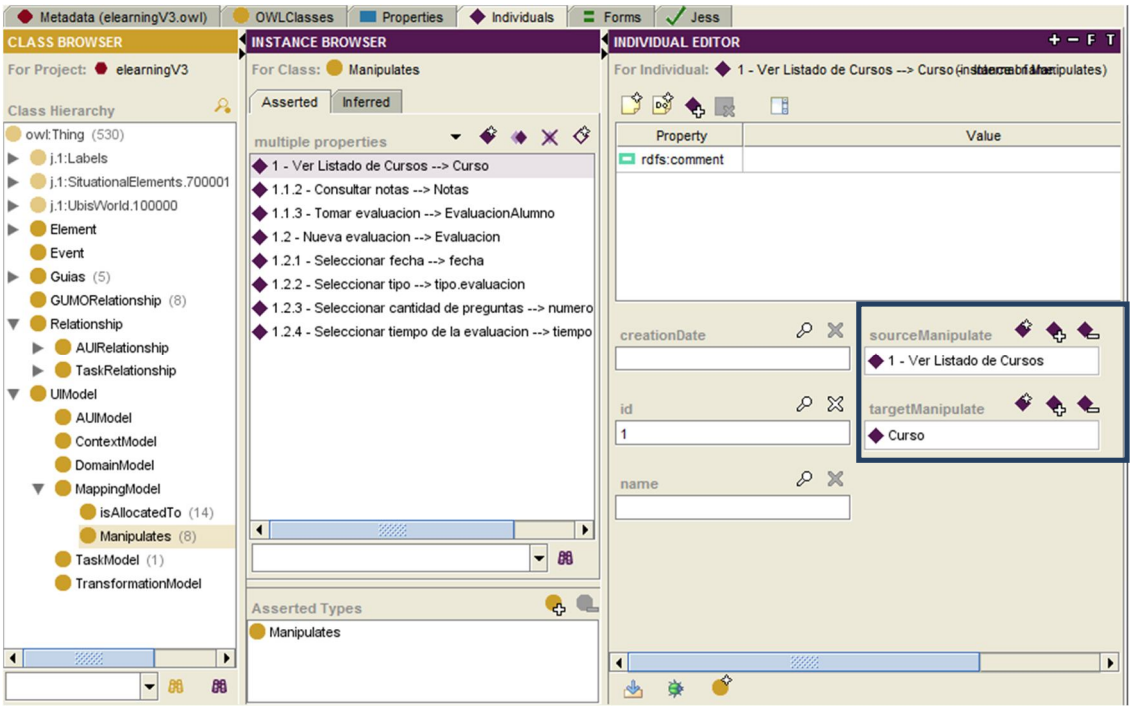


Imagen C.15. Clase *Manipulates*. Instanciación de la *relación* entre tareas y clases o atributos.

C.3. EJECUCIÓN DE REGLAS DE TRANSFORMACIÓN

C.3.1. CONSOLA DE JESS

Una vez instanciados todos los modelos se puede comenzar a probar el modelo abstracto generado y por lo tanto las interfaces concretas.

Se comenzará ejecutando las reglas que generan el modelo de abstracto y por último las que generan las interfaces concretas.

Para ejecutar las reglas es necesario situarse dentro de la pestaña de Jess. En la cual se pueden observar varias sub pestañas. La primera de ellas es la de la consola. Imagen C.16.

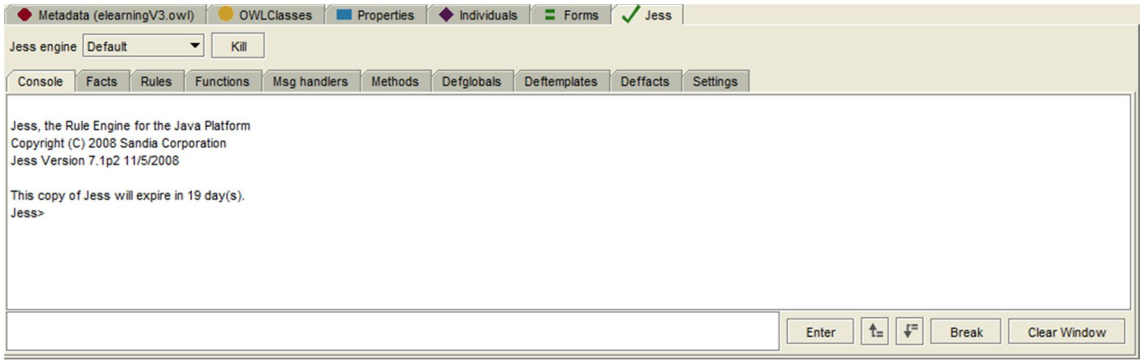


Imagen C.16. Consola Jess dentro de Protégé.

