UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Título: Estimación temprana de proyectos de software mediante Léxico Extendido del Lenguaje y Puntos de Caso de Uso
Autores: Vido, Alan Jonatan
Director: Rossi, Gustavo
Codirector: Antonelli, Leandro
Asesor profesional:
Carrera: Licenciatura en Informática

Resumen

Actualmente existe un gran número de técnicas y herramientas para realizar estimaciones en los procesos de software, pero muchas de ellas requieren de gran volumen de información del proyecto que se está analizando, dificultando una estimación temprana del esfuerzo requerido para desarrollar dicho proyecto.
Aquellos analistas que trabajan con el Léxico Extendido del Lenguaje, al contar con este modelo en etapas tempranas del software, pueden inferir ciertas características del proyecto, como pueden ser los Casos de Uso, las clases y entidades de base de datos que formaran parte del diseño del proyecto.
Por otro lado, existen técnicas de estimación de esfuerzo ampliamente utilizadas y estandarizadas que se valen de estas características, como por ejemplo Puntos Caso de Uso, pero que en una etapa temprana de elicitación de requerimientos no son aplicables por falta de información.
Este trabajo pretende brindar a los usuarios que utilizan Léxico Extendido del Lenguaje en su proceso de elicitación de requerimientos, una herramienta que, a partir de la información recabada en las etapas tempranas de dicho proceso, proporcione una estimación del esfuerzo necesario para realizar el proyecto, basada en un método ampliamente utilizado y estandarizado.

Palabras Claves

Estimación de esfuerzo, Métricas, Gestión de Proyectos, Léxico Extendido del Lenguaje, Puntos Caso de Uso, Elicitación de Requerimientos

Trabajos Realizados

Durante el transcurso del presente trabajo, se analizó la bibliografía, haciendo énfasis en los temas que marcan el estado del arte de los conceptos abordados. Se diseñó un método de tipo pipeline, que sintetiza el conocimiento adquirido, a fin de lograr estimaciones de esfuerzo en etapas tempranas de la elicitación de requisitos. Se implementó una aplicación Web, que da soporte al método planteado, lo que requirió del análisis e integración de las tecnologías disponibles al momento de su desarrollo, junto con el diseño de heurísticas y los algoritmos que las implementan, a fin de inferir los datos que la herramienta requiere.

Conclusiones

La realización del presente trabajo concluye con la presentación de un método, que permite realizar una estimación del esfuerzo requerido para el desarrollo de una aplicación en etapas tempranas de la elicitación de requisitos. Para dar soporte a dicho método se desarrolló una aplicación web, que tomando como entrada el Léxico Extendido del Lenguaje de una aplicación y un conjunto de datos por parte del usuario, realiza una estimación según el método de Puntos Caso de Uso, obteniendo una estimación del esfuerzo necesario para el desarrollo de dicha aplicación.

Trabajos Futuros

Para continuar la línea de investigación y desarrollo seguida por el presente trabajo, se podría hacer un análisis estadístico de las estimaciones obtenidas a lo largo de distintos proyectos y los valores reales relativos al esfuerzo, a fin de mejorar los parámetros y factores de ajuste que utilizan las estimaciones en el método desarrollado, logrando de esta manera, estimaciones más precisas.
Por otro lado, se podría integrar la herramienta con otras técnicas de estimación y/o datos de entrada, permitiendo su utilización en etapas avanzadas de la elicitación de requisitos, desarrollo y mantenimiento del producto para mejorar su gestión.

Fecha de la presentación: Noviembre de 2014

Contenido

Índice de Ilustraciones

Índice de Tablas

A Carlita, mi inspiración,

A Clara, mi apoyo incondicional,

A mi familia, que me brindo la posibilidad de estar realizando este trabajo,

A mis amigos y compañeros de trabajo, que me soportaron todo este tiempo,

A Leandro Antonelli, por guiarme en el desarrollo de esta tesina.

Introducción

El objetivo de esta tesina es brindar los elementos necesarios para lograr una estimación temprana de un proyecto de software, mediante una herramienta automatizada, que utilice como entrada la información recabada en las primeras etapas de elicitación de requerimientos y un conjuntos de factores representativos del proyecto en cuestión, y produzca como salida una estimación del esfuerzo necesario para su realización. Para alcanzar dicho objetivo se brindará tanto un marco conceptual como una herramienta que ayude dentro de este marco.

Actualmente existe un gran número de herramientas y metodologías para realizar estimaciones en los procesos de software, pero muchas de ellas requieren de gran volumen de información del proyecto que se está analizando, dificultado una estimación temprana del esfuerzo requerido para desarrollar dicho proyecto.

Aquellos analistas que trabajan con el Léxico Extendido del Lenguaje (LEL, del inglés Lenguage Extended Lexicon) [**1**] [**2**], al contar con este modelo en etapas tempranas del software, pueden inferir ciertas características del proyecto, como puede ser los Casos de Uso (UC, del inglés, Use Cases) probables, que aparecerán en posteriores etapas de la elicitación de requerimientos, así como también las clases y entidades de base de datos que formaran parte del diseño del proyecto.

Por otro lado, existen técnicas de estimación de esfuerzo ampliamente utilizadas y estandarizadas que se valen de las características antes mencionadas, como por ejemplo Puntos Caso de Uso (UCP, del inglés, Use Case Points) [**3**] [**4**], pero que en una etapa temprana de elicitación de requerimientos no son aplicables por falta de información.

Con este trabajo se pretende brindar a los usuarios que utilizan LEL [**5**] [**6**] en su proceso de elicitación de requerimientos, una herramienta que, a partir de la información recabada en las etapas tempranas de dicho proceso, proporcione una estimación del esfuerzo necesario para realizar el proyecto, basada en un método ampliamente utilizado y estandarizado como es UCP, permitiendo la integración de la herramienta a su proceso con un mínimo de esfuerzo de adaptación del formato del LEL y facilitando de esta manera decisiones en la viabilidad, costos, composiciones de equipos de trabajo, y demás factores involucrados en la planificación de un proyecto.

Este capítulo tiene por objetivo presentar el trabajo realizado e introducir al lector en el contexto donde nace la motivación, partiendo de las definiciones básicas en la gestión de proyectos, pasando por las métricas y su importancia dentro de la gestión, para luego introducir los conceptos básicos de LEL, terminado con la presentación del método, objeto del presente trabajo.

Según la IEEE [**7**], la gestión en la ingeniería de software puede definirse como, la aplicación de actividades administrativas (planeación, coordinación, medición, monitorización, control y reporte) para asegurar que el desarrollo y el mantenimiento del software sea sistemático, disciplinado y cuantificable.

Para Pressman [**8**], la gestión eficaz de un proyecto de software, está basada en las cuatro P's, que corresponden al Personal, al Producto, al Proceso y al Proyecto, ya que el personal debe organizarse en equipos eficaces, motivados para hacer un software de alta calidad y coordinados para alcanzar una comunicación efectiva. Los requisitos del producto deben comunicarse desde el cliente al desarrollador, dividirse (descomponerse) en las partes que lo constituyen y distribuirse para que trabaje el equipo de software. El proceso debe adaptarse al personal y al problema. Se selecciona una estructura común de proceso, se aplica un paradigma apropiado de ingeniería del software y se elige un conjunto de tareas para completar el trabajo. Finalmente, el proyecto debe organizarse de una manera que permita al equipo de software tener éxito.

Sommerville [**9**] se refiere a la gestión de proyectos de software como una parte esencial de la ingeniería de software y hace énfasis en que una buena gestión no puede garantizar el éxito de un proyecto, pero seguramente una mala gestión garantice su fracaso, alterando su cronograma, incrementado su costo o incumpliendo los requerimientos.

La ingeniería de software es diferente a otras ingenierías, lo que dificulta su gestión, las principales diferencias con las demás ingenierías radican en que el producto es intangible ya que el software es intangible, no existen procesos del software estándar, debido a que estos varían de una organización a otra. Por otro lado, también hay que considerar que los grandes proyectos de software son, a priori, únicos, por lo tanto son diferentes a proyectos previos y en consecuencia es muy difícil anticipar los problemas que surgirán a lo largo de su ciclo de vida.

Todos estos factores contribuyen a que la gestión del desarrollo del software sea un proceso complejo, en donde intervienen diversos actores, cada uno con diferentes capacidades y conocimientos previos, que deben realizar tareas distintas y variadas de forma coordinada. Esta complejidad, sumada a que en las etapas iniciales de un proyecto no se tiene una gran cantidad de conocimiento del software a desarrollar, provoca que sea difícil definir un cronograma preciso [**10**]. Las imprecisiones en estos cronogramas generan complicaciones en el cumplimiento de las tareas que definen, originando retrasos o incrementos de costos a los que es muy difícil sobreponerse [**11**].

Partiendo de lo expuesto, es evidente que cuanto más conocimiento se tenga del proyecto en las etapas iniciales, se tendrán mayores chances de producir cronogramas verosímiles [**12**], con hitos alcanzables en tiempo y costos, será más fácil definir la conformación de los equipos que llevarán a cabo las diferentes tareas, desde la gestión hasta el desarrollo, logrando de esta manera incrementar las posibilidades de que un proyecto finalice en tiempo y forma, tal como se lo pacta con el cliente.

Para poder predecir o estimar la duración de una tarea debemos contar con información que nos conduzca a dicho resultado, es decir, necesitamos de alguna manera medir la tarea a realizar, y luego utilizar ésta medida para obtener el esfuerzo que conllevará la tarea, y así poder planificarla

correctamente. Según la norma ISO 14598-1 [**13**] una métrica es un método de medición definido y su escala de medición, la medición es la actividad que usa la definición de la métrica para producir el valor de una medida, una medida es un número o una categoría asignada a un atributo de una entidad.

Es importante medir el proceso de ingeniería de software y el producto que se elabora, ya que es la forma más objetiva de comprender y mejorar el proceso de desarrollo y el producto que se elabora. Si no se realizan mediciones, no hay forma de determinar si se está mejorando, las decisiones se basan sólo en evaluaciones subjetivas, lo que puede llevar a malas estimaciones o interpretaciones erróneas del proceso. Para establecer objetivos de mejora es necesario conocer el estado actual de desarrollo del software.

Las métricas de software son observaciones periódicas sobre algunos atributos o aspectos del producto y proceso de software. Proveen una base para el desarrollo y validación de los modelos del desarrollo de software y pueden utilizarse para mejorar la productividad y la calidad. Por lo tanto, la medición se emplea para establecer una línea base del proceso, a partir de la cual se evalúan las mejoras. La línea base son datos recopilados en proyectos previos de desarrollo de software, que contienen medidas de proyectos y métricas derivadas de estos. Los datos de la línea base deben tener los siguientes atributos: razonablemente precisos, deben recopilarse de tantos proyectos como sea posible, las medidas deben ser consistentes y las aplicaciones que se están estimando deben ser similares [**12**].

Las métricas se evalúan y se produce un conjunto de indicadores que guían el proyecto o proceso, estos indicadores se pueden utilizar para: evaluar la estabilidad y capacidad del proceso, interpretar los resultados de las observaciones, predecir costos y recursos para el futuro, proveer líneas base, graficar tendencias e identificar oportunidades de mejora. Una vez obtenidos los indicadores se debe ejecutar un control para verificar que el proceso se comporte consistentemente, identificar las áreas donde éste se encuentra, o no, bajo control, y aplicar las medidas correctivas donde sea necesario. Se debe también analizar los resultados y compararlos con promedios de proyectos similares anteriores realizados dentro de la organización, para generar conclusiones y establecer tendencias para los diferentes comportamientos.

De acuerdo a Kan [**14**], “Las métricas de software pueden ser clasificadas en tres categorías: métricas del producto, métricas del proceso y métricas del proyecto. Las métricas del producto describen características del producto tales como tamaño, complejidad, características de diseño, rendimiento y nivel de calidad. Las métricas del proceso pueden ser utilizadas para mejorar el proceso de desarrollo y mantenimiento del software. Algunos ejemplos son efectividad de la remoción de defectos durante el desarrollo y el tiempo de respuesta en el proceso de corrección de defectos. Las métricas del proyecto describen las características y ejecución del proyecto. Algunos ejemplos son: el número de desarrolladores de software, el comportamiento del personal durante el ciclo de vida de éste, el costo, el cronograma y la productividad. Algunas métricas pertenecen a múltiples categorías. Por ejemplo, las métricas de calidad del proceso de un proyecto, son ambas métricas del proceso y métricas del proyecto”.

En la gestión de proyectos, y más específicamente en la planeación de los mismos, surge la necesidad de generar una calendarización de las actividades a realizar durante el desarrollo del producto, siendo esto muy dificultoso en etapas tempranas de la elicitación de requisitos, por no poseer la información suficiente para medir el tamaño del producto. Éste trabajo se centra en la elaboración de un método que permita medir el tamaño del software a desarrollar en las primeras etapas de la

elicitación de requisitos, por ende requiere de una técnica de estimación basada en alguna métrica que especifique el tamaño del producto y de las entidades sobre las que se tomaran las mediciones que resultaran en dicha métrica.

Como se expuso en el párrafo anterior, nuestro interés está puesto en las etapas tempranas de la elicitación de requisitos, de ahí que se toma como punto de partida para el método propuesto, el LEL, que es un glosario que permite describir el lenguaje del dominio de la aplicación utilizando el lenguaje natural.

Su objetivo es describir ciertas palabras o frases peculiares a la aplicación cuya comprensión es vital para poder comprender el contexto de la misma. LEL está inspirado en una idea bien simple "entender el lenguaje del problema sin preocuparse por entender el problema". De esta forma, luego de comprender el lenguaje, el analista podrá escribir requerimientos utilizando como base el conocimiento adquirido a través del lenguaje capturado.

El LEL es un glosario en el cual se definen símbolos (términos o frases). A diferencia del diccionario tradicional que posee sólo un tipo de definición por término (puede haber muchas acepciones, sin embargo, todas son significados del término que se está describiendo). En el LEL cada símbolo se define a través de dos atributos: la noción e impactos. La noción describe la denotación, es decir, describe las características intrínsecas y substanciales del símbolo. Por su parte, los impactos describen la connotación, es decir, un valor secundario que adopta por asociación con un significado estricto.

El LEL centra su proceso constructivo en el momento en que los involucrados en la elicitación de requisitos recaban información del dominio del problema, registrándola en dicho LEL, el que es validado por el cliente, logrando un acuerdo en la definición de sus términos. De esta manera se tiene información muy valiosa acerca del dominio del problema y del producto que lo modelará.

Teniendo como punto de partida el LEL del dominio de una aplicación, éste trabajo propone derivar las entidades necesarias para construir la métrica que permita estimar el tamaño del producto, partiendo de la información contenida en dicho LEL. Acorde a esto, una métrica ampliamente utilizada en la estimación de proyectos, especialmente los orientados a objetos, es UCP, pero dicha métrica realiza sus mediciones sobre los Casos de Uso que componen un producto, para subsanar esta incompatibilidad entre la información que se dispone y la métrica escogida para realizar las estimaciones, se utilizan una transformación que permite derivar los UC desde el LEL, sumada a heurísticas que permiten inferir detalles de dichos UC, obteniendo de esta manera la información mínima e indispensable para utilizar dicho método de estimación.

UCP es un método de estimación de esfuerzo para proyectos de software a partir de sus Casos de Uso. Fue desarrollado por Gustav Karner, inspirándose en el método de punto de función de Albrecht. Subsecuentemente fue modificado por varias personas de Objective Systems, LLC, basándose en su experiencia con Objectory, una metodología creada por Jacobson para el desarrollo de aplicaciones con una metodología orientada a objetos.

El método se basa en cuatro variables principales para dar una estimación del esfuerzo, estas son: los actores involucrados en los Casos de Uso, las características de los Casos de Uso, el factor de complejidad técnica del proyecto y los factores de entorno del proyecto. A estas cuatro variables se les puede adicionar un factor de productividad para obtener el esfuerzo del proyecto en horas hombre.

De esta manera se logra calcular el esfuerzo estimado para el desarrollo de un proyecto de software en una etapa temprana de la elicitación de requisitos. Cabe destacar que, como toda métrica, ésta estimación debe recalcularse a medida que se adquiere conocimiento dentro del proceso de gestión del proyecto, para poder mejorarla y optimizar sus resultados, los que variaran dependiendo de la precisión con que se construyan los elementos de los que depende.

En los siguientes capítulos se brinda un marco conceptual de los elementos teóricos en los que se basa el método propuesto, para luego dedicar un capitulo al análisis de dicho método y a la herramienta que le da soporte, prosiguiendo con un capitulo puramente técnico en donde se analizan las tecnologías utilizadas en la construcción de la herramienta y sus interacciones a fin de cumplir con el objetivo de la misma, finalizando dicho capitulo con un detalle de los aspectos más interesantes de su construcción, una guía de instalación y un manual de usuario, a fin de invitar al lector a utilizarla. Por último se enuncian las conclusiones a las que condujo el presente trabajo.

Marco Teórico

Este capítulo brinda un marco teórico en el que se sustenta el método propuesto en el presente trabajo, su objetivo principal es proporcionar al lector los conocimientos básicos utilizados a lo largo de esta tesina abordando los elementos teóricos que permitieron su desarrollo.