El conjunto de reglas está dividido en dos archivos:

1. Reglas - M Abstracto.clp: Posee el conjunto de reglas que generan las instancias en las clases de la ontología, que representa el modelo abstracto.
2. Reglas - M Concreto.clp: Posee el conjunto de reglas que generan las interfaces concretas.

Para ejecutar estos archivos se debe ingresar el siguiente código dentro del campo de entrada de la consola.

```
(batch "ruta al archivo/Reglas - M Abstracto.clp")
```

Es necesario ejecutar en primer lugar las reglas que generan el modelo abstracto ya que las reglas para las interfaces concretas utilizarán las instancias generadas anteriormente, pertenecientes al modelo abstracto. El código mencionado anteriormente es suficiente para crear las instancias del modelo abstracto.

C.3.2. GENERACIÓN DEL MODELO ABSTRACTO

Una vez ejecutado el código anterior se pueden observar las instancias generadas, Imagen C.17.

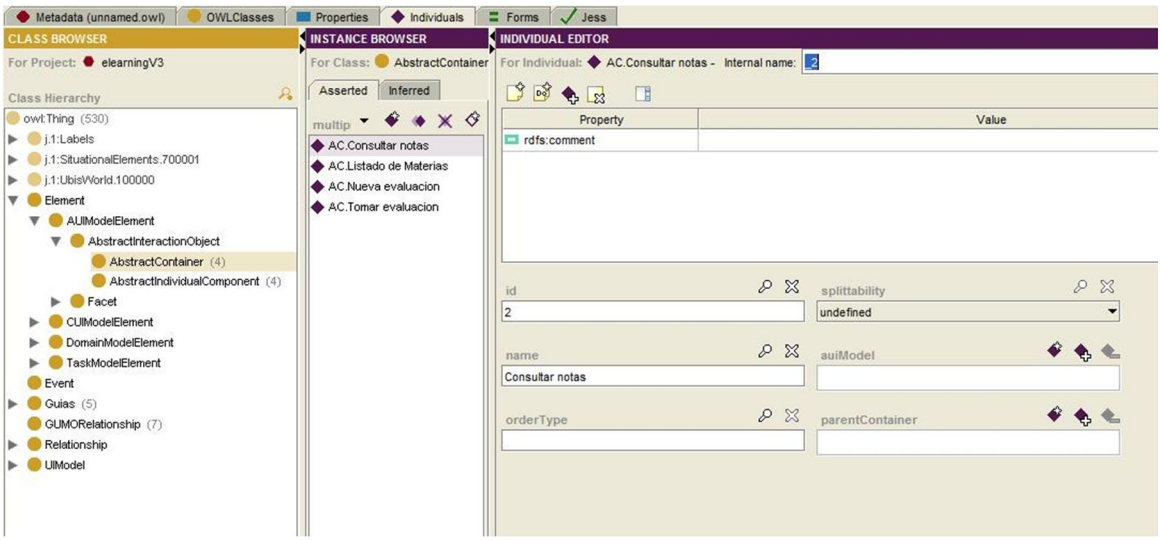


Imagen C.17. Instancias de clase *AbstractContainer*.

También dentro de la clase Facet se encontrarán instancias generadas por las reglas ejecutadas al introducir el código anterior (Imagen C.18). Dentro de la clase Facet se encuentran todos los elementos manipulados por las interfaces (*Control*, *Input*, *Navigation*, *Output*) y de estos se crean instancias para poder ser representadas en las interfaces concretas.