El capítulo está dividido en cinco secciones, las cuales comienzan con una introducción a los Casos de Uso, prosiguiendo con la estimación basada en Use Case Points, para luego introducir los conocimientos referentes al Léxico Extendido del Lenguaje, después de lo cual se amalgaman estos conocimientos exponiendo un método que permite derivar UC desde el LEL del dominio de una aplicación, para finalizar el capítulo presentando un enfoque similar al adoptado en este trabajo, marcando sus diferencias y similitudes. Dando de esta manera una introducción a los elementos teóricos involucrados en esta tesina, a la vez que se introduce al lector en el estado del arte en lo que a los temas tratados concierne.

Use Cases

Un Caso de Uso (UC, del inglés Use Cases) [**15**] es una descripción de los pasos o las actividades que deberán realizarse para llevar a cabo algún proceso. Los personajes o entidades que participarán en un Caso de Uso se denominan actores. En el contexto de ingeniería del software, un UC es una secuencia de interacciones que se desarrollarán entre un sistema y sus actores en respuesta a un evento que inicia un actor principal sobre el propio sistema. (Se considera actor primario o principal al actor que inicia una comunicación con el sistema, en cambio sí el sistema es quien inicia dicha comunicación el actor será secundario).

Creados en 1986 por Ivar Jacobson, importante contribuyente al desarrollo de los modelos de UML y proceso unificado, se han realizado muchas mejoras al concepto que se estableció entonces, pero probablemente la más influyente y significativa, en términos de definición del término Caso de Uso, fue la de Alistair Cockburn en el libro Escribir casos de uso efectivos publicado en el año 2000. [**16**]

Un Caso de Uso es una técnica para la captura de requisitos potenciales de un nuevo sistema o una actualización de software. Cada UC proporciona uno o más escenarios que indican cómo debería interactuar el sistema con el usuario o con otro sistema para conseguir un objetivo específico. Normalmente para su redacción, se evita el empleo de jergas técnicas, prefiriendo en su lugar un lenguaje más cercano al usuario final. En ocasiones, se utiliza a usuarios sin experiencia junto a los analistas para el desarrollo de Casos de Uso.

Los diagramas de Casos de Uso, por ejemplo, sirven para visualizar de una manera gráfica y sencilla, la comunicación y el comportamiento de un sistema mediante su interacción con usuarios y/o sistemas. O lo que es igual, un diagrama que muestra la relación entre los actores y los Casos de Uso en un sistema. Una relación es una conexión entre los elementos del modelo, por ejemplo la especialización y la generalización son relaciones. Los diagramas de Casos de Uso se utilizan para ilustrar los requerimientos del sistema al mostrar cómo reacciona a eventos que se producen en su ámbito o en él mismo.

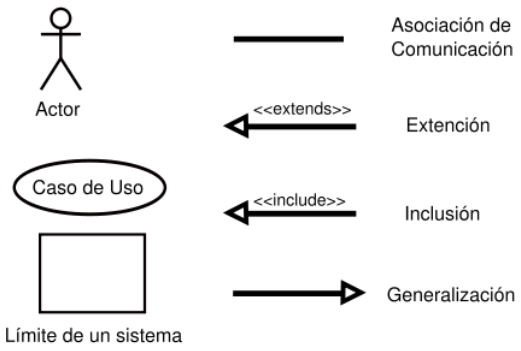

Ilustración 1 - Elementos de un Diagrama de Casos de Uso

Cada Caso de Uso se centra en describir cómo alcanzar una única meta o tarea de negocio. Desde una perspectiva tradicional de la ingeniería de software, un UC describe una característica del sistema. Para la mayoría de los proyectos de software, esto significa que quizás a veces es necesario especificar diez o centenares de Casos de Uso para definir completamente el sistema. El grado de formalidad de un proyecto particular de software y la etapa del proyecto influenciarán en el nivel del detalle requerido en cada Caso de Uso.

Los UC pretenden ser herramientas simples para describir el comportamiento del software o de los sistemas. Cada uno contiene una descripción textual de todas las maneras en que los actores previstos podrían trabajar con el software o el sistema. Los Casos de Uso no describen ninguna funcionalidad interna (oculta al exterior) del sistema, ni explican cómo se implementará; simplemente muestran los pasos que el actor sigue para realizar una tarea.

Un Caso de Uso debe describir una tarea del negocio que sirva a una meta de negocio, tener un nivel apropiado del detalle y ser bastante sencillo como para que un desarrollador lo elabore en un único intento.

Tipos de Relaciones en un Diagrama de Casos de Uso

Comunica (<<communicates>>): Relación (asociación) entre un actor y un UC que denota la participación del actor en dicho UC.

Usa (<<uses>>) (o <<include>> en la nueva versión de UML): Relación de dependencia entre dos UC que denota la inclusión del comportamiento de un UC en otro.

Extiende (<<extends>>): Relación de dependencia entre dos UC que denota que un UC es una especialización de otro.

Se utiliza una relación de tipo <<extends>> entre UC cuando nos encontramos con un Caso de Uso similar a otro pero que hace algo más que éste (variante). Por contra, utilizaremos una relación tipo <<uses>> cuando nos encontramos con una parte de comportamiento similar en dos UC y no queremos repetir la descripción de dicho comportamiento común.

En una relación <<extends>>, un actor que lleve a cabo el UC base puede realizar o no sus extensiones. Mientras, en una relación <<include>> el actor que realiza el UC base también realiza el UC incluido.

Por último en un diagrama de Casos de Uso, además de las relaciones entre UC y actor (asociaciones) y las dependencias entre UC (<<include>> y <<extends>>), pueden existir relaciones de herencia ya sea entre UC o entre actores.

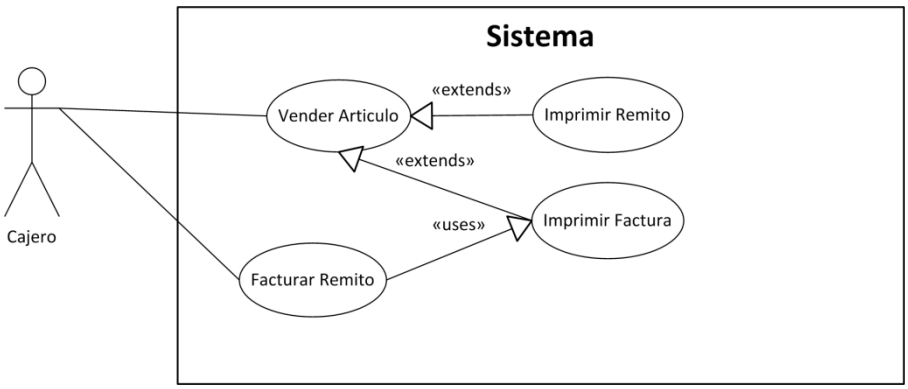

Ilustración 2 - Ejemplo Diagrama de Casos de Uso

Se denomina modelo de Casos de Uso a la combinación de UC y sus correspondientes diagramas. Los modelos de Casos de Uso se suelen acompañar por un glosario que describe la terminología utilizada. El glosario y el modelo de Casos de Uso son importantes puntos de partida para el desarrollo de los diagramas de clases.

Caso de Uso:	
Descripcion:	
Actores participantes:	
Pre-condiciones:	
Flujos	
Flujo Principal	
1	
2	
N	Finaliza el caso de uso.
Flujos Alternativos	
A1.	
Flujos de Excepcion	
E1.	
Post-condiciones:	

Ilustración 3 - Ejemplo Descripción de un Caso de Uso

Ventajas

La técnica de Caso de Uso tiene éxito en sistemas interactivos, ya que expresa la intención que tiene el actor (su usuario) al hacer uso del sistema.

Como técnica de extracción de requerimiento permite que el analista se centre en las necesidades del usuario, qué espera éste lograr al utilizar el sistema, evitando que la gente especializada en informática dirija la funcionalidad del nuevo sistema basándose solamente en criterios tecnológicos.

Limitaciones

Los Casos de Uso pueden ser útiles para establecer requisitos de comportamiento, pero no establecen completamente los requisitos funcionales ni permiten determinar los requisitos no funcionales.

Use Case Points

Es un método de estimación de esfuerzo para proyectos de software a partir de sus Casos de Uso. Fue desarrollado por Gustav Karner en 1993 [**17**], inspirandose en el método de punto de función de Albrecht [**18**]. Subsecuentemente fue modificado por varias personas de Objective Systems, LLC, basándose en su experiencia con Objectory, una metodología creada por Jacobson para el desarrollo de aplicaciones con una metodología orientada a objetos [**19**].

El método se basa en cuatro variables principales para dar una estimación del esfuerzo, estas son: los actores involucrados en los Casos de Uso, las características de los Casos de Uso, el factor de complejidad técnica del proyecto y los factores de entorno del proyecto. A estas cuatro variables se les puede adicionar un factor de productividad para obtener el esfuerzo del proyecto en horas hombre.

Actores

Los actores involucrados en los Casos de Uso se clasifican de acuerdo a la forma en la que interactúan con el sistema, separándolos en tres categorías, los actores simples son aquellos que representan a otros sistemas que interactúan con el sistema a desarrollar por medio de una interfaz de programación (API, del inglés Application Programming Interface) y se les asigna un peso de 1 (uno), los actores medios representan a otros sistemas que interactúan con el sistema a desarrollar a través de un protocolo o a una persona que interactúan con el sistema a través de una interfaz en modo texto y se les asigna un peso de 2 (dos), por último los actores complejos representan a personas que interactúan con el sistema mediante una interfaz gráfica (GUI, del inglés Graphical User Interface) y se les asigna un peso de 3 (tres). Una vez categorizados los actos se obtiene el peso de actores desajustado (UAW, del inglés Unadjusted Actor Weight) totalizando los actores en cada categoría, multiplicando este número por el peso de dicha categoría y finalmente sumando los productos.

$$UAW = A_s + A_m * 2 + A_c * 3$$

Siendo:

$$A_s = Cantidad\ de\ actores\ simples$$

$$A_m = Cantidad\ de\ actores\ medios$$

$$A_c = Cantidad\ de\ actores\ complejos$$

Tipo de actor	Descripción	Peso
Simple	Otro sistema que interactúa con el sistema a desarrollar mediante una interfaz de programación (API).	1
Medio	Otro sistema interactuando a través de un protocolo (ej. TCP/IP) o una persona interactuando a través de una interfaz en modo texto	2
Complejo	Una persona que interactúa con el sistema mediante una interfaz gráfica (GUI).	3

Tabla 1 - Resumen de Actores

Casos de Uso

Según sus características los Casos de Uso son categorizados en tres categorías, Los Casos de Uso simples son aquellos que poseen interfaz de usuario simple, interactúan con una sola entidad de base de datos, su escenario de éxito consiste en tres o menos pasos y su implementación requiere de cinco o menos clases, a estos se les asigna un peso de 5 (cinco). Los Casos de Uso medios requieren de un mayor diseño de interfaz de usuario, interactúan con dos o más entidades de base de datos, sus escenarios de éxito comprenden de cuatro a siete pasos i su implementación involucra de cinco a diez clases, a los Casos de Uso de esta categoría les corresponde un peso de 10 (diez). Por último los Casos de Uso complejos requieren de interfaces de usuario complejas o procesamiento complejo, interactúan con tres o más entidades de base de datos, su escenario de éxito conlleva más de siete pasos y su implementación involucra más de diez clases, les corresponde un peso de 15 (quince). Cabe aclara que para que un Caso de Uso entre en una determinada categoría no debe superar ninguna de las restricciones de la misma.). Una vez categorizados los Casos de Uso se obtiene el peso de Casos de Uso desajustado (UUCW, del inglés Unadjusted Use Case Weight) totalizando los actores en cada categoría, multiplicando este número por el peso de dicha categoría y finalmente sumando los productos.

$$UUCW = UC_s * 5 + UC_m * 10 + UC_c * 15$$

Siendo:

$$UC_s = Cantidad\ de\ casos\ de\ uso\ simples$$

$$UC_m = Cantidad\ de\ casos\ de\ uso\ medios$$

$$UC_c = Cantidad\ de\ casos\ de\ uso\ complejos$$

Tipo de CU	Descripción	Peso
Simple	Interfaz de usuario simple, interactúan con una sola entidad de base de datos, su escenario de éxito consiste en tres o menos pasos y su implementación requiere de cinco o menos clases.	5
Medio	Mayor diseño de interfaz de usuario, interactúan con dos o más entidades de base de datos, sus escenarios de éxito comprenden de cuatro a siete pasos i su implementación involucra de cinco a diez clases.	10
Complejo	Interfaz de usuario compleja o procesamiento complejo, interactúan con tres o más entidades de base de datos, su escenario de éxito conlleva más de siete pasos y su implementación involucra más de diez clases.	15

Tabla 2 - Resumen de Casos de Uso

Factor Total de Complejidad Técnica

Para calcular el factor total de complejidad técnica, se definen trece factores estándar de complejidad técnica que representan el impacto de determinadas situaciones en la productividad de un proyecto, a cada uno se le asigna un peso dependiendo de su impacto relativo al mismo. Para cada proyecto el equipo de desarrollo deberá evaluar estos factores y asignarles un valor entre cero y cinco que represente la complejidad percibida para dicho factor, asignar un valor de cero lo vuelve irrelevante, una asignación de cinco lo define como fuerte. Una vez definida la complejidad percibida de cada factor, se la multiplica por el peso del factor y se obtiene el factor calculado, los que luego se suman y se ajustan con dos constantes para obtener el factor total de complejidad técnica.

$$TCF = C_1 + C_2 \sum_{i=1}^{13} W_i * F_i$$

Siendo:

$$C_1 = 0.6$$

$$C_2 = .01$$

$$W_i = El\ Peso\ del\ factor\ i$$

$$F_i = El\ valor\ de\ complejidad\ percibida$$

Factor técnico	Descripción	Peso
T1	Sistema distribuido.	2
T2	Objetivos de performance o tiempo de respuesta.	1
T3	Eficiencia del usuario final.	1
T4	Procesamiento interno complejo.	1
T5	El código debe ser reutilizable.	1
T6	Facilidad de instalación.	0.5
T7	Facilidad de uso.	0.5
T8	Portabilidad.	2
T9	Facilidad de cambio.	1
T10	Concurrencia.	1
T11	Incluye objetivos especiales de seguridad.	1
T12	Provee acceso directo a terceras partes.	1
T13	Se requiere facilidades especiales de entrenamiento a usuario.	1

Tabla 3 - Resumen Factores Técnicos

Factor Total de Complejidad Ambiental

Los factores ambientales indican la influencia del equipo involucrado en el desarrollo del proyecto, para esto se definen ocho factores ambientales a los que se les asigna un peso dependiendo de su impacto en el proyecto. Para cada proyecto equipo de desarrollo evalúa estos factores y les asigna un valor que representa el impacto percibido que tendrán en dicho proyecto. Asignar un valor de uno representa un fuerte impacto negativo para el proyecto, un valor de cinco representa un fuerte impacto positivo y un valor de cero lo vuelve irrelevante. Una vez definida la complejidad percibida de cada factor, se la multiplica por el peso del factor y se obtiene el factor calculado, los que luego se suman y se ajustan con dos constantes para obtener el factor total de complejidad ambiental.

$$ECF = C_1 + C_2 \sum_{i=1}^{13} W_i * F_i$$

Siendo:

$$C_1 = 0.6$$

$$C_2 = .01$$

$$W_i = El\ Peso\ del\ factor\ i$$

$$F_i = El\ valor\ de\ complejidad\ percibida$$

Factor	Descripción	Peso
E1	Familiaridad con el modelo de proyecto utilizado.	1.5
E2	Experiencia en la aplicación.	0.5
E3	Experiencia en orientación a objetos.	1
E4	Capacidad del analista líder.	0.5
E5	Motivación.	1
E6	Estabilidad de los requerimientos	2
E7	Personal part-time	-1
E8	Dificultad del lenguaje de programación	-1

Tabla 4 - Resumen Factores Ambientales

Calculo de Puntos Caso de Uso

Para realizar el cálculo de Puntos Caso de Uso (UCP, del inglés Use Case Points) se emplea la siguiente formula:

$$UCP = UUCP * TCF * ECF$$

En donde

$$UUCP = UAW + UUCW$$

De esta manera se obtiene el esfuerzo necesario estimado para desarrollar el proyecto expresado en Puntos Caso de Uso, si lo que se desea es obtener la estimación expresada en horas hombre, se debe multiplicar los UCP por un factor de productividad (PF, del inglés Productivity Factor) que expresa la relación entre UCP y horas hombres necesarias para completarlo. Este factor puede venir dado por datos históricos recogidos de proyectos anteriores o en el caso de no contar con estos datos se sugiere utilizar un valor entre quince y treinta dependiendo de la experiencia del equipo de desarrollo.

$$Estimacion\ Total = UCP * PF$$

Léxico Extendido del Lenguaje

El Léxico Extendido del Lenguaje (LEL, del inglés Lenguage Extended Lexicon) [**20**]es un glosario que permite describir el lenguaje del dominio de la aplicación utilizando el lenguaje natural.

Su objetivo es describir ciertas palabras o frases peculiares a la aplicación cuya comprensión es vital para poder comprender el contexto de la misma. LEL está inspirado en una idea bien simple "entender el lenguaje del problema sin preocuparse por entender el problema". De esta forma, luego de comprender el lenguaje, el analista podrá escribir requerimientos utilizando como base el conocimiento adquirido a través del lenguaje capturado. [**21**]

El LEL es un glosario en el cual se definen símbolos (términos o frases). A diferencia del diccionario tradicional que posee sólo un tipo de definición por término (puede haber muchas acepciones, sin embargo, todas son significados del término que se está describiendo). En el LEL cada símbolo se define a través de dos atributos: la noción (notion) y los impactos (behavioural responses).La noción describe la denotación, es decir, describe las características intrínsecas y substanciales del símbolo. Por su parte, los impactos describen la connotación, es decir, un valor secundario que adopta por asociación con un significado estricto.

Existen dos principios que se deben seguir al describir símbolos: el principio de circularidad (también llamado principio de cierre o clausura) y el principio de vocabulario mínimo. El principio de circularidad establece que durante la descripción de los símbolos se debe maximizar el uso de otros símbolos descriptos en el LEL. Por su parte, el principio de vocabulario mínimo complementa el principio de circularidad y establece que en las descripciones se debe minimizar el uso de símbolos externos al LEL y los que se utilicen deben tener una definición clara, unívoca y no ambigua.

Ambos principios son vitalmente importantes para obtener un LEL autocontenido y altamente conectado. Estas características redundan en beneficios para comprender el lenguaje de la aplicación, y también hacen que el LEL pueda ser visto como un grafo, el cual, al contener nodos con información textual determina que el mismo sea un hipertexto.

Los símbolos se deben categorizar en una de cuatro categorías básicas con el fin de especializar la descripción de los atributos. En principio hay 3 ontologías básicas que permiten modelar el mundo. Ellas son: entidades, actividades y afirmaciones. Con estos 3 elementos se puede describir al mundo, puesto que permiten modelar las cosas, las actividades con las que interactúan las cosas y finalmente las afirmaciones o verdades que las cosas o actividades poseen o persiguen.

Para el LEL se determinan cuatro categorías básicas que son una extensión de las tres ontologías. Las cuatro categorías básicas son: sujeto, objeto, verbo y estado. Tanto los sujetos como los objetos son una especialización de las entidades. Luego, las actividades se corresponden con los verbos. Y finalmente las afirmaciones se corresponden con estados.

Nombre:	
Sinónimos:	
Tipo:	
Noción:	
Impactos:	

Ilustración 4 - Template Símbolo del LEL

Los sujetos se corresponden con elementos activos dentro del contexto de la aplicación, mientras que los objetos se corresponden con elementos pasivos. Por su parte, los verbos son las acciones que realizan los sujetos utilizando los objetos. Finalmente, los estados representan las situaciones en las que se pueden encontrar los sujetos o los objetos antes y después de realizar las acciones.

Categoría	Características	Noción	Impactos
Sujeto	Elementos activos que realizan acciones	Características o condiciones que los sujetos satisfacen	Acciones que el sujeto realiza
Objeto	Elementos pasivos con los cuales los sujetos realizan acciones	Características o atributos que los objetos poseen	Acciones que son realizadas con los objetos
Verbo	Acciones que los sujetos realizan con los objetos	Objetivo que el verbo persigue	Pasos necesarios para completar la acción
Estado	Situaciones en las cuales se pueden encontrar los sujetos y los objetos	Situación que representa el estado	Acciones que deben realizarse para cambiar a otro estado

Tabla 5 - Resumen Símbolos del LEL

El LEL no brinda una descripción de requerimientos, sino que sólo permite describir el lenguaje del dominio, a pesar de que hay trabajos que muestran cómo identificar requerimientos a partir del LEL.

La utilidad del LEL como modelo ha sido observada en los trabajos de Breitman [**22**] y Gruber [**23**].Breitman describe un proceso para estudiar el LEL con el fin de producir una ontología, por lo cual, ella estudia el lenguaje del contexto de la aplicación y a partir del mismo construye una ontología. Grubber por su parte define ontología como “una especificación explícita y formal de una conceptualización compartida”. Si esta conceptualización compartida mencionada por Gruber es una conceptualización compartida sobre el mundo real, lo será sobre la aplicación con la cual se está trabajando y por lo tanto puede ser considerada como modelo de la misma. Por lo tanto, con el LEL se captura el lenguaje del contexto de la aplicación y a partir de él se construye una ontología la cual puede ser considerada como modelo de la aplicación, lo que en definitiva nos lleva a que el mismo LEL puede ser considerado como un modelo de la aplicación. Y esto se indica explícitamente en [**22**] ya que se menciona que “la filosofía subyacente al LEL cae en la categoría de contextualización, de acuerdo a la cual, las particularidades de un contexto de uso de aplicación deben ser comprendidas en detalle antes de que se puedan producir los requerimientos”.

En principio LEL es una herramienta muy conveniente para expertos sin habilidades técnicas, sin embargo, las personas con tales habilidades obtendrán un mayor beneficio con su uso. La conveniencia del LEL proviene de 3 características significativas: es fácil de aprender, es fácil de usar y posee buena expresividad. Hay experiencias en dominios complejos que validan la conveniencia del LEL. Gil et al. [**6**] indican que: “la experiencia de construir un LEL de una aplicación completamente desconocida para los ingenieros de requerimientos y con un lenguaje altamente complejo, puede ser considerada exitosa, desde el momento en que los usuarios fueron los que notaron que los ingenieros de requerimientos habían desarrollado un gran conocimiento sobre la aplicación”. Por su

parte, Cysneiros et al. [**5**] Indican que: “el uso del LEL fue muy bien aceptado y comprendido por los interesados (stakeholders). Dado que los interesados (stakeholders) no eran expertos en los dominios tan complejos y específicos en los que trabajaron, los autores creen que el LEL pude ser adecuado para utilizarse en muchos dominios”.

La calidad del modelo depende del esfuerzo invertido en la construcción como así también en la capacidad de representación. En el párrafo anterior se mostró que hay experiencias que justifican la efectividad del LEL como instrumento para capturar conocimiento y lo más importante aún, es que es una herramienta adecuada para la comunicación del conocimiento capturado. Durante el desarrollo participan muchos actores que poseen diferentes capacidades y habilidades, y dado que el LEL utiliza lenguaje natural, es una herramienta apropiada para expertos, ingenieros de requerimientos y desarrolladores. Y esto es de gran relevancia al contribuir positivamente en los esfuerzos necesarios para realizar validación de estos modelos. Kaplan et al. [**24**] han observado que la simplicidad de construcción y organización provistas por el LEL es un factor clave en la validación con los usuarios.

El proceso de construcción de LEL [**20**] [**22**] se puede resumir en las siguientes etapas: (i) identificación de los símbolos, (ii) categorización de los símbolos identificados, (iii) descripción de los símbolos en función de la categoría definida, (iv) identificación de sinónimos a partir de las descripciones, (v) control de las definiciones y especialización de las categorías y finalmente (vi) validación con los interesados (stakeholders).

La identificación de símbolos comienza por determinar las fuentes a partir de las cuales se va a estudiar el lenguaje de la aplicación. Básicamente es posible realizarlo a partir de texto escrito como podría ser documentación relacionada con un sistema o también se podría realizar a partir de interacciones con los interesados (stakeholders) como podrían ser entrevistas. Dado que el objetivo del LEL es capturar el lenguaje, es necesario identificar los símbolos más representativos para definirlos en el LEL. La lengua escrita y la oral son muy distintas. Al expresarse en forma escrita se suele ser más preciso y se suelen utilizar muchos sinónimos con el fin de mejorar la redacción. De todas formas, los términos específicos característicos de un dominio se suelen utilizar sin recurrir a sinónimos, por lo cual los símbolos candidatos a identificar son aquellos que más se repiten, los que se encuentran enfatizados (con negrita, cursiva o subrayaros), aquellos que se ubican en los títulos, descripción de figuras o tablas. En la expresión oral se suele ser más desorganizado y se suele repetir términos. En este caso, aquellos términos más nombrados y en los cuales se enfatiza cambiando el tono de voz por ejemplo, son aquellos a los cuales hay que prestarle atención para considerarlos.

Una vez identificados los símbolos y previo a la descripción de los mismos es necesario categorizarlos, es decir, se debe determinar si el símbolo es sujeto, objeto, verbo o estado. Es necesario esta categorización, puesto a partir de ella se determina de qué forma (es decir, con qué información) se debe caracterizar (describir) cada uno de los símbolos. Esta categorización no es definitiva, sino que es una categorización preliminar y luego en la descripción puede surgir que la categoría determinada no era la correcta.

La descripción de los símbolos implica escribir la noción y los impactos de cada símbolo, de forma tal que los atributos se ajusten a la plantilla (template) establecido según la categoría a la que pertenece el símbolo. Es importante tener presente los principios de circularidad y de vocabulario mínimo durante esta fase, para cumplir con los mismos. Por otra parte, puede suceder que la descripción de los símbolos no surja del conocimiento que el analista recopiló, sino que necesita recurrir a las fuentes para completar la descripción. En este caso, podría suceder que aparezcan nuevos términos que necesiten ser definidos.

Una vez descriptos los símbolos es posible identificar sinónimos. Podría suceder que al identificar los símbolos no se haya reparado en que dos símbolos eran sinónimos por el vago conocimiento que se poseía sobre el dominio. Sin embargo, luego de describirlos es el momento para verificar si dos símbolos independientes poseen descripciones similares. En ese caso, se deben unir ambas descripciones en una sola y ambos términos se utilizan como sinónimos que identifican al concepto.

La identificación de sinónimos es en cierta forma un control del LEL por parte del analista, puesto que consiste en una revisión de los símbolos para factorizarlos. Sin embargo, no es el único control que es posible hacer. Tal vez se descubre que un concepto es muy amplio, por lo cual, se podría definir un concepto genérico y todas las especializaciones que fueran necesarias. También podría suceder que se utilizan términos para describir otros, y estos términos que se utilizan merecen ser definidos.

Además, podría surgir la necesidad de especializar la categorización básica de símbolos. Por ejemplo, podría suceder que el dominio con el cual se trabaja merece describir una gran cantidad de sujetos y fuera necesario determinar distintas especializaciones de sujetos, porque habría que describir símbolos que se corresponden con estas subcategorías. Esta necesidad debe surgir en el control y deben crearse las subcategorías con la plantilla (template) que deben cumplir los símbolos de estas nuevas subcategorías y también debe adecuarse la descripción de los símbolos con las nuevas subcategorías.

Finalmente, la última etapa en la construcción del LEL es la validación del LEL obtenido. Claramente es la fase más costosa y sensible, y en cierta forma es utópica, dada la necesidad de validarlo con las mismas fuentes que permitieron su construcción y que no siempre están disponibles. Sin embargo, siempre es posible solicitar a algún interesado (stakeholder) que realice alguna revisión exploratoria con el fin de validar el producto obtenido.

Buenas prácticas en la escritura de un Léxico Extendido del Lenguaje

Si bien todas las fases de construcción del LEL requieren un esfuerzo importante, el hecho de describir en símbolos el conocimiento adquirido es una etapa con cierta criticidad. Para ello existe una herramienta que ofrece ciertas bondades y facilidades: C&L [**25**] [**26**].

A fin de homogeneizar la escritura del LEL y facilitar su comprensión y procesamiento es conveniente seguir los lineamientos propuestos por Antonelli et. al. [**27**] en el documento “Buenas prácticas en la especificación del dominio de una aplicación”, donde se proponen diez guías a seguir, dichas guías están basadas en la redacción textual de requerimientos como en el diseño orientado objetos, aportando la cohesión necesaria para interpretar al LEL desde la perspectiva de los humanos así como también facilitando su procesamiento. A continuación se hace una breve reseña de los lineamientos propuestos

Guía 1: Información a incluir en la noción y en los impactos

La descripción de la noción y los impactos se debe ajustar a la categoría del símbolo que se debe describir. Para los sujetos, la noción debe describir las características o condiciones que debe satisfacer para poder considerarse como tal. Por su parte, los impactos, deben describir las acciones que realizan. Para los objetos, la noción debe describir los atributos o características constitutivas del objeto, mientras que los impactos deben describir las acciones que se realizan sobre ellos. Para los verbos, la noción debe describir el objetivo o fin que persiguen, mientras que los impactos deben describir los pasos necesarios para cumplir con la acción. Finalmente, la noción de los estados debe

describir la situación que representa, mientras que los impactos deben describir las acciones que se pueden realizar a partir de ese estado y el nuevo estado que se puede acceder.

Esta guía se sustenta en que la descripción de cada categoría está basada en el análisis y diseño orientado a objetos propuesto por Wirfs-Brock [**28**]. Las categorías sujeto y objeto, se corresponden con objetos (clases) es por ello que la noción caracteriza a estos elementos, mientras que los impactos describen las acciones vinculadas con cada uno de ellos. Por su parte, los verbos se corresponden con el comportamiento de los objetos (métodos), es por ello que la noción describe conceptualmente la acción, mientras que los impactos detallan la misma. Finalmente, la categoría estado, guarda estrecha relación con el patrón estado [**29**], por lo cual, su descripción tiene como finalidad describir una máquina de estados

Guía 2: Formato que deben cumplir cada una de las expresiones de los impactos de los sujetos, objetos y verbos

Las acciones descriptas en los impactos de sujetos, objetos y verbos deben cumplir la estructura: sujeto + verbo + objeto. Es importante destacar que el objeto al final de la oración queda condicionado por el verbo. Es decir, si el verbo es transitivo, se debe indicar un objeto. Caso contrario, no es necesario.

Esta guía se sustenta en que los impactos describen acciones, que pueden considerarse como funciones o requerimientos de un sistema. El estándar IEEE 830 [**30**] determina que los requerimientos deben describirse de la forma: “El sistema debe acción”. Luego, Kovitz [**31**] especializa esta descripción, incorporando el rol: “El rol debe acción”. La guía presentada incorpora además el objeto.

Guía 3: Formato que deben cumplir cada una de las expresiones de los impactos de los estados

Las acciones descriptas en los impactos de los estados deben cumplir la estructura: sujeto + verbo + objeto + nuevo estado. Es importante destacar que el objeto queda condicionado por el verbo, ya que si el verbo no es transitivo el objeto no se debe indicar.

Esta guía se sustenta en que Chelimsky [**32**] propone especificar comportamiento en desarrollo ágil con el formato: given (situación inicial) when (acción) then (situación final). Básica mente describe una máquina de estados con la transición necesaria para pasar de un estado a otro. Para la regla propuesta, el símbolo estado que se está describiendo es la situación inicial, luego, las transiciones (acciones) se describen con: sujeto + verbo + objeto, mientras que el estado final con: nuevo estado.

Guía 4: Simplicidad en los impactos

Los impactos deben ajustarse al formato indicado en las guías 2 y 3, y si fuera necesario agregar información adicional, debe definirse alguno de los símbolos involucrados y agregar la información adicional en la descripción de ese símbolo.

Esta guía se sustenta en que el estándar IEEE 830 [**30**] establece que una especificación de requerimientos debe ser modificable, y para ello la modularidad es una herramienta indispensable. La misma estrategia se aplica al LEL.

Guía 5: Auto referencias explícitas en los impactos de los sujetos y objetos

En la descripción de los impactos de un sujeto y de un objeto, cuando se debe referenciar al sujeto y objeto, se lo debe hacer explícitamente y no utilizar un pronombre.

Esta guía se sustenta en que la ambigüedad es uno de los problemas que se intentan evitar en la descripción de requerimientos [**31**] [**33**], es por ello que se evita utilizar el pronombre.

Guía 6: No utilizar auto referencias en los impactos de verbos o en la noción de cualquier tipo de símbolo

No se debe describir la noción de ninguna categoría de símbolos usando el mismo símbolo que se está describiendo. Tampoco se deben describir los impactos de los verbos usando el mismo verbo.

Esta guía se sustenta en que no es correcto definir el símbolo en términos de sí mismo o tampoco es posible descomponer la tarea en términos de sí misma.

Guía 7: No se deben utilizar frases débiles

No se deben utilizar frases débiles tales como: puede, podría, etc.

Esta guía se sustenta en que las frases débiles dan a lugar cierta subjetiva sobre la descripción, por lo cual, no deben ser utilizadas [**34**]

Guía 8: Relación "es un" ("is a")

Dados dos símbolos en donde uno es una especialización del otro, no se debe repetir información en ambos. En la noción del más específico se debe colocar la expresión "es un" + símbolo genérico.

Esta guía se sustenta en que esta relación es utilizada en representación del conocimiento, modelos conceptuales y diseño y programación orientada a objetos [**35**]. En particular, en programación orientada a objetos existe el principio de sustitución denominado principio de sustitución Liskov [**36**].

Guía 9: Relación "desempeña el rol"

Dados símbolos sujetos, en donde hay características que no son propias de los sujetos, sino del rol que desempeñan, se debe definir un símbolo con las características de los sujetos y la descripción propia del rol. Luego, en los sujetos que desempeñan ese rol, se deben enriquecer la noción con la expresión "desempeña el rol" + rol.

Esta guía se sustenta en que en diseño orientado a objetos existe el patrón the role object pattern que permite adaptar un objeto a diferentes necesidades de sus objetos clientes adosándole distintos roles [**37**].