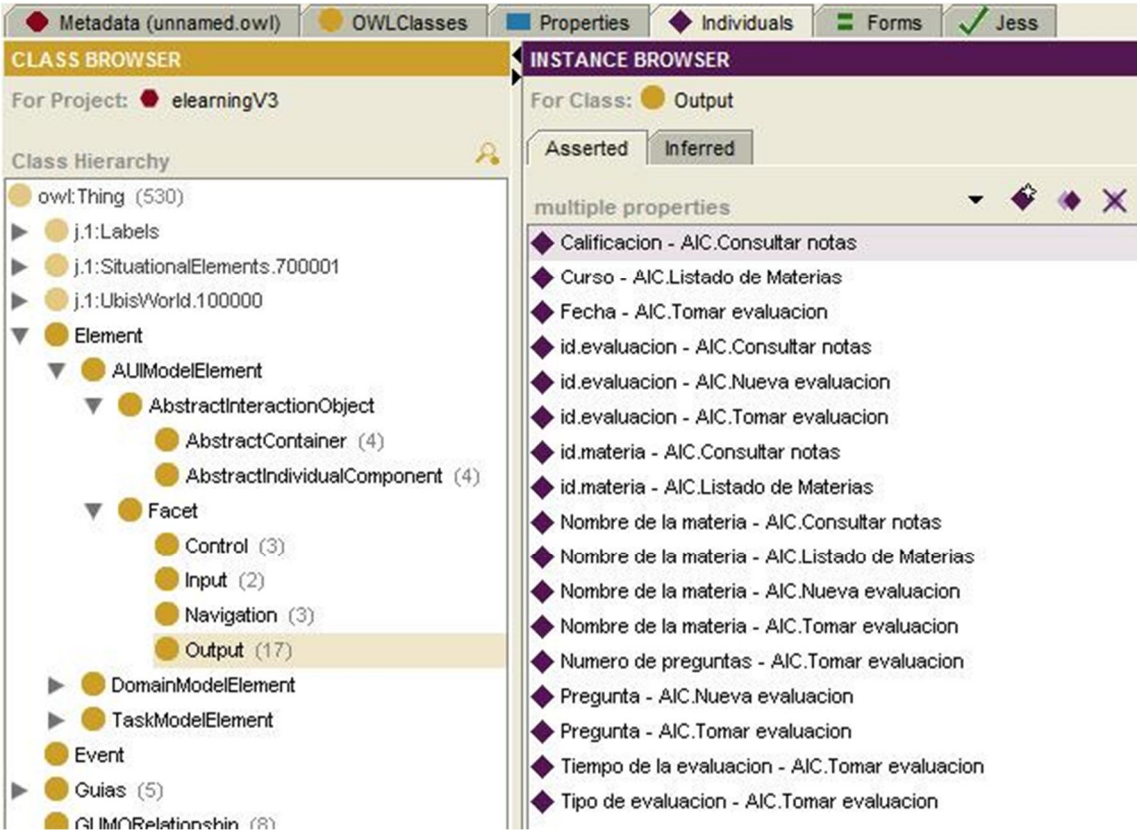


Imagen C.18. Instancias de clase *Output*

El modelo abstracto representa las interfaces en una forma independiente a dispositivos en los cuales se visualizarán las interfaces. Por esto no es necesario modificar las instancias si se necesitan otro tipo de interfaces concretas. Es necesario ejecutar nuevamente estas reglas si se realizaron cambios a los modelos instanciados antes del abstracto.

C.3.3. GENERACIÓN DE INTERFACES CONCRETAS

Para generar estas interfaces es necesario ejecutar la siguiente regla.

```
(batch "Ruta al archivo/Reglas - M Concreto.clp")
```

Los resultados de la ejecución de este código pueden observarse en la consola, Imagen C.19.