Guía 10: Relación "tiene un"

Dados dos símbolos de categoría objeto, en donde uno es una parte del otro, en la noción del objeto contenedor se debe indicar la expresión "tiene un" + objeto contenido.

Esta guía se sustenta en que esta relación es utilizada en diseño de base de datos como así también en, modelos conceptuales, y diseño y programación orientada a objetos [**35**].

Transformación de Léxico Extendido del Lenguaje a Use Cases

La transformación propuesta por Antonelli et. al. [**38**] Se basa en que los Casos de Uso representan interacciones con la aplicación y dado que los verbos del LEL representan acciones dentro del alcance de la aplicación, entonces cada verbo debería derivarse en un Caso de Uso. El identificador del Caso de Uso estará dado por el nombre del verbo, el objetivo del UC estará dado por la noción del verbo (los verbos tienen por noción el objetivo de la acción). El escenario de éxito principal del UC está dado por los impactos del verbo (Los impactos de un verbo describen los pasos necesarios para alcanzar su objetivo). Los actores del Caso de Uso se obtiene a partir de los sujetos definidos en el LEL, ya que estos están relacionados con los verbos de manera natural dado que sus impactos son las acciones que dichos sujetos llevan a cabo, de esta manera se propone como actor principal al sujeto que lleva a cabo la acción establecida en el verbo. Para obtener las precondiciones y postcondiciones del CU se utilizan los estados definidos en el LEL, identificando los que están relacionados al verbo que describe el Caso de Uso. En algunos casos puede darse que no existan estados relacionados con el verbo quedando las precondiciones y/o postcondiciones sin definir. En lenguaje ATL [**39**] la transformación estaría dada por:

```
helper LEL def: StateUsingVerbAsTransition(): Symbol =
(self.referencedInBehaviouralResponsesFrom()->select (x|x.isState())-> first ())
rule LEL2UseCase {
from s : Symbol (s.isVerb())
to u : UseCase (
    u.useCase <- s.name
    u.goalInContext <- s.notion
    u.preconditions <- (StateUsingVerbAsTransition).name
    u.successEndCondition <- (StateUsingVerbAsTransition)
    behaviouralResponses() ->
select (x|x.isState()) -> first()).name
    u.primaryActor <- s.referencedInBehaviouralResponsesFrom() ->
    select
     (x|x.isSubject()) -> first()
u.mainSuccessScenario <- s.behaviouralResponses }
```

Ilustración 5 - Transformación de LEL a Use Cases Expresada en ATL

Viéndola en forma gráfica la transformación definida anteriormente responde al siguiente diagrama:

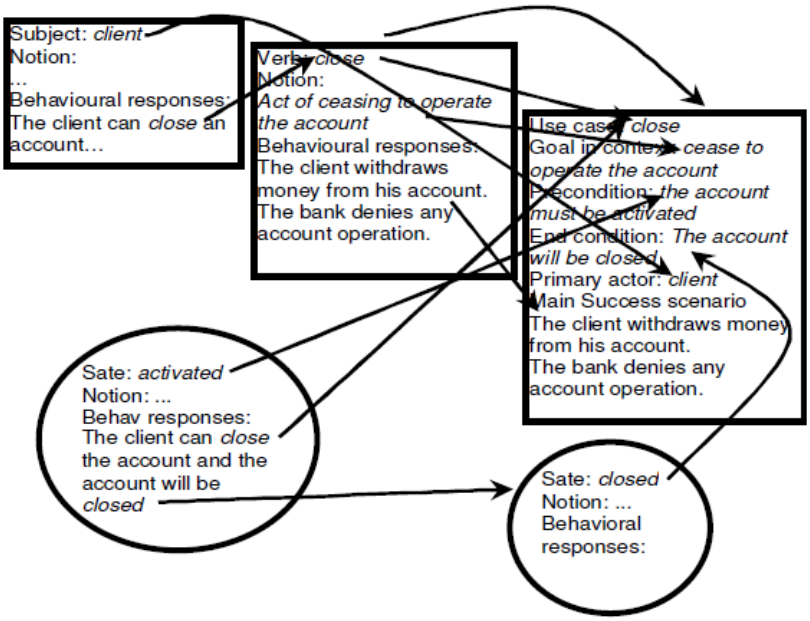

Ilustración 6 - Grafica de la Transformación de LEL a Use Cases

Mediante esta técnica obtenemos un Caso de Uso simple compuesto por los mínimos elementos necesarios para expresarlo, a saber el CU queda definido con su nombre, su objetivo, sus pre y post condiciones y su flujo normal.

LEL Points

LEL Points es una técnica propuesta por Antonelli et. al. en [**40**], que plantea como obtener una medida similar a los puntos de Casos de Uso desajustado (UUCP) partiendo del LEL de una aplicación, se basa en que el LEL captura el lenguaje de la aplicación y que algunos de las expresiones de este lenguaje serán verbos que estarán relacionados con los Casos de Uso que describirán la aplicación. Partiendo de estas premisas propone que el tamaño del LEL llamado LELP puede ser utilizado en la técnica de estimación de Puntos Caso de Uso en reemplazo del UUCP.

LEL Point parte de la premisa de que el LEL utilizado como entrada describe una aplicación específica, ya que un LEL puede no describir el lenguaje del dominio de una aplicación y dar lugar a múltiples aplicaciones diferentes, para evitar que esto pase el LEL debe estar definido para describir específicamente el dominio de la aplicación a estimar con la técnica propuesta.

Utilizando el mismo principio que UCP, LELP propone calcular LEL points desajustados (ULELP, del inglés Unadjusted Language Extended Lexicon Points) que se compone de la suma del peso desajustado de verbos (UVW, del inglés Unadjusted Verb Weight) y del peso desajustado de los sujetos (USW, del inglés Unadjusted Subject Weight). El enfoque propuesto en LELP no considera medidas equivalentes al factor de productividad, a los factores de complejidad técnica ni a los factores de complejidad ambiental ya que se basa en la analogía con el cálculo del peso desajustado de los Casos de Uso, así mismo indica que en algunos casos es posible utilizar lo indicado por UCP para el cálculo de dichos factores. De esta manera

$$ULELP = UVW + USW$$

Basándose en que el cálculo de UAW en UCP, que consiste en analizar los actores que utilizaran la aplicación y que de acuerdo con [**38**] los actores serán representados con símbolos de tipo sujeto, el USW se obtiene categorizando estos símbolos según el siguiente criterio, un sujeto será simple si representa a otro sistema interactuando mediante una API y se le asignara un peso de uno (1), será un sujeto promedio si representa a otro sistema interactuando mediante un protocolo o a una persona interactuando mediante una interfaz basada en texto, en este caso se le asignara un peso de dos (2), por ultimo serán sujetos complejos los que representen a personas interactuando mediante una GUI y se les asignara un peso de tres (3).

Tipo de sujeto	Descripción	Peso
Simple	Otro sistema a través de una API	1
Medio	otro sistema interactuando mediante un protocolo o a una persona interactuando mediante una interfaz basada en texto	2
Complejo	Personas interactuando mediante una GUI	3

Tabla 6 - Resumen de USW

Para obtener el USW se suman los pesos asignados a los términos de tipo sujeto

$$USW = \sum W_{si}$$

Siendo W_{si} el peso asignado al sujeto i.

Siguiendo con el paralelismo planteado, para el cálculo de UVW se toma como referencia el cálculo de UUCW definido en UCP y de esta manera los símbolos de tipo verbo son categorizados basándose en el número de pasos que tienen sus impactos, el número de símbolos de tipo objeto y de tipo sujeto que son referenciados en sus impactos y la complejidad de la UI que requerirá la implementación de la funcionalidad descripta por dicho verbo. Esto se debe a que basándose en [**38**], los pasos definidos en los impactos hacen referencia a los pasos descriptos en el escenario principal de un UC, y según [**28**] los términos de tipo sujeto son candidatos a ser representados por clases y los de tipo objeto por entidades de base de datos. Por último se hace referencia a la complejidad de la UI que se requerirá en la implementación de la funcionalidad descripta. Una vez analizados estos aspectos se categorizan los verbos en simples, a los que se les asigna un peso de cinco (5), medios, a los que les corresponde un peso de diez (10) o complejos a los que se les asigna un peso de quince (15). La siguiente tabla resume lo expuesto.

Tipo de Verbo	UI	Objetos	Pasos	Sujetos	Peso
Simple	Simple	1	<=3	>=4	5
Medio	Media	2	4 - 7	5 - 10	10
Complejo	Compleja	>2	>7	>10	15

Tabla 7 - Resumen UVW

Para obtener el UVW se suman los pesos asignados a los términos de tipo verbo

$$UVW = \sum W_{vi}$$

Siendo W_{vi} el peso asignado al verbo i.

Por último se obtiene ULELP con la siguiente formula

$$ULELP = UVW + USW$$

Palabras Finales

En este capítulo se abordaron los conceptos de Caso de Uso, Puntos Caso de Uso, Léxico extendido del Lenguaje, transformación de Léxico extendido del Lenguaje en Casos de Uso y LEL Points, La elección y orden de los temas están intrínsecamente ligados al método que se describe en el siguiente capítulo.

Para comenzar, se introdujo el concepto de Caso de Uso que actúa como una abstracción de la representación de un requerimiento, está identificado por un nombre, tiene un objetivo principal y un conjunto de actores que lo llevan a cabo. Su ejecución se guía por un flujo, que inicia con un conjunto de precondiciones y finaliza produciendo un conjunto de postcondiciones.

Luego de esto se presentó Use Case Points, que es un método de estimación del tamaño de una aplicación que permite estimar el esfuerzo necesario para desarrollar la aplicación basándose en la información provista por los Casos de Uso, sumado a los factores ambientales que describen al equipo que llevará a cabo dicho desarrollo, y la complejidad técnica que tendrá la aplicación.

El tercer concepto en aparecer en el presente capitulo es el de Léxico Extendido del Lenguaje, que representa una forma de capturar información del dominio de un problema sin entender el problema como un todo. Dado que el LEL actúa como un glosario en donde se definen los conceptos que aplican al problema que se está analizando, es una gran herramienta para la unificación de conocimiento entre el cliente y el equipo de elicitación de requisitos, y más aún, es una construcción que, de llevarse a cabo como práctica habitual en la elicitación de requisitos, aglutina una gran cantidad de conocimiento en etapas tempranas de dicha elicitación. Para ayudar a la correcta construcción del LEL se incluye, al final de la correspondiente sección, un conjunto de guías, cuya finalidad es estandarizar el proceso constructivo del LEL, a partir de la correcta definición de sus componentes.

Prosiguiendo con los conceptos analizados, se continua con la introducción de la transformación propuesta por Antonelli et. al. [**38**] que permite derivar los Casos de Uso partiendo del LEL del dominio de una aplicación, dando lugar de esta manera a la posibilidad de utilizar el método de Use Case Points en los Casos de Uso derivados del Léxico Extendido del Lenguaje del dominio de una aplicación y, como se mencionó anteriormente, dicho LEL está disponible en etapas tempranas de la elicitación de requisitos. Partiendo de esto, el siguiente capítulo trata en detalle la obtención de la correspondiente estimación, proponiendo un método para obtenerla.

Para finalizar el capítulo se hizo una reseña del método denominado LEL Points, el que representa parte del estado del arte en lo que a estimación basada en LEL concierne, dicho método permite calcular LEL Points desajustados, que se compone de la suma del peso desajustado de verbos y del peso desajustado de los sujetos, Lo que representa una analogía con el Peso desajustado de los Casos de Uso en la técnica de Puntos Caso de Uso. Esta analogía planteada por LEL Points marca el camino a seguir por el método propuesto en el siguiente capítulo.

Estrategia

Este capítulo está dedicado a describir el método propuesto por el presente trabajo, así como también a describir la herramienta que se desarrolló para darle soporte a dicho método.

El presente capítulo está constituido por tres secciones, la primera corresponde al método planteado, en donde se lo describe y analiza como un todo, para luego hacer énfasis en cada etapa que lo compone. La segunda sección se centra en la herramienta construida para darle soporte a dicho método haciendo énfasis y describiendo sus principales características. Por último, la tercera sección analiza la implementación de los conceptos y técnicas más relevantes, haciendo énfasis en los principales problemas que surgieron en su desarrollo.

Método

Para comenzar con la explicación del método utilizaremos un enfoque que parta de la solución a una necesidad e iremos viendo cómo se amalgaman diferentes técnicas y conceptos para llegar a la satisfacción de dicho requerimiento, En este trabajo se planteó como necesidad poder calcular UCP partiendo desde el LEL del dominio de una aplicación.

Tomando esta necesidad como objetivo y viendo que a priori no puede utilizarse el LEL como entrada para el cálculo de UCP, se desarrolló un proceso de tipo pipeline que aplica transformaciones al LEL proporcionado como entrada, genera resultados intermedios y acepta entradas del usuario, produciendo como salida el resultado del cálculo de UCP integrando en un único proceso los conceptos introducidos en el capítulo anterior.

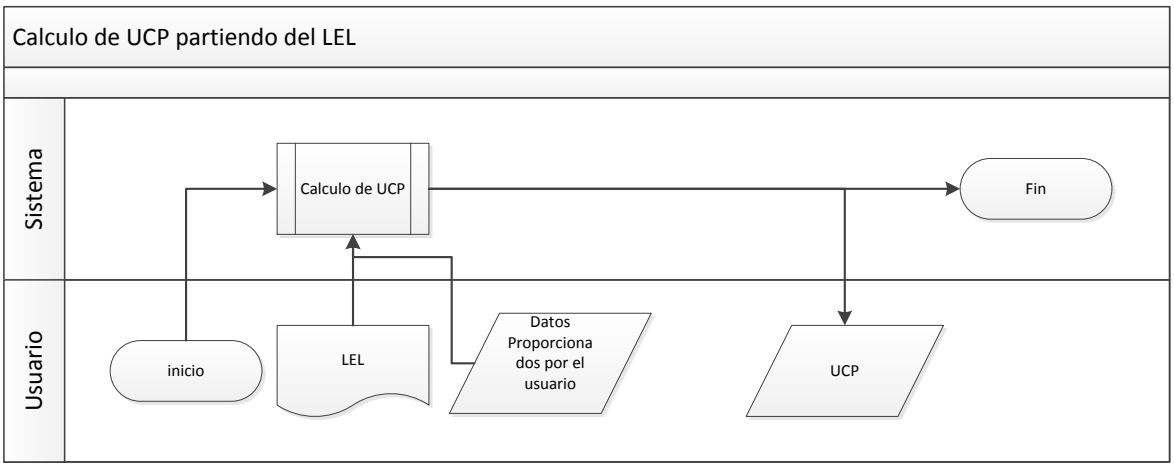

Ilustración 7 - Calculo de UCP Partiendo del LEL, Visión de Alto Nivel

El proceso creado en este trabajo parte de un LEL, a partir del cual genera los posibles Casos de Uso del proyecto, para ello aplica la transformación descripta por Antonelli et. al. [**38**] Que permite derivar Casos de Uso partiendo de un LEL, Los UC generados en este paso son Casos de Uso simples que cuentan con los datos mínimos e indispensables para que puedan ser utilizados en el cálculo de UCP.

Partiendo de estos Casos de Uso generados por la transformación, se obtiene los actores intervinientes en dichos UC, que posteriormente serán clasificados por el usuario quedando asociados a alguna de las categorías definidas en el método de UCP, a saber simple, medio, complejo o pudiendo ignorarlo en los cálculos si es categorizado para descartar, siendo esta una categoría introducida por este trabajo a fin de hacer más configurable el proceso.

Una vez obtenidos los UC, es necesario estimar el número de entidades de base de datos que se afectan y el número de clases intervinientes en cada Caso de Uso generado en el paso anterior, para ello este trabajo se basa en lo propuesto por Antonelli et al. en [**40**], adoptando la siguiente estrategia: para obtener el número de clases necesarias para implementar un UC se cuentan las referencias a términos de tipo sujeto en los pasos del Caso de Uso. Para obtener el número de entidades de base de datos afectadas por el UC se cuentan las referencias a términos de tipo objeto en los paso del Caso de Uso.

Las referencias a términos de tipo sujeto se utilizan para inferir el número de clases necesarias para implementar el UC, basándonos en [**28**] y debido a que por su definición, los términos de tipo sujeto son, elementos activos que realizan acciones. Esta definición permite hacer una analogía con las

clases, ya que serán estas las que tengan la responsabilidad de definir e implementar las acciones que podrán ejecutar sus instancias.

Las referencias a términos de tipo objeto se utilizan para inferir el número de entidades de base de datos afectadas por el UC, basándonos en [**28**] y debido a que por su definición son, elementos pasivos con los cuales los sujetos realizan acciones, evidenciando que las acciones que sobre ellos ejecuten los sujetos deberán registrarse y almacenarse de alguna manera, siendo por ello candidatos a representarse por entidades de base de datos que suplan esta necesidad de almacenamiento.

Cabe destacar que tanto los términos de tipo sujetos como los de tipo objeto son una especialización de lo que se consideran entidades y que a priori ambos podrían ser representados como entidades de base de datos, pero en este método se hace énfasis en el comportamiento que adoptan acorde a su definición, entendiendo que los de tipo sujeto se encargan de definir entidades que realiza acciones y los de tipo objeto representan entidades sobre las cuales estas acciones son ejecutadas e inherentemente provocaran cambios en su estado, dando lugar a la necesidad de almacenar dichos cambios.

Una vez obtenidos los UC, estimadas las entidades de base de datos afectadas por cada UC y las clases necesarias para su implementación se llega a un punto donde se vuelve necesaria la definición por parte del usuario de la complejidad de la interfaz de usuario que requerirá el UC, la definición de los factores de complejidad ambiental y los de complejidad técnica.

El proceso requiere que el usuario categorice la interfaz que requiere el UC de maneara acorde a lo especificado por la técnica de cálculo de UCP, a saber, el usuario debe asignarle al Caso de Uso una GUI compleja, media o simple. Pudiendo también en este momento descartar el Caso de Uso para no tenerlo en cuenta en el cálculo final. La participación del usuario es requería para esta asignación ya que la complejidad de la GUI no puede ser inferida a partir del LEL.

A su vez el usuario es responsable de definir los pesos asignados a los factores de complejidad técnica y a los factores de complejidad ambiental ya que los primeros representan la relevancia de ciertas características técnicas propias del desarrollo del proyecto y los segundos definen al equipo que intervendrá en el desarrollo del proyecto, quedando ambas caracterizaciones por fuera de los objetivos del LEL y por ende no pudiendo derivarlas del mismo.

En este punto del proceso se cuenta con información suficiente para calcular UCP, ya que en su cálculo intervienen UAW (calculado a partir de los actores y las categorizaciones de los mismos echa por el usuario), UUCW (calculado a partir de los Casos de Uso y de las correspondiente categorizaciones), TCF (calculado basándose en los pesos asignados a los factores de complejidad técnica) y ECF (calculado basándose en los pesos asignados a los factores de complejidad ambiental). Una vez calculados los Puntos Caso de Uso resta multiplicarlos por el factor de productividad para obtener una estimación en horas hombre del esfuerzo necesario para desarrollar el proyecto.

Para determinar el esfuerzo necesario en horas hombre se debe multiplicar el valor obtenido del cálculo de UCP por el factor de productividad (PF), este factor puede ser introducido por el usuario o se puede utilizar el valor sugerido por el sistema, que se calcula basándose en los datos introducidos por el usuario correspondientes a los factores de complejidad ambiental de la siguiente manera: Se contabilizan cuántos factores de los que afectan al factor de ambiente están por debajo del valor medio (3), para los factores E1 a E6. Se contabilizan cuántos factores de los que afectan al Factor de ambiente están por encima del valor medio (3), para los factores E7 y E8. Si el total es 2 o menos, se sugiere el factor de conversión 20 horas-hombre/Punto de Casos de Uso, es decir, un Punto de Caso de Uso toma 20 horas-hombre. Si el total es 3 o 4, se sugiere el factor de conversión 28 horas-hombre/Punto de Casos de Uso, es decir, un Punto de Caso de Uso toma 28 horas-hombre. Si el total es mayor o igual que 5, se sugiere el factor de conversión 36 horas-hombre/Punto de Casos de Uso, es decir, un Punto de Caso de Uso toma 36 horas-hombre.

Quedando de esta manera estimado el esfuerzo necesario para desarrollar el proyecto basándonos en la información contenida en el Léxico Extendido del Lenguaje, confeccionado en las primeras etapas de la elicitación de requisitos. Cabe destacar que como el proyecto se encuentra en una etapa temprana en la que los requerimientos son muy volátiles o poco explícitos, las estimaciones obtenidas por este (u otros) métodos aplicados a dichas etapas los hacen susceptibles a variaciones considerables en sus resultados, que dependen directamente de la calidad de los documentos que se utilizan como entrada para calcular las estimaciones mencionadas. Es menester del usuario aplicar el método tantas veces como la documentación o los parámetros requeridos cambien a lo largo del proyecto para que la estimación sea lo más precisa posible. A continuación se presenta una representación gráfica del proceso propuesto per este trabajo.

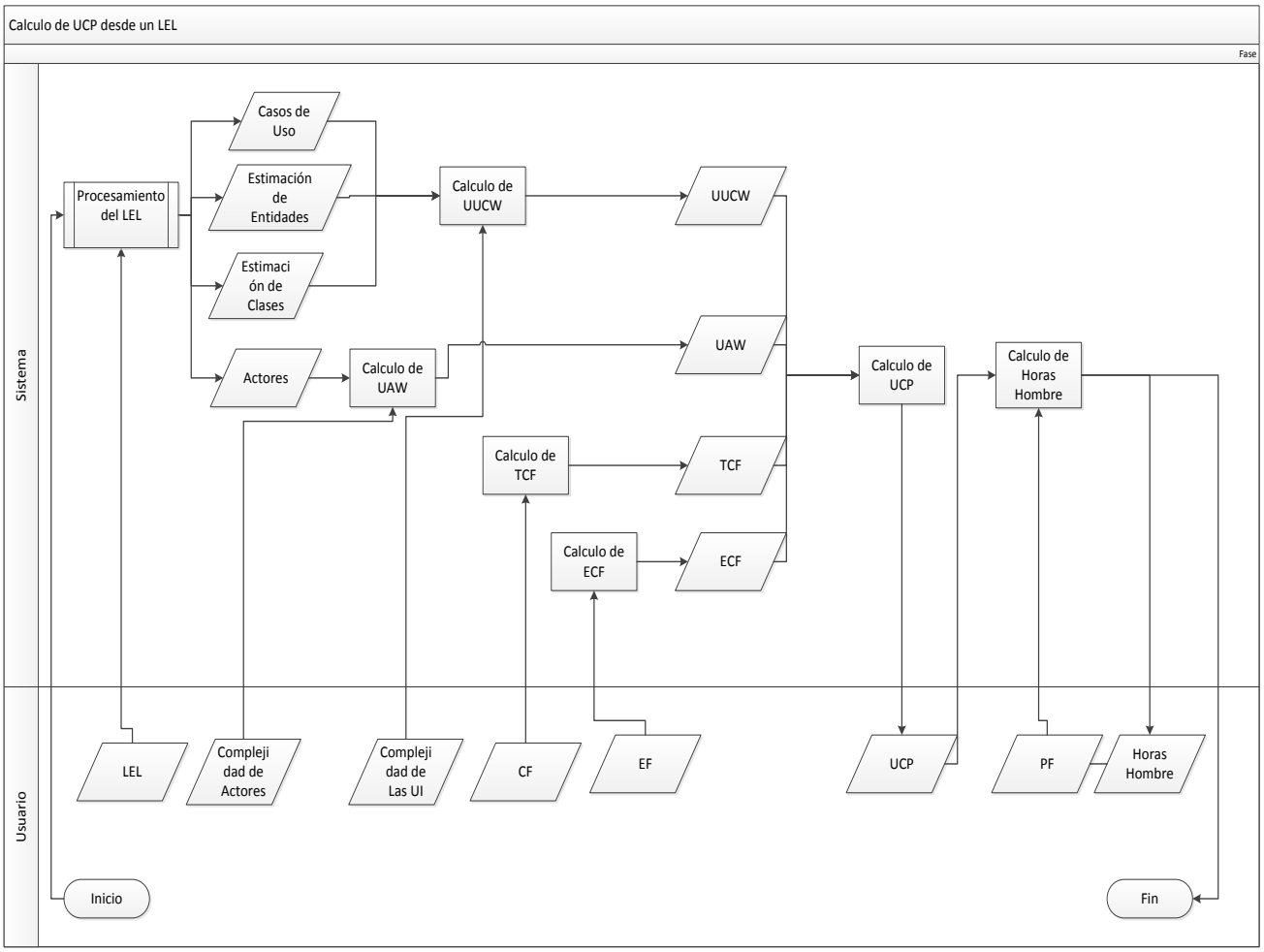

Ilustración 8 - Calculo de UCP Partiendo del LEL, Descripcion Grafica del Proceso

En la siguiente sección se describe la herramienta que se desarrolló para dar soporte al método planteado anteriormente partiendo de sus características principales y luego haciendo énfasis en los temas críticos que surgieron durante su desarrollo.

Herramienta

A lo largo de esta sección se analizara la herramienta desarrollada a fin de implementar el método propuesto en la sección anterior, para llevar a cabo este análisis primero introduciremos la herramienta como un todo y luego se analizaran sus principales características, y como estas permiten que el usuario lleve a cabo su cometido.

La herramienta desarrollada a lo largo de este trabajo es una aplicación Web que permite calcular Use Case Points a partir de un LEL, por ello, se la llamo “LEL to UCP”. Presenta una interfaz tabular en donde es posible acceder a sus principales funciones, que son la importación de LEL, la visualización del LEL importado, la visualización de los Casos de Uso generados, la categorización de UC y Actores, la especificación de los factores técnicos , la especificación de los factores ambientales, el cálculo de UCP y el cálculo de horas-hombre. A su vez la herramienta tiene la capacidad de gestionar cuentas de usuario permitiendo que cada uno administre sus propios proyectos.

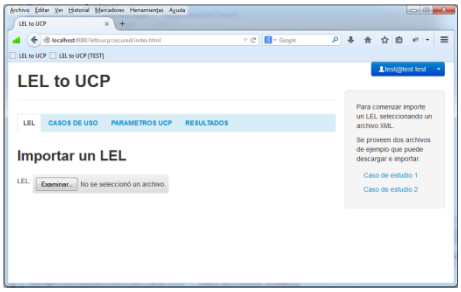

Ilustración 9 - Pantalla Principal de la Aplicación

La pantalla principal de la herramienta permite al usuario importar un LEL a la aplicación, para ello debe subir un archivo XML que contenga la definición a importar. Dicho archivo debe cumplir con las condiciones de formato y estructura definidas en la siguiente sección bajo el título “Formato de Archivos de LEL”. A fin de facilitar la comprensión de la herramienta su interfaz presenta ayuda contextual para las posibles interacciones de cada solapa y pone a disposición del usuario dos archivos de LEL que pueden usarse como ejemplo.

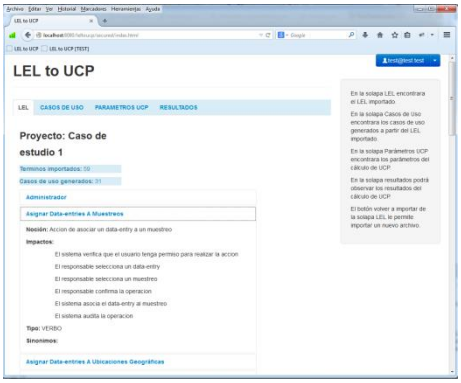

Ilustración 10 - Pantalla LEL

La pantalla de vista del LEL se presenta una vez que se importó exitosamente un archivo y muestra un resumen de la cantidad de símbolos procesado y los Casos de Uso creados, seguido por los símbolos importados. Por practicidad de uso, se utilizó una visualización expandible para contener la descripción de cada término, de manera tal que solo se visualiza su nombre, y al clicar sobre él, se detalla su noción, impactos, tipo de término y sinónimos. Si se clica sobre un término expandido este vuelve a su representación inicial.

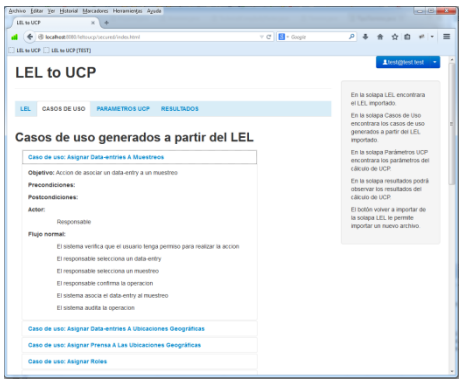

Ilustración 11 - Pantalla UC

Si el LEL importado se procesó exitosamente y la herramienta pudo derivar Casos de Uso según lo indicado por el método expuesto anteriormente, en la pantalla de vista de Casos de Uso se listan los UC generados a partir del LEL importado. Al igual que en la pantalla de vista del LEL se utilizó una visualización expandible que inicialmente muestra el título del UC y al clicar sobre este se muestra su objetivo, precondiciones, postcondiciones, actor principal, y su flujo normal, de la misma manera, si el usuario clica sobre un elemento expandido este se contrae a su representación inicial.

La pantalla de configuración de parámetros de UCP se compone de cuatro subpantallas correspondientes a los distintos grupos de parámetros, por una cuestión de organización conceptual se tiene las pantallas de configuración de peso de Casos de Uso desajustado, de peso de actores desajustado, de factores de complejidad técnica y de factores de complejidad ambiental.

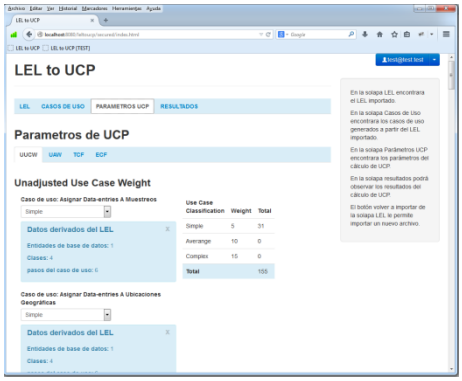

Ilustración 12 - Pantalla UUCW

La pantalla de peso de Casos de Uso desajustado (UUCW) informa por cada Caso de Uso generado por la herramienta los paramentos inferidos del LEL, a saber, el número de clases involucradas en su desarrollo y el número de entidades de base de datos afectadas por el UC. Esta pantalla permite cambiar la categorización del UC realizada por la herramienta o descartar el Caso de Uso en la estimación final y visualizar como estos cambios afectan el UUCW calculado.

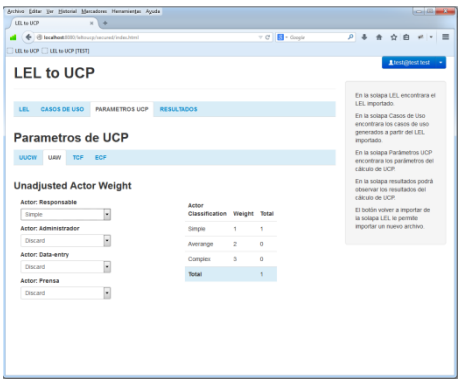

Ilustración 13 - Pantalla UAW

La pantalla de peso desajustado de actores (UAW) permite categorizar los actores detectado por la herramienta en las categorías previstas por el método UCP o descartar ciertos actores para su procesamiento, también permite visualizar como los cambios en las categorías impactan en el UAW calculado por la aplicación.

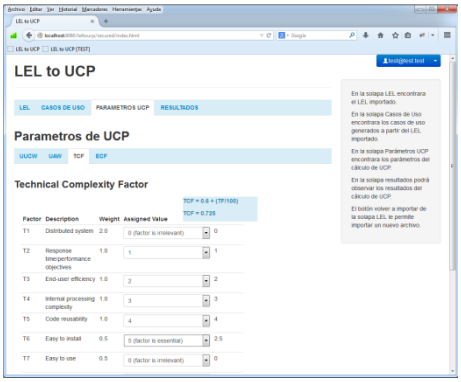

Ilustración 14 - Pantalla TCF

La pantalla correspondiente a los factores de complejidad técnica (TCF) permite configurar los pesos asignados a cada uno de los trece factores propuestos en la definición de UCP y visualizar como cada combinación de factores afecta el TCF calculado por la aplicación.

Al igual que la anterior la pantalla correspondiente a los factores de complejidad ambiental (ECF) permite configurar los pesos asignados a ocho factores definidos por UCP y visualizar su impacto en el ECF calculado por la herramienta.

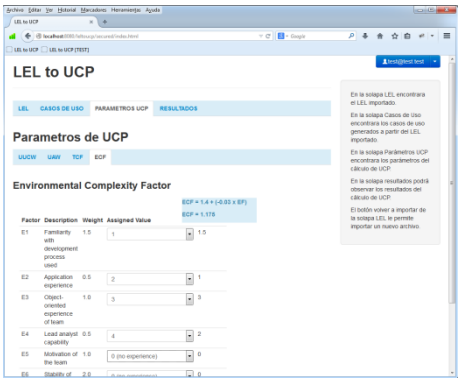

Ilustración 15 - Pantalla ECF

La pantalla de resultados presenta la información referente al cálculo de UCP siguiendo lo descripto en el capítulo anterior y además muestra el factor de productividad estimado desde los factores de complejidad ambiental, permitiendo su modificación y su correspondiente impacto en la estimación de Horas-Hombre necesarias para el desarrollo de los UC derivados del LEL.

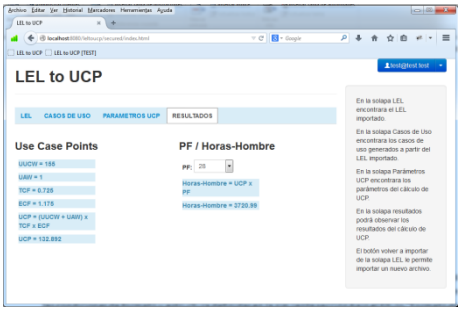

Ilustración 16 - Pantalla Resultados

Además de las características enunciadas la herramienta da soporte para la creación de cuentas de usuario, recuperación de contraseñas, y un manejo básico de los proyectos generados, manteniendo un historial de los proyectos importados por cada usuario y brindándole la posibilidad de abrirlos o eliminarlos en cualquier momento, a la vez que mantiene una consistencia entre los datos almacenados en el servidor y las acciones que el usuario ejecuta sobre la interfaz, manteniendo de esta manera los proyectos actualizados en todo momento. Estos aspectos y los detalles técnicos de implementación de la herramienta, formatos de archivos importados, instalación y requerimientos de la misma se abordan en detalle en el capítulo siguiente.