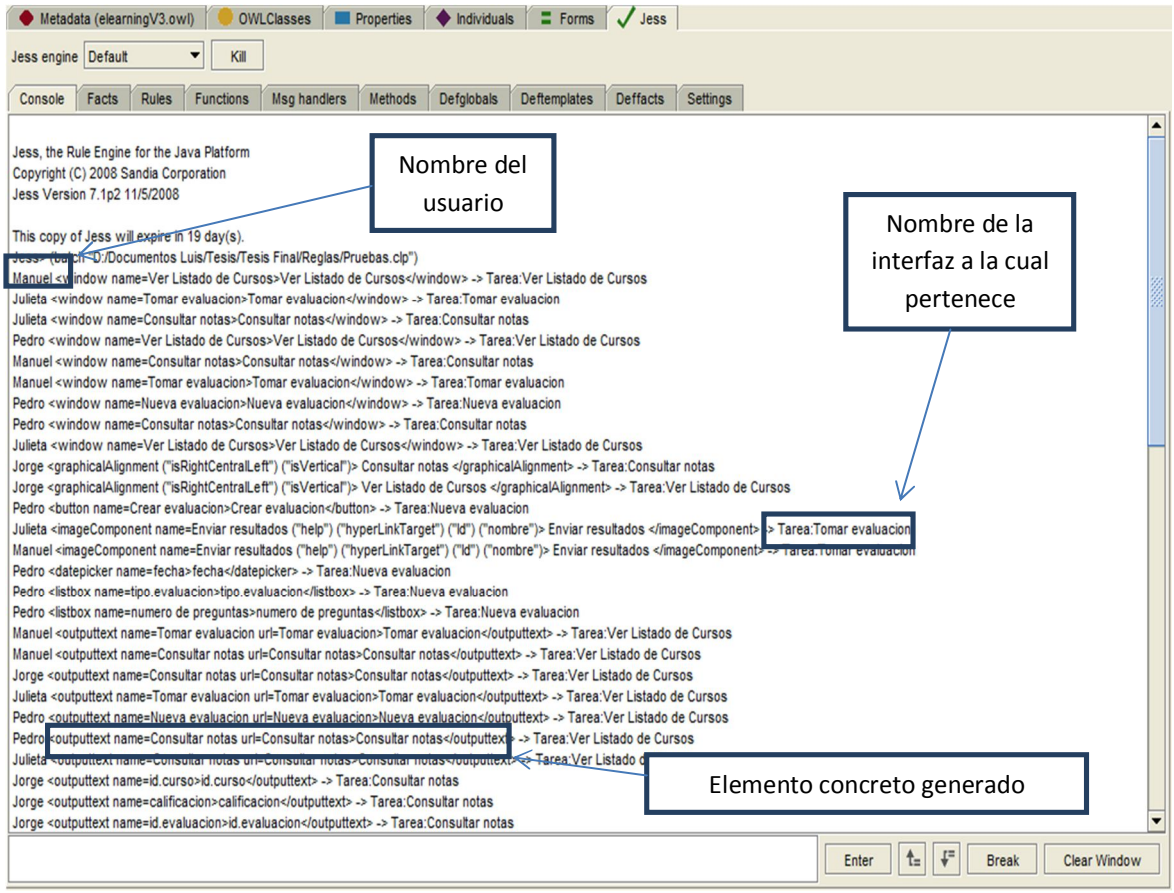


Imagen C.19. Elementos de las interfaces concretas luego de la ejecución de las reglas.

Las limitaciones del motor Jess como lenguaje y herramienta para este tipo de tareas son importantes. Aunque funciona correctamente dentro del entorno de Protégé.

El resultado arrojado por el prototipo en esta instancia del desarrollo son múltiples líneas de texto. Cada línea contiene tres elementos dentro de ella. El primero es el nombre del usuario, el segundo es el elemento concreto propiamente dicho con los atributos modificados o el mismo objeto modificado para adaptarse a las guías de diseño aplicadas a cada usuario. El tercer elemento es la interfaz a la que pertenece tal elemento.

En esta instancia el usuario (desarrollador o diseñador) deberá organizar estos elementos concretos según su usuario y la interfaz a la que pertenezca. Una vez organizada la información podrá interpretar cada una para volcarlas al desarrollo o diseño de las interfaces finales.

A continuación se muestra la salida del ejemplo utilizado en este trabajo, para que el usuario pueda observar con detalle como es la salida de este prototipo y le sirva de ejemplo para poder utilizar en su trabajo.