Conceptos y Técnicas

En esta sección se describe como la herramienta desarrollada implementa el método propuesto en la sección anterior, haciendo énfasis en los principales problemas que surgieron en su desarrollo. Primero se detalla lo referente a la implementación de la transformación que permite pasar de un LEL a Casos de Uso, luego se describe el algoritmo que se utiliza para inferir las clases y entidades necesarias en cada Caso de Uso derivado por la herramienta, también se detallan los datos de los cuales el usuario es responsable y por último se analiza como la herramienta utiliza la información generada y provista por el usuario para realizar las estimaciones y sugerir valores al operador.

Derivación de Casos de Uso desde el Léxico Extendido del Lenguaje

Para obtener un modelo de Casos de Uso a partir de un LEL, el método descripto en la sección anterior propone utilizar la transformación creada por Antonelli et. al. [**38**]. Esta transformación toma como entrada en LEL y genera un modelo de Casos de Uso como salida. Expresada en ATL [**39**] la transformación introducida en el capítulo anterior es la siguiente

```
helper LEL def: StateUsingVerbAsTransition(): Symbol =
(self.referencedInBehaviouralResponsesFrom()->select (x|x.isState())-> first ())
rule LEL2UseCase {
from s : Symbol (s.isVerb())
to u : UseCase (
    u.useCase <- s.name
    u.goalInContext <- s.notion
    u.preconditions <- (StateUsingVerbAsTransition).name
    u.successEndCondition <- (StateUsingVerbAsTransition)
    behaviouralResponses() ->
select (x|x.isState()) -> first()).name
    u.primaryActor <- s.referencedInBehaviouralResponsesFrom() ->
    select
     (x|x.isSubject()) -> first()
u.mainSuccessScenario <- s.behaviouralResponses }
```

Ilustración 17 - Transformación de LEL a Use Cases Expresada en ATL

Si analizamos como la implementación de la herramienta hace uso de los datos provistos por el LEL para generar los Casos de Uso tenemos que en primera instancia la herramienta realiza una clasificación de los símbolos según su tipo, luego de esto, por cada símbolo de tipo verbo genera un Caso de Uso al que va describiendo de la siguiente manera, primero define su título a partir de del termino (verbo) que está procesando, luego define el objetivo del UC correspondiendo con la noción del termino procesado, seguidamente hace uso de los términos de tipo estado para localizar las precondiciones del UC, a continuación hace uso de los términos tipo objeto y de los términos de tipo estado para localizar y asociar las post condiciones, posteriormente busca entre los términos de tipo sujeto si es posible asociar alguno de ellos como actor principal del UC y por último completa el flujo normal del UC basándose en los impactos definidos para el termino procesado. Viendo de forma gráfica el flujo de información dentro del algoritmo que implementa la transformación es el siguiente

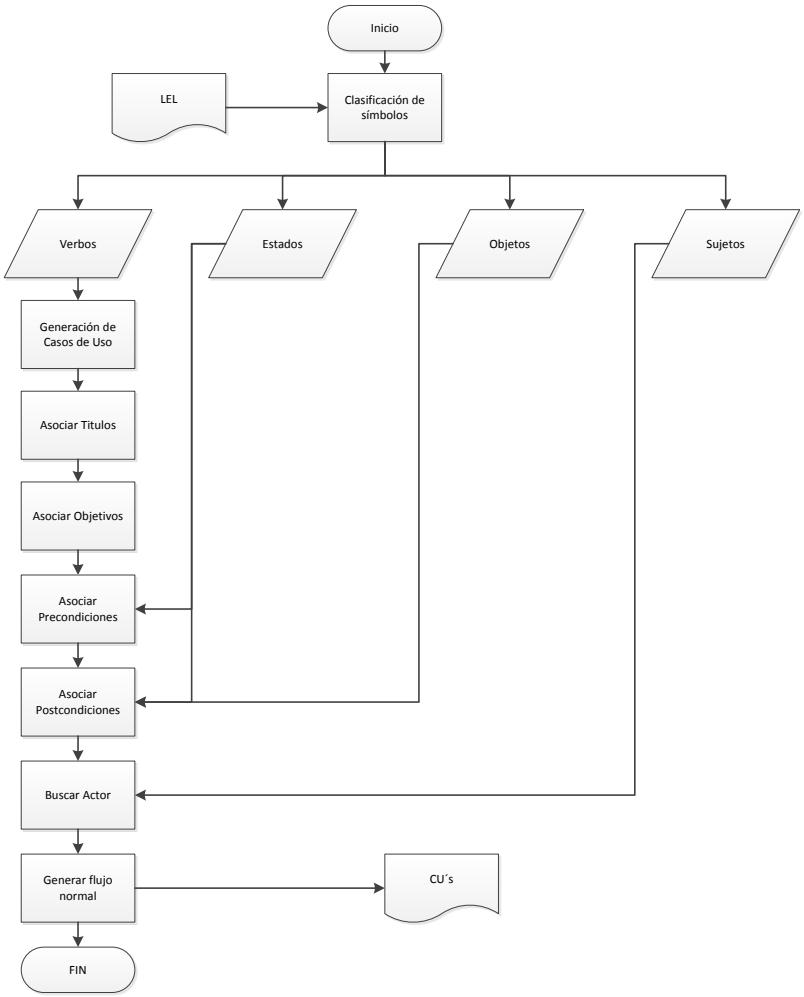

Ilustración 18 - Vista gráfica del Flujo de Información en la Transformación de LEL en CU

En primer lugar la aplicación parcea el LEL proporcionado por el usuario generando un modelo interno, que representa el conjunto de símbolos descripto en dicho LEL, consecuentemente procede a la categorización de dichos símbolos para facilitar su procesamiento, de esta manera divide los símbolos en cuatro colecciones, a saber verbos, objetos, sujetos y estados.

Para dar inicio a la generación de UC el algoritmo itera sobre la colección de términos de tipo verbo y por cada Verbo genera un Caso de Uso al cual titula, según la transformación utilizada, con el nombre del término que se está procesado y define sus objetivos acorde a la noción de dicho término.

Desde el punto de vista algorítmico las transformaciones necesarias para los pasos que abarcan la generación de UC, asociación de títulos, asociación de objetivos y generar el flujo normal no son más que adaptaciones de formato de la información para satisfacer el modelo de UC propuesto en la aplicación. Los pasos indicados como asociación de precondiciones, asociación de postcondiciones y buscar actores son más complejos y ameritan una descripción más profunda que se hace a continuación.

Para encontrar las precondiciones de un Caso de Uso, la transformación utilizada como modelo indica que es necesario encontrar un término de tipo Estado que utilice el Verbo que se está procesando como transición dentro de sus impactos. Considerando que el LEL cumple con los lineamientos descriptos por Antonelli et. al. en [**27**] (Guía 3), Las acciones descriptas en los impactos de los estados deben cumplir la estructura: sujeto + verbo + objeto + nuevo estado, Para encontrar la precondición, se debe analiza el conjunto de términos de tipo Estado en busca de aquel que utilice el

Verbo que se está procesando (o alguno de sus sinónimos) como transición en la descripción de sus impactos. Para esto se analiza heurísticamente la composición del nombre del Verbo y de cada uno de sus sinónimos en busca del verbo lingüístico que lo describe (para lograr esto se buscan los verbos en infinitivo). Una vez encontrados dichos verbos se procede a encontrar su raíz, que se utilizara luego junto con el resto de la oración que compone el nombre del Verbo procesado, para intentar ubicar un impacto de un Estado que utilice un verbo con la misma raíz como transición entre Estados y que posea la mayor cantidad de coincidencias con el resto de la oración. Determinando de esta manera que dicho Estado es la precondición al Caso de Uso que se está generando. Es necesario mencionar que dependiendo del dominio de la aplicación habrá UC que no posean precondiciones y que esta es una condición valida en la descripción de un Caso de Uso.

Para obtener la postcondición se toma como partida el término de tipo Estado utilizado como precondición, es decir que si no hay precondición no se podrá definir una postcondición. Partiendo del Estado que define la precondición del UC y basándose en la misma afirmación que la búsqueda de precondiciones se procede a buscar en los impactos del término Estado uno que en su composición lingüística termine referenciando a otro Estado, para ello se procede a descartar las referencias a términos de tipo objeto y todo lo que las antecede, ya que según [**27**] (Guía 3), Las acciones descriptas en los impactos de los estados deben cumplir la estructura: sujeto + verbo + objeto + nuevo estado, quedando de esta manera determinado el nuevo Estado, siendo este último, la postcondición indicada en la transformación. Es menester aclarar que la obtención de pre y post condiciones depende de que el LEL utilizado como entrada este descripto conforme a los lineamientos planteados en [**27**].

Al momento de obtener el actor principal del UC se procede a buscar entre los términos de tipo Sujeto, al que haga referencia al Verbo que está siendo procesado entre sus impactos, para esto el algoritmo se basa en [**27**] (Guía 2) que indica que las acciones descriptas en los impactos de sujetos, objetos y verbos deben cumplir la estructura: sujeto + verbo + objeto, entonces se procede con la búsqueda del verbo de la siguiente manera, primero se obtienen los verbos lingüísticos utilizados en el nombre y los sinónimos del término que se está procesando(a saber, el Verbo que determina el UC), luego se procede a encontrar heurísticamente la raíz de estos verbos y separarla del resto de la oración y por último se busca sobre los impactos de los Sujetos aquel que referencie al verbo indicado por la raíz y tenga la mayor cantidad de correspondencias en el resto de la oración, determinando de esta manera al Sujeto que será el actor principal del UC generado. De no encontrar un actor principal por esta vía se adopta otro enfoque para ubicar al Sujeto en cuestión, que consiste en una búsqueda plana del término de tipo sujeto que cuente con más referencias al Verbo que se está procesando entre sus impactos. Esto se hace debido a que muchas veces las composiciones lingüísticas propias del lenguaje llevan a usar variaciones de las palabras que dificultan su búsqueda, así como también los verbos irregulares complican estos procesos.

Por último, y como lo indica la transformación implementada, se procede a generar el flujo normal del Caso de Uso, para esto se crea un paso de dicho flujo por cada impacto que contenga el Verbo que se está procesando. De esta manera se completa la definición del UC de acuerdo con lo propuesto en este método. Es menester evidenciar que dicho UC variara conforme se modifique la definición del LEL utilizado como entrada y que es responsabilidad del usuario mantener esta consistencia, reimportando el LEL las veces que fuera necesario.

Derivación de Clases y Entidades desde el Léxico Extendido del Lenguaje

Además de la experiencia del usuario, dos factores importantes en la categorización de los Casos de Uso son, la cantidad de clases involucradas en el desarrollo del UC y la cantidad de entidades de base de datos afectadas por el UC. La herramienta desarrollada según el método propuesto calcula estos valores basándose en lo determinado en [**28**] y en las analogías trazadas en la sección anterior.

De esta manera para determinar el número de clases involucradas en el desarrollo del Caso de Uso, la herramienta cuenta las referencias que se hacen a términos de tipo Sujeto, este conteo se lleva a cabo dentro del bucle que itera los términos de tipo Verbo que se describió en la sección anterior y se realiza utilizando una heurística que facilita dicha búsqueda. Esta heurística consiste en estandarizar las oraciones que describen los impactos del Verbo que se está procesando, convirtiendo las palabas de la oración en singular y eliminado los símbolos de puntuación que pudiese contener y luego procediendo a comparar el resultado de la operación anterior con la estandarización correspondiente del nombre y cada uno de los sinónimos de los Sujetos descriptos en el LEL.

Prosiguiendo con este enfoque se realiza una operación similar para encontrar el número de entidades de base de datos afectadas por el Caso de Uso, con la diferencia de que en este caso se utiliza la misma heurística para realizar el conteo de las referencias que se hacen a términos de tipo Objeto en los impactos del Verbo que se está procesando.

Una vez determinados estos valores y basándose en lo descripto por el método UCP para el cálculo de UUCW, la herramienta realiza una categorización inicial del Caso de Uso asumiendo una interfaz de usuario simple y basándose solo en los valores de las entidades de base de datos afectadas, las clases involucradas en el UC y la cantidad de pasos en que consiste su flujo normal. Cabe destacar que el usuario de la herramienta debe determinar la complejidad de la UI de cada Caso de Uso y esto afectara la categorización inicial realizada por la herramienta que solo tiene por objetivo indicar al usuario cuales UC tiene mayor probabilidad de ser más complejos en su implementación.

Datos y Parámetros a Cargo del Usuario

En esta sección examinaremos los datos que la herramienta deja a cargo del usuario y el porqué de dicha decisión. A saber el usuario es responsable de ingresar la complejidad de la UI de cada Caso de Uso, de determinar la complejidad de los actores, de determinar los factores de complejidad técnica, los factores de complejidad ambiental y por ultimo de definir el factor de productividad.

Los datos enumerados en la sección anterior describen aspectos subjetivos del proyecto que se está realizando y dependen enteramente de la experiencia y capacitación final del usuario de la herramienta. La determinación de la complejidad de la UI o la exclusión de un UC impacta directamente en el cálculo de UUCW y depende puramente del conocimiento que se tenga en un momento determinado, de cuál es la metodología de interacción con el usuario, requerida para dicho UC. La categorización de un actor se ve reflejada directamente en el cálculo de UAW y depende de criterios que pueden no estar definidos al momento de ejecutar la herramienta o no ser determinados desde el LEL.

Por su parte los factores de complejidad técnica y ambiental describen la complejidad del producto y las capacidades del equipo de desarrollo, estos dependen del conocimiento de requerimientos no funcionales de la aplicación, así como también de un pleno conocimiento de las capacidades de las personas involucradas en el desarrollo del producto especificado, impactando en el cálculo final de UCP.

Por último el factor de productividad describe el tiempo que le toma al equipo de desarrollo ejecutar un Punto Caso de Uso, esta medida está influenciada por la complejidad del producto y las limitaciones del equipo de desarrollo, e impacta directamente en la conversión de UCP a horas hombre de esfuerzo.

Como se enuncio en los párrafos anteriores los factores descriptos en esta sección son a priori volátiles, en las etapas tempranas de la elicitación de requisitos, dependiendo estos de la información con la que se cuenta en un determinado momento, del juicio, sabiduría y experiencia de quien aplica el método implementado por esta herramienta, permitiéndole a quien define dichos valores experimentar con distintas combinaciones, buscando la que más se adapte a sus necesidades y quedando a su criterio la definición de estos parámetros.

Cálculos Realizados por la Herramienta

Siguiendo con lo especificado por el método de UCP, la herramienta desarrollada integra los datos proporcionados por el usuario con los inferidos desde el LEL para lograr una estimación del esfuerzo necesario para desarrollar la aplicación descripta en el mismo. A saber los cálculos realizados son los siguientes UAW, UUCW, TCF, ECF, UCP, PF estimado y las horas-hombre necesarias para el desarrollo. A continuación se describen que datos interviene en cada cálculo y de donde provienen.

Para el cálculo del peso desajustado de los actores (UAW), la herramienta cuenta con los datos provenientes de la categorización de los actores llevada a cabo por el usuario y la única operación necesaria es totalizar dichos valores según la siguiente formula.

$$UAW = A_s + A_m * 2 + A_c * 3$$

Siendo:

$$A_s = Cantidad\ de\ actores\ simples$$

$$A_m = Cantidad\ de\ actores\ medios$$

$$A_c = Cantidad\ de\ actores\ complejos$$

Al momento de calcular el peso desajustado de los Casos de Uso (UUCW), la herramienta cuenta con los datos provenientes de la categorización de la UI de cada UC realizada por el usuario y con los datos inferidos al procesar el LEL, a saber se tiene el número de pasos del flujo normal del UC, la cantidad de entidades de base de datos afectadas por el Caso de Uso y las clases necesarias para implementar dicho UC. Utilizando el conjunto de estos valores la herramienta categoriza los Casos de Uso que no se han excluido del cálculo según lo indicado por el método UCP en la Tabla 2 - Resumen de Casos de Uso.