Manuel <window name=Ver Listado de Cursos>Ver Listado de Cursos</window> -> Tarea:Ver Listado de Cursos
Julieta <window name=Tomar evaluacion>Tomar evaluacion</window> -> Tarea:Tomar evaluacion
Julieta <window name=Consultar notas>Consultar notas</window> -> Tarea:Consultar notas
Pedro <window name=Ver Listado de Cursos>Ver Listado de Cursos</window> -> Tarea:Ver Listado de Cursos
Manuel <window name=Consultar notas>Consultar notas</window> -> Tarea:Consultar notas
Manuel <window name=Tomar evaluacion>Tomar evaluacion</window> -> Tarea:Tomar evaluacion
Pedro <window name=Nueva evaluacion>Nueva evaluacion</window> -> Tarea:Nueva evaluacion
Pedro <window name=Consultar notas>Consultar notas</window> -> Tarea:Consultar notas
Julieta <window name=Ver Listado de Cursos>Ver Listado de Cursos</window> -> Tarea:Ver Listado de Cursos
Jorge <graphicalAlignment ("isRightCentralLeft") ("isVertical")> Consultar notas </graphicalAlignment> -> Tarea:Consultar notas
Jorge <graphicalAlignment ("isRightCentralLeft") ("isVertical")> Ver Listado de Cursos </graphicalAlignment> -> Tarea:Ver Listado de Cursos
Pedro <button name=Crear evaluacion>Crear evaluacion</button> -> Tarea:Nueva evaluacion
Julieta <imageComponent name=Enviar resultados ("help") ("hyperLinkTarget") ("Id") ("nombre")> Enviar resultados </imageComponent> -> Tarea:Tomar evaluacion
Manuel <imageComponent name=Enviar resultados ("help") ("hyperLinkTarget") ("Id") ("nombre")> Enviar resultados </imageComponent> -> Tarea:Tomar evaluacion
Pedro <datepicker name=fecha>fecha</datepicker> -> Tarea:Nueva evaluacion
Pedro <listbox name=tipo.evaluacion>tipo.evaluacion</listbox> -> Tarea:Nueva evaluacion
Pedro <listbox name=numero de preguntas>numero de preguntas</listbox> -> Tarea:Nueva evaluacion
Manuel <outputtext name=Tomar evaluacion url=Tomar evaluacion>Tomar evaluacion</outputtext> -> Tarea:Ver Listado de Cursos
Manuel <outputtext name=Consultar notas url=Consultar notas>Consultar notas</outputtext> -> Tarea:Ver Listado de Cursos
Jorge <outputtext name=Consultar notas url=Consultar notas>Consultar notas</outputtext> -> Tarea:Ver Listado de Cursos
Julieta <outputtext name=Tomar evaluacion url=Tomar evaluacion>Tomar evaluacion</outputtext> -> Tarea:Ver Listado de Cursos
Pedro <outputtext name=Nueva evaluacion url=Nueva evaluacion>Nueva evaluacion</outputtext> -> Tarea:Ver Listado de Cursos
Pedro <outputtext name=Consultar notas url=Consultar notas>Consultar notas</outputtext> -> Tarea:Ver Listado de Cursos

```
Julieta <outputtext name=Consultar notas url=Consultar
notas>Consultar notas</outputtext> -> Tarea:Ver Listado de
Cursos
Jorge <outputtext name=id.curso>id.curso</outputtext> ->
Tarea:Consultar notas
Jorge <outputtext name=calificacion>calificacion</outputtext> -
> Tarea:Consultar notas
Jorge <outputtext name=id.evaluacion>id.evaluacion</outputtext>
-> Tarea:Consultar notas
Jorge <outputtext name=nombre.curso>nombre.curso</outputtext> -
> Tarea:Consultar notas
Julieta <timesensor
name=tiempo.evaluacion>tiempo.evaluacion</timesensor> ->
Tarea:Tomar evaluacion
Jorge <outputtext name=nombre.curso>nombre.curso</outputtext> -
> Tarea:Ver Listado de Cursos
Jorge <outputtext name=id.curso>id.curso</outputtext> ->
Tarea:Ver Listado de Cursos
Julieta <outputtext name=id.curso ("textSize")> id.curso
</outputtext> -> Tarea:Consultar notas
Manuel <vocalOutput name=id.curso ("isInterruptible") ("pitch")
("intonation") ("volume")> id.curso </vocalOutput> ->
Tarea:Consultar notas
Pedro <outputtext name=id.curso ("textSize")> id.curso
</outputtext> -> Tarea:Consultar notas
Julieta <outputtext name=calificacion ("textSize")>
calificacion </outputtext> -> Tarea:Consultar notas
Manuel <vocalOutput name=calificacion ("isInterruptible")
("pitch") ("intonation") ("volume")> calificacion
</vocalOutput> -> Tarea:Consultar notas
Pedro <outputtext name=calificacion ("textSize")> calificacion
</outputtext> -> Tarea:Consultar notas
Julieta <outputtext name=id.evaluacion ("textSize")>
id.evaluacion </outputtext> -> Tarea:Consultar notas
Manuel <vocalOutput name=id.evaluacion ("isInterruptible")
("pitch") ("intonation") ("volume")> id.evaluacion
</vocalOutput> -> Tarea:Consultar notas
Pedro <outputtext name=id.evaluacion ("textSize")>
id.evaluacion </outputtext> -> Tarea:Consultar notas
Julieta <outputtext name=pregunta ("textSize")> pregunta
</outputtext> -> Tarea:Tomar evaluacion
Manuel <vocalOutput name=pregunta ("isInterruptible") ("pitch")
("intonation") ("volume")> pregunta </vocalOutput> ->
Tarea:Tomar evaluacion
Julieta <outputtext name=nombre.curso ("textSize")>
nombre.curso </outputtext> -> Tarea:Tomar evaluacion
Manuel <vocalOutput name=nombre.curso ("isInterruptible")
("pitch") ("intonation") ("volume")> nombre.curso
</vocalOutput> -> Tarea:Tomar evaluacion
Pedro <outputtext name=id.evaluacion ("textSize")>
id.evaluacion </outputtext> -> Tarea:Nueva evaluacion
Pedro <outputtext name=pregunta ("textSize")> pregunta
</outputtext> -> Tarea:Nueva evaluacion
Julieta <outputtext name=tipo.evaluacion ("textSize")>
tipo.evaluacion </outputtext> -> Tarea:Tomar evaluacion
Manuel <vocalOutput name=tipo.evaluacion ("isInterruptible")
```

```
("pitch") ("intonation") ("volume")> tipo.evaluacion
</vocalOutput> -> Tarea:Tomar evaluacion
Pedro <outputtext name=nombre.curso ("textSize")> nombre.curso
</outputtext> -> Tarea:Nueva evaluacion
Julieta <outputtext name=nombre.curso ("textSize")>
nombre.curso </outputtext> -> Tarea:Consultar notas
Manuel <vocalOutput name=nombre.curso ("isInterruptible")
("pitch") ("intonation") ("volume")> nombre.curso
</vocalOutput> -> Tarea:Consultar notas
Pedro <outputtext name=nombre.curso ("textSize")> nombre.curso
</outputtext> -> Tarea:Consultar notas
Manuel <vocalOutput name=nombre.curso ("isInterruptible")
("pitch") ("intonation") ("volume")> nombre.curso
</vocalOutput> -> Tarea:Ver Listado de Cursos
Pedro <outputtext name=nombre.curso ("textSize")> nombre.curso
</outputtext> -> Tarea:Ver Listado de Cursos
Julieta <outputtext name=nombre.curso ("textSize")>
nombre.curso </outputtext> -> Tarea:Ver Listado de Cursos
Manuel <vocalOutput name=id.curso ("isInterruptible") ("pitch")
("intonation") ("volume")> id.curso </vocalOutput> -> Tarea:Ver
Listado de Cursos
Pedro <outputtext name=id.curso ("textSize")> id.curso
</outputtext> -> Tarea:Ver Listado de Cursos
Julieta <outputtext name=id.curso ("textSize")> id.curso
</outputtext> -> Tarea:Ver Listado de Cursos
Julieta <outputtext name=numero de preguntas ("textSize")>
número de preguntas </outputtext> -> Tarea:Tomar evaluacion
Manuel <vocalOutput name=numero de preguntas
("isInterruptible") ("pitch") ("intonation") ("volume")> número
de preguntas </vocalOutput> -> Tarea:Tomar evaluacion
Julieta <outputtext name=fecha ("textSize")> fecha
</outputtext> -> Tarea:Tomar evaluacion
Manuel <vocalOutput name=fecha ("isInterruptible") ("pitch")
("intonation") ("volume")> fecha </vocalOutput> -> Tarea:Tomar
evaluacion
Manuel <timesensor name=tiempo.evaluacion ("time+20")>
tiempo.evaluacion </timesensor> -> Tarea:Tomar evaluacion
Julieta <outputtext name=id.evaluacion ("textSize")>
id.evaluacion </outputtext> -> Tarea:Tomar evaluacion
Manuel <vocalOutput name=id.evaluacion ("isInterruptible")
("pitch") ("intonation") ("volume")> id.evaluacion
</vocalOutput> -> Tarea:Tomar evaluación
```