Una vez realizada la categorización anterior se obtiene el número de Casos de Uso complejos, promedio y simples que componen la aplicación descripta en el LEL. Luego de efectuar dicha categoría la herramienta aplica la siguiente formula obteniendo así el UUCW.

$$UUCW = UC_s * 5 + UC_m * 10 + UC_c * 15$$

Siendo:

$$UC_s = Cantidad\ de\ casos\ de\ uso\ simples$$

$$UC_m = Cantidad\ de\ casos\ de\ uso\ medios$$

$$UC_c = Cantidad\ de\ casos\ de\ uso\ complejos$$

Continuando con lo enunciado la herramienta procede al cálculo del factor total de complejidad técnica (TCF) y al del factor de complejidad ambiental (ECF) totalizando los pesos asignados por el usuario a los factores correspondientes a cada categoría de la siguiente manera.

$$TCF = C_1 + C_2 \sum_{i=1}^{13} W_i * F_i$$

Siendo:

$$C_1 = 0.6$$

$$C_2 = .01$$

$$W_i = El\ Peso\ del\ factor\ i$$

$$F_i = El\ valor\ de\ complejidad\ percibida$$

$$ECF = C_1 + C_2 \sum_{i=1}^{13} W_i * F_i$$

Siendo:

$$C_1 = 0.6$$

$$C_2 = .01$$

$$W_i = El\ Peso\ del\ factor\ i$$

$$F_i = El\ valor\ de\ complejidad\ percibida$$

En este punto es posible el cálculo de los Puntos Caso de Uso (UCP) y para ello se combinan los resultados parciales obtenidos hasta el momento siguiendo la fórmula propuesta por el método correspondiente.

$$UCP = UUCP * TCF * ECF$$

En donde

$$UUCP = UAW + UUCW$$

Antes de realizar la estimación de horas-hombre necesarias para el desarrollo, la herramienta realiza un análisis de los factores de complejidad ambiental y de complejidad técnica para proporcionar un estimado del factor de productividad (PF) de la siguiente manera: Se contabilizan cuántos factores de los que afectan al factor de ambiente están por debajo del valor medio (3), para los factores E1 a E6. Se contabilizan cuántos factores de los que afectan al Factor de ambiente están por encima del valor medio (3), para los factores E7 y E8. Si el total es 2 o menos, se sugiere el factor de conversión 20 horas-hombre/Punto de Casos de Uso, es decir, un Punto de Caso de Uso toma 20 horas-hombre. Si el total es 3 o 4, se sugiere el factor de conversión 28 horas-hombre/Punto de Casos de Uso, es decir, un Punto de Caso de Uso toma 28 horas-hombre. Si el total es mayor o igual que 5, se sugiere el factor de conversión 36 horas-hombre/Punto de Casos de Uso, es decir, un Punto de Caso de Uso toma 36 horas-hombre.

Una vez estimado el PF, la herramienta permite que el usuario modifique el valor estimado y toma dicho valor para realizar la estimación de horas-hombre necesarias para el desarrollo de la aplicación siguiendo la fórmula que se enuncia a continuación.

$$Estimacion\ Total = UCP * PF$$

Palabras Finales

En este capítulo se presentó el método, objeto del presente trabajo, que permite obtener una estimación del esfuerzo necesario para el desarrollo de una aplicación, utilizando la técnica de Puntos Caso de Uso en etapas tempranas de la elicitación de requisitos, y se introdujo la herramienta LEL to UCP que se desarrolló para dar soporte a dicho método.

El método está basado en un proceso tipo pipeline que toma como entrada el Léxico Extendido de Lenguaje del dominio de una aplicación, el que debe estar definido siguiendo los lineamientos propuestos en el capítulo anterior, para luego transformarlos en Casos de Uso, siguiendo la transformación descrita previamente. Luego infiere las clases involucradas en cada uno de los Casos de Uso obtenidos y las entidades de base de datos afectadas en cada uno de dichos Casos de Uso, para luego requerir la intervención del usuario a fin de configurar los parámetros que no se pueden inferir desde el Léxico Extendido del Lenguaje, obteniendo de esta manera, la información necesaria para realizar la estimación mediante el método de Puntos de Casos de Uso, el que permite al final del pipeline mencionado, la obtención de los Puntos Caso de Uso necesarios para el desarrollo de la aplicación que se está estimando. Por último el método describe una técnica para calcular el factor de productividad basada en los factores ambientales ingresados por el usuario, permitiendo convertir los Puntos Caso de Uso en una estimación expresada en Horas-Hombre

Para dar soporte al método propuesto se desarrolló una herramienta que implementa el pipeline descripto en el párrafo anterior, la herramienta en cuestión es una aplicación Web que acepta como entrada el Léxico Extendido del Lenguaje de una aplicación mediante un archivo en formato XML, realiza la transformación propuesta e infiere lo parámetros mencionados mediante la aplicación de heurísticas, para luego presentar los resultados obtenidos al usuario, el que proporcionará los parámetros que no se pueden inferir desde el Léxico Extendido del Lenguaje. Una vez completados dichos parámetros, la aplicación presentará los resultados de la estimación en Puntos Caso de Uso y Horas-Hombre.

Por último se dedicó una sección a analizar los aspectos teóricos de los problemas surgidos en la implementación de la transformación y la inferencia de datos, describiendo las directrices adoptadas a fin de solucionar dichas dificultades en el proceso de desarrollo, que se abordarán desde una visión más técnica en el siguiente capítulo.

Aspectos Técnicos de la Herramienta

Este capítulo aborda los aspectos técnicos de la herramienta introducida en el capítulo anterior y brinda un manual de usuario a fin facilitar la utilización de la misma. Para estructurar dicho análisis se divide el capítulo en cuatro secciones principales, la primera trata las tecnologías utilizadas, la segunda el diseño de la aplicación, la tercera el proceso de instalación de la herramienta y por último se presenta el manual de usuario.

Tecnologías Aplicadas

En los títulos siguientes se enumeraran y describirán las tecnologías, productos, herramientas, de mayor importancia, que fueron utilizadas en el desarrollo de LEL to UCP. Comenzaremos esta reseña describiendo los principales elementos tecnológicos que su utilizaron en la interfaz de usuario, prosiguiendo con los propios del back-end de la aplicación, continuando con el motor de base de datos elegido, llegando en el final al entorno de desarrollo, el manejo de dependencias, control de código y el contenedor de aplicaciones WEB utilizado.

HTML

HTML, (del inglés, HyperText Markup Language), es el lenguaje de marcado estándar para la creación de páginas WEB, permite componer elementos definidos mediante tags para crear una página web, Un navegador web puede leer e interpretar los archivos escritos en HTML y componerlos en páginas web visibles o audibles. Los elementos HTML son los bloques de construcción de los sitios web, permiten embeber imágenes y objetos para crear formularios interactivos.

HTML provee una manera de crear documentos estructurados denotando su semántica estructural, por ejemplo, determinando encabezados, párrafos, listas, enlaces, etc. Un documento en HTML puede referenciar scripts escritos en JavaScript u otros lenguajes permitiendo que la ejecución de dichos scripts alteren el comportamiento del documento. Por otro lado, y a fin de manejar la visualización y el estilo del documento presentado al usuario por un navegador web, dicho navegador hace uso de las hojas de estilo CSS (del inglés, Cascading Style Sheets) para determinar la forma en la que se mostrara cada elemento del documento HTML. Como se enunció HTML al igual que CSS son estándares y su mantenimiento y evolución están a cargo del W3C (del inglés, World Wide Web Consortium).

En 1980 el físico Tim Berners-Lee, en aquel momento contratista del CERN, propuso y prototipó ENQUIRE, un sistema para usar y compartir documentos entre los investigadores del CERN, en 1989, propuso en un memo un sistema de hipertexto basado en la internet [**41**]. Para finales de la década de los 90, especifico HTML y desarrollo el software de navegador y del servidor web. El primer documento publicado que describe HTML es “HTML Tags”, puesto en internet por Berners-Lee a finales de 1991, describe dieciocho elementos que comprenden el diseño inicial de HTML.

Berners-Lee considero que HTML debía ser una aplicación de SGML y se define como tal por la IETF (de inglés, Internet Engineering Task Force) a mediados de 1993 con la publicación de la propuesta de la especificación de HTML "Hypertext Markup Language (HTML)" Internet-Draft, escrita por Berners-Lee y Dan Connolly, la que incluye un DTD (del inglés, Document Type Definition) de SGML para definir la gramática. El boceto expiró luego de seis meses, pero fue notable por su reconocimiento de la etiqueta propia del navegador Mosaic usada para insertar imágenes sin cambio de línea, que reflejaba la filosofía del IETF de basar estándares en prototipos con éxito. De la misma manera, el boceto competidor de Dave Raggett HTML+ (del inglés, Hypertext Markup Format), de finales de 1993, sugería estandarizar características ya implementadas, como las tablas. Luego de que ambos bocetos expiraran a principios de 1994 el IETF creo el HTML Working Group, el que en 1995 completo “HTML 2.0”, la primera especificación de HTML tratada como estándar y en la que se basaron las implementaciones futuras.

Desde 1996 la especificación de HTML es mantenida por el W3C, y a partir de año 2000 es considerada un estándar internacional por la norma ISO/IEC 15445:2000. Actualmente HTML se

encuentra en su versión 4.01 y desde el año 2004 se está desarrollando HTML 5 que se encuentra parcialmente implementado.

El HTML se escribe en forma de etiquetas, rodeadas por corchetes angulares (<,>). HTML también puede describir, hasta un cierto punto, la apariencia de un documento, y puede incluir o hacer referencia a un script, el cual puede afectar el comportamiento de navegadores web y otros procesadores de HTML. Consta de varios componentes vitales, entre ellos los elementos y sus atributos, tipos de dato y la declaración de tipo de documento.

Los elementos son la estructura básica de HTML, tienen dos propiedades básicas: atributos y contenido. Cada atributo y contenido tiene ciertas restricciones para que se considere válido al documento HTML. Un elemento generalmente tiene una etiqueta de inicio, por ejemplo, <nombre-de-elemento> y una etiqueta de cierre, por ejemplo, </nombre-de-elemento>. Los atributos del elemento están contenidos en la etiqueta de inicio y el contenido está ubicado entre las dos etiquetas, por ejemplo, <nombre-de-elemento atributo="valor">Contenido</nombre-de-elemento>. Algunos elementos, tales como
, no tienen contenido ni llevan una etiqueta de cierre.

El marcado estructural describe el propósito del texto. Por ejemplo, <h2></h2> establece su contenido como un encabezado de segundo nivel. El marcado estructural no define cómo se verá el elemento, pero la mayoría de los navegadores web han estandarizado el formato de los elementos. Puede aplicarse un formato específico al texto por medio de CSS.

El marcado presentacional describe la apariencia del texto, sin importar su función. Por ejemplo, indica que los navegadores web visuales deben mostrar el contenido en negrita, pero no indica qué deben hacer los navegadores web que muestran el contenido de otra manera (por ejemplo, los que leen el texto en voz alta). En el caso de e , existen elementos que se visualizan de la misma manera pero tienen una naturaleza más semántica: y . Es fácil ver cómo un lector de pantalla debería interpretar estos dos elementos. Sin embargo, son equivalentes a sus correspondientes elementos presentacionales: un lector de pantalla no debería decir más fuerte el nombre de un libro, aunque éste esté en itálicas en una pantalla. La mayoría del marcado presentacional ha sido desechada con HTML 4.0, en favor de CSS.

El marcado hipertextual se utiliza para enlazar partes del documento con otros documentos o con otras partes del mismo documento. Para crear un enlace es necesario utilizar la etiqueta de ancla <a> junto con el atributo href, que establecerá la dirección URL a la que apunta el enlace. Por ejemplo, http://www.un-sitio.org. También se pueden crear enlaces sobre otros objetos, tales como imágenes .

La mayoría de los atributos de un elemento son pares nombre-valor, separados por un signo de igual (=) y escritos en la etiqueta de comienzo de un elemento, después del nombre de éste. El valor puede estar rodeado por comillas dobles o simples, aunque ciertos tipos de valores pueden estar sin comillas en HTML (pero no en XHTML). De todas maneras, dejar los valores sin comillas es considerado poco seguro. En contraste con los pares nombre-elemento, hay algunos atributos que afectan al elemento simplemente por su presencia (tal como el atributo ismap para el elemento img).

El diseño en HTML, aparte de cumplir con las especificaciones propias del lenguaje, debe respetar ciertos criterios de accesibilidad web, siguiendo unas pautas o las normativas y leyes vigentes en los países donde se regule dicho concepto. Se encuentra disponible y desarrollado por el W3C a través

de las Pautas de Accesibilidad al Contenido Web WCAG 1.0 / WCAG 2.0, aunque muchos países tienen especificaciones propias, como es el caso de España con la Norma UNE 139803.

HTTP

HTTP (del inglés, Hypertext Transfer Protocol) es el protocolo usado en cada transacción de la World Wide Web, fue desarrollado por el W3C y la IETF, colaboración que culminó en 1999 con la publicación de una serie de RFC (del inglés, Request For Comments), el más importante de ellos es el RFC 2616 [**42**] [**43**] que especifica la versión 1.1.

HTTP define la sintaxis y la semántica que utilizan los elementos de software de la arquitectura web (clientes, servidores, proxies) para comunicarse. Es un protocolo orientado a transacciones perteneciente a la capa de aplicación y sigue el esquema petición-respuesta entre un cliente y un servidor. Al cliente que efectúa la petición se lo conoce como user agent. A la información transmitida se la llama recurso y se la identifica mediante un localizador uniforme de recursos URL (del inglés, Uniform Resource Locator). Los recursos pueden ser archivos, el resultado de la ejecución de un programa, una consulta a una base de datos, la traducción automática de un documento, etc. HTTP es un protocolo sin estado, es decir, que no guarda ninguna información sobre conexiones anteriores.

Una transacción HTTP está formada por un encabezado seguido, opcionalmente, por una línea en blanco y algún dato. El encabezado especificará cosas como la acción requerida del servidor, o el tipo de dato retornado, o el código de estado. El uso de campos de encabezados enviados en las transacciones HTTP le da gran flexibilidad al protocolo. Estos campos permiten que se envíe información descriptiva en la transacción, permitiendo así la autenticación, cifrado e identificación de usuario. Un encabezado es un bloque de datos que precede a la información propiamente dicha, por lo que muchas veces se hace referencia a él como metadato.

Como respuesta a un pedido HTTP el servidor retorna, junto con el cuerpo de la respuesta, un código de estado que indica si la petición fue correcta o no. Los códigos de error típicos indican que el archivo solicitado no se encontró, que la petición no se realizó de forma correcta o que se requiere autenticación para acceder al archivo.

HTTP permite enviar documentos de todo tipo y formato, es ideal para transmitir multimedia, como gráficos, audio y video. Esta libertad es una de sus mayores ventajas.

HTTP ha pasado por múltiples versiones del protocolo, muchas de las cuales son compatibles con las anteriores. El RFC 2145 describe el uso de los números de versión de HTTP. El cliente le dice al servidor al principio de la petición la versión que usa, y el servidor usa la misma o una anterior en su respuesta. La versión 0.9 que es considerada obsoleta. Soporta sólo un comando, GET, y además no especifica el número de versión HTTP. No soporta cabeceras. Como esta versión no soporta POST, el cliente no puede enviarle mucha información al servidor. La versión HTTP/1.0 de mayo de 1996 es la primera revisión del protocolo que especifica su versión en las comunicaciones, y todavía se usa ampliamente, sobre todo en servidores proxy. La versión HTTP/1.1 de junio de 1999 es la versión actual; las conexiones persistentes están activadas por defecto y funcionan bien con los proxies. También permite al cliente enviar múltiples peticiones a la vez por la misma conexión (pipelining) lo que hace posible eliminar el tiempo de Round-Trip delay por cada petición.

HTTP define ocho métodos, algunas veces referido como verbos, que indican la acción que desea que se efectúe sobre el recurso identificado. Lo que este recurso representa, si los datos pre-existentes o datos que se generan de forma dinámica, depende de la aplicación del servidor. A menudo, el recurso corresponde a un archivo o la salida de un ejecutable que residen en el servidor.

El método HEAD pide una respuesta idéntica a la que correspondería a una petición GET, pero sin el cuerpo de la respuesta. Esto es útil para la recuperación de meta-información escrita en los encabezados de respuesta, sin tener que transportar todo el contenido. El método GET pide una representación del recurso especificado. El método POST envía los datos para que sean procesados por el recurso identificado. Los datos se incluirán en el cuerpo de la petición. Esto puede resultar en la creación de un nuevo recurso o de las actualizaciones de los recursos existentes o ambas cosas. El método PUT sube, carga o realiza un upload de un recurso especificado, es el camino más eficiente para subir archivos a un servidor, esto es porque en POST utiliza un mensaje multiparte y el mensaje es decodificado por el servidor. En contraste, el método PUT permite escribir un archivo en una conexión socket establecida con el servidor. El método DELETE Borra el recurso especificado. El método TRACE solicita al servidor que envíe de vuelta en un mensaje de respuesta, en la sección del cuerpo de entidad, toda la data que reciba del mensaje de solicitud. Se utiliza con fines de comprobación y diagnóstico. El método OPTIONS devuelve los métodos HTTP que el servidor soporta para un URL específico. Esto puede ser utilizado para comprobar la funcionalidad de un servidor web mediante petición, en lugar de un recurso específico. El método CONNECT se utiliza para saber si se tiene acceso a un host, no necesariamente la petición llega al servidor, este método se utiliza principalmente para saber si un proxy da acceso a un host bajo condiciones especiales, como por ejemplo flujos de datos bidireccionales encriptados, como lo requiere SSL.

CSS

Hoja de estilo en cascada o CSS (del inglés, Cascading Style Sheets) [**44**] es un lenguaje usado para definir la presentación de un documento estructurado escrito en HTML o XML. El World Wide Web Consortium (W3C) es el encargado de formular la especificación de las hojas de estilo que servirán de estándar para los agentes de usuario o navegadores.

La idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación. La información de estilo puede ser definida en un documento separado o en el mismo documento HTML. En este último caso podrían definirse estilos generales en la cabecera del documento o en cada etiqueta particular mediante el atributo style.

CSS tiene una sintaxis muy sencilla, que usa unas cuantas palabras clave tomadas del inglés para especificar los nombres de varias propiedades de estilo. Una hoja de estilo se compone de una lista de reglas. Cada regla o conjunto de reglas consiste en uno o más selectores y un bloque de declaración (o bloque de estilo), con los estilos a aplicar para los elementos del documento que cumplan con el selector que les precede. Cada bloque de estilos se define entre llaves, y está formado por una o varias declaraciones de estilo con el formato propiedad:valor.

En CSS, los selectores marcarán qué elementos se verán afectados por cada bloque de estilo que les siga, y pueden afectar a uno o varios elementos a la vez, en función de su tipo, nombre (name), ID, clase (class), posición dentro del DOM (del inglés, Document Object Model).

Por ejemplo, el elemento de HTML <h1> indica que un bloque de texto es un encabezamiento y que es más importante que un bloque etiquetado como <h2>. Versiones antiguas de HTML permitían atributos extra dentro de la etiqueta abierta para darle formato (como el color o el tamaño de fuente). No obstante, cada etiqueta <h1> debía disponer de la información si se deseaba un diseño consistente para una página y, además, una persona que leía esa página con un navegador perdía totalmente el control sobre la visualización del texto.

Cuando se utiliza CSS, la etiqueta <h1> no debería proporcionar información sobre cómo será visualizado, solamente marca la estructura del documento. La información de estilo, separada en una hoja de estilo, especifica cómo se ha de mostrar <h1>: color, fuente, alineación del texto, tamaño y otras características no visuales, como definir el volumen de un sintetizador de voz, por ejemplo.

Por otro lado, antes de que estuviera disponible CSS, la única forma de componer espacialmente una página era el uso de tablas <table>. Aunque esta era una técnica cómoda y versátil, ello conllevaba el uso de un elemento con una semántica particular, y en el que la distribución de los datos no se ajustaban al flujo de la información que se obtenía en la vista desde los navegadores habituales, lo que redundaba en una merma en la accesibilidad a la página por parte de otros navegadores (orientados a personas con alguna deficiencia sensorial, o a ciertos dispositivos electrónicos). Mediante el uso de CSS, se ha permitido eliminar el uso de tablas para el diseño, usándolas solamente para la muestra de datos tabulados.
Para dar formato a un documento HTML, puede emplearse CSS de tres formas distintas, un estilo en línea (inline) es un método para insertar el lenguaje de estilo de página directamente dentro de una etiqueta HTML. Esta manera de proceder no es totalmente adecuada. El incrustar la descripción del formateo dentro del documento de la página Web, a nivel de código, se convierte en una manera larga, tediosa y poco elegante de resolver el problema de la programación de la página. Dado que los clientes de correo electrónico no soportan las hojas de estilos externas, y que no existen estándares que los fabricantes de clientes de correo respeten para utilizar CSS en este contexto, la solución más

recomendable para maquetar correos electrónicos, es utilizar CSS dentro de los propios elementos (inline).

Una hoja de estilo interna, que es una hoja de estilo que está incrustada dentro de un documento HTML, dentro del elemento <head>, marcada por la etiqueta <style>. De esta manera se obtiene el beneficio de separar la información del estilo del código HTML propiamente dicho. Se puede optar por copiar la hoja de estilo incrustada de una página a otra (esta posibilidad es difícil de ejecutar si se desea para guardar las copias sincronizadas). En general, la única vez que se usa una hoja de estilo interna, es cuando se quiere proporcionar alguna característica a una página Web en un simple fichero, por ejemplo, si se está enviando algo a la página Web.

Una hoja de estilo externa, es una hoja de estilo que está almacenada en un archivo diferente al archivo donde se almacena el código HTML de la página Web. Esta es la manera de programar más potente, porque separa completamente las reglas de formateo para la página HTML, de la estructura básica de la página.

CSS se ha creado en varios niveles y perfiles, cada nivel de CSS se construye sobre el anterior, generalmente añadiendo funciones al previo. Los perfiles son, generalmente, parte de uno o varios niveles de CSS definidos para un dispositivo o interfaz particular. Actualmente, pueden usarse perfiles para dispositivos móviles, impresoras o televisiones.

La primera especificación oficial de CSS, recomendada por la W3C fue CSS1 [**45**], publicada en diciembre 1996, y abandonada en abril de 2008, define un conjunto de funcionalidades que incluyen propiedades de las fuentes como color de texto, fondos, bordes u otros elementos. Atributos del texto, como espaciado entre palabras, letras, líneas. Alineación de textos, imágenes, tablas u otros. Propiedades de caja, como margen, borde, relleno o espaciado. Propiedades de identificación y presentación de listas.

La especificación CSS2 [**46**]fue desarrollada por la W3C y publicada como recomendación en mayo de 1998, y abandonada en abril de 2008. Como ampliación de CSS1, se ofrecieron, entre otras características, las funcionalidades propias de las capas como posicionamiento relativo, absoluto, fijo, niveles z-index, etcétera. El concepto de media types. Soporte para las hojas de estilo auditivas. Texto bidireccional, sombras, etcétera.

La primera revisión de CSS2, usualmente conocida como CSS 2.1 [**47**], corrige algunos errores encontrados en CSS2, elimina funcionalidades poco soportadas o inoperables en los navegadores y añade alguna nueva especificación. De acuerdo al sistema de estandarización técnica de las especificaciones, CSS2.1 tuvo el estatus de candidate recommendation durante varios años, pero la propuesta fue rechazada en junio de 2005; en junio de 2007 fue propuesta una nueva versión candidata, y ésta actualizada en 2009, pero en diciembre de 2010 fue nuevamente rechazada. En abril de 2011 CSS 2.1 volvió a ser propuesta como candidata, y después de ser revisada por el W3C Advisory Committee, fue finalmente publicada como recomendación oficial el 7 de junio de 2011.

A diferencia de CSS2, que fue una gran especificación que definía varias funcionalidades, CSS3 [**48**] está dividida en varios documentos separados, llamados módulos. Cada módulo añade nuevas funcionalidades a las definidas en CSS2, de manera que se preservan las anteriores para mantener la compatibilidad. Los trabajos en CSS3, comenzaron a la vez que se publicó la recomendación oficial de CSS2, y los primeros borradores de CSS3 fueron liberados en junio de 1999. Debido a la modularización de CSS3, diferentes módulos pueden encontrarse en diferentes estados de su

desarrollo, de forma que a fechas de noviembre de 2011, hay alrededor de cincuenta módulos publicados, tres de ellos se convirtieron en recomendaciones oficiales de la W3C en 2011: Selectores, Espacios de nombres y Color.

Una de las limitaciones que se encuentran en el uso del CSS hasta la versión CSS2.1, vigente, es que los selectores no pueden usarse en orden ascendente según la jerarquía del DOM (hacia padres u otros ancestros) como se hace mediante XPath, la razón que se ha usado para justificar esta carencia por parte de la W3C, es para proteger el rendimiento del navegador, que de otra manera, podría verse comprometido.

También se puede considerar una desventaja la dificultad para el alineamiento vertical; así como el centrado horizontal se hace de manera evidente en CSS2.1, el centrado vertical requiere de diferentes reglas en combinaciones no evidentes, o no estándares. La ausencia de expresiones de cálculo numérico para especificar valores o que las pseudo-clases dinámicas como :hover no se pueden controlar o deshabilitar desde el navegador, lo que las hace susceptibles de abuso por parte de los diseñadores en banners, o ventana emergentes.

Entre las ventajas cabe destacar el control centralizado de la presentación de un sitio web completo con lo que se agiliza de forma considerable la actualización del mismo. La separación del contenido de la presentación, que facilita al creador, diseñador, usuario o dispositivo electrónico que muestre la página, la modificación de la visualización del documento sin alterar el contenido del mismo, sólo modificando algunos parámetros del CSS. También es necesario considerar la optimización del ancho de banda de la conexión, pues pueden definirse los mismos estilos para muchos elementos con un sólo selector; o porque un mismo archivo CSS puede servir para una multitud de documentos. Pero uno de los aspectos en los que más impacta el uso de CSS es en la accesibilidad del documento, pues con el uso del CSS se evitan antiguas prácticas necesarias para el control del diseño, como las tablas, que iban en perjuicio de ciertos usos de los documentos, por parte de navegadores orientados a personas con algunas limitaciones sensoriales.

XML

XML (del inglés, eXtensible Markup Language) es un lenguaje de marcas desarrollado por el W3C [**49**] [**50**] utilizado para almacenar datos en forma legible. Deriva del lenguaje SGML y permite definir la gramática de lenguajes específicos para estructurar documentos grandes. A diferencia de otros lenguajes, XML da soporte a bases de datos, siendo útil cuando varias aplicaciones deben comunicarse entre sí o integrar información. XML se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable. Es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

Proviene de un lenguaje inventado por IBM en los años setenta, llamado GML (del inglés, Generalized Markup Language), que surgió por la necesidad que tenía la empresa de almacenar grandes cantidades de información, en 1986 trabajaron conjuntamente con la ISO para normalizarlo, creando SGML (del inglés, Standard Generalized Markup Language), capaz de adaptarse a un gran abanico de problemas. A partir de él se han creado otros lenguajes para almacenar información.

En el año 1989 Tim Berners-Lee creó la web, y junto con ella el lenguaje HTML. Este lenguaje se definió en el marco de SGML y es ampliamente la aplicación más conocida de este estándar. Los navegadores web, sin embargo siempre han puesto pocas exigencias al código HTML que interpretan, y así las páginas web son caóticas y no cumplen con la sintaxis. Estas páginas web dependen fuertemente de una forma específica de lidiar con los errores y las ambigüedades, lo que hace a las páginas más frágiles y a los navegadores más complejos. Otra limitación del HTML es que cada documento pertenece a un vocabulario fijo, establecido por un DTD, por lo tanto no se pueden combinar elementos de diferentes vocabularios. Asimismo es imposible para un intérprete analizar el documento sin tener conocimiento de su gramática, por ello los navegadores resolvieron esto incluyendo lógica ad hoc para el HTML, en vez de incluir un analizador genérico.

Se buscó entonces definir un subconjunto del SGML que permita mezclar elementos de diferentes lenguajes, es decir, que los lenguajes sean extensibles, que la creación de parsers sea simple, sin ninguna lógica especial para cada lenguaje, y que no se acepte nunca un documento con errores de sintaxis. Para hacer esto XML deja de lado muchas características de SGML que estaban pensadas para facilitar la escritura manual de documentos, XML en cambio está orientado a hacer las cosas más sencillas para los programas automáticos que necesiten interpretar el documento.

XML y sus extensiones han sido regularmente criticadas por su nivel de detalle y complejidad. El mapeo del modelo de árbol básico de XML hacia los sistemas de tipos de lenguajes de programación o bases de datos puede ser difícil, especialmente cuando se utiliza XML para el intercambio de datos altamente estructurados entre aplicaciones.

Por otro lado XML es extensible, después de diseñado y puesto en producción, es posible extender XML con la adición de nuevas etiquetas, de modo que se pueda continuar utilizando sin complicación alguna, El analizador es un componente estándar, no es necesario crear uno específico para cada versión de lenguaje, esto posibilita el empleo de cualquiera de los analizadores disponibles, evitando bugs y acelerando el desarrollo de aplicaciones. Si un tercero decide usar un documento creado en XML, es sencillo entender su estructura y procesarlo, mejorando la compatibilidad entre aplicaciones, permitiendo comunicar aplicaciones de distintas plataformas, sin que importe el origen de los datos.

Otra ventaja de XML es que transforma datos en información, pues se le añade un significado concreto y la asocia a un contexto, con lo cual se obtiene flexibilidad para estructurar documentos.

La tecnología XML busca dar solución al problema de expresar información estructurada de la manera más abstracta y reutilizable posible. Que la información sea estructurada quiere decir que se compone de partes bien definidas, y que esas partes se componen a su vez de otras partes. Entonces se tiene un árbol de trozos de información, que se denominan elementos, y se los señala mediante etiquetas. Una etiqueta consiste en una marca hecha en el documento, que señala una porción de éste como un elemento, un pedazo de información con un sentido claro y definido, las etiquetas o tags, tienen la forma <nombre>, donde nombre es el nombre del elemento que se está señalando.

Los documentos denominados bien formados son aquellos que cumplen con todas las definiciones básicas de formato y pueden, por lo tanto, analizarse correctamente por cualquier analizador sintáctico (parser) que cumpla con la norma. Los documentos deben seguir una estructura estrictamente jerárquica con lo que respecta a las etiquetas que delimitan sus elementos. Una etiqueta debe estar correctamente incluida en otra, es decir, deben estar correctamente anidadas. Los elementos con contenido deben estar correctamente cerrados con su etiqueta de finalización. Los documentos XML sólo permiten un elemento raíz del que todos los demás sean parte, es decir, solo pueden tener un elemento inicial. Los valores de los atributos en XML siempre deben estar encerrados entre comillas simples o dobles.

El XML es sensible a mayúsculas y minúsculas y existe un conjunto de caracteres llamados espacios en blanco (espacios, tabuladores, retornos de carro, saltos de línea) que los procesadores XML tratan de forma diferente en el marcado XML.

Es necesario asignar nombres a las estructuras, tipos de elementos, entidades, elementos particulares, etc. Las construcciones como etiquetas, referencias de entidad y declaraciones, se denominan marcas; son partes del documento que el procesador XML espera entender. El resto del documento entre marcas son los datos comprensibles por las personas.

Un documento XML está formado por el prólogo y por el cuerpo del documento, aunque no es obligatorio, los documentos XML pueden empezar con unas líneas que describen la versión XML, el tipo de documento y otras cosas, el prólogo de un documento XML contiene una declaración XML que es la sentencia que declara o identifica al documento como un documento XML. Una declaración de tipo de documento que, enlaza al documento con su DTD o esquema. En algunos casos el DTD puede estar incluido en la propia declaración. El Prólogo puede incluir comentarios e instrucciones de procesamiento. Un ejemplo simple consistiría en una sentencia como <?xml version="1.0" encoding="UTF-8"?>

A diferencia del prólogo, el cuerpo no es opcional en un documento XML, este debe contener solo un elemento raíz, característica indispensable también para que el documento esté bien formado. Los elementos XML pueden tener contenido (más elementos, caracteres o ambos), o bien ser elementos vacíos, también pueden tener atributos, que son una manera de incorporar características o propiedades a los elementos de un documento, estos deben ir entre comillas. Por ejemplo, <elemento atributo = “Valor”></elemento>.Existen entidades para representar caracteres especiales para que, de esta forma, no sean interpretados como marcado en el procesador XML, como por ejemplo, la entidad predefinida: & que identifica al carácter &. También existen las secciones CDATA que son una construcción en XML para especificar datos utilizando cualquier carácter sin que se interprete como marcado XML. Por último es posible agregar comentarios a modo informativo

para el programador que han de ser ignorados por el procesador, estos tienen el siguiente formato <!--- comentario ---> o <!-- comentario -->.

Un documento bien formado solamente indica que su estructura sintáctica básica es correcta, es decir, que se componga de elementos, atributos y comentarios como XML especifica que se escriban. Cada aplicación de XML, es decir, cada lenguaje definido con esta tecnología, necesitará especificar cuál es exactamente la relación que debe verificarse entre los distintos elementos presentes en el documento. Esta relación entre elementos se especifica en un documento externo o definición que puede ser un DTD (del inglés, Document Type Definition) o como un esquema definido en XSchema. Crear una definición equivale a crear un nuevo lenguaje de marcado, para una aplicación específica. El DTD define los tipos de elementos, atributos y entidades permitidas, y puede expresar algunas limitaciones para combinarlos. Los documentos XML que se ajustan a su DTD son denominados válidos.

Un Schema es algo similar a un DTD, define qué elementos puede contener un documento XML, cómo están organizados, qué atributos y de qué tipo pueden tener sus elementos, los esquemas se definen usando XML, permiten especificar los tipos de datos y son extensibles.

XML Schema es un lenguaje de esquema utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML de una forma muy precisa, más allá de las normas sintácticas impuestas por el propio lenguaje XML. Se consigue así una percepción del tipo de documento con un nivel alto de abstracción. Fue desarrollado por el W3C y alcanzó el nivel de recomendación en mayo de 2001.

El W3C empezó a trabajar en XML Schema en 1998. La primera versión se convirtió en una recomendación oficial en mayo de 2001. Una segunda edición revisada está disponible desde octubre de 2004. Esta recomendación está desarrollada en tres partes, XML Schema Parte 0 Primer [**51**], que es una introducción no normativa al lenguaje, que proporciona una gran cantidad de ejemplos y explicaciones detalladas para una primera aproximación a XML Schema. XML Schema Parte 1 Structures [**52**] que es una extensa descripción de los componentes del lenguaje y XML Schema Parte 2 Datatypes [**53**] que complementa la Parte 1 con la definición de los tipos de datos incorporados en XML Schema y sus restricciones.

XML Schema es un lenguaje de esquema escrito en XML, basado en la gramática y pensado para proporcionar una mayor potencia expresiva que los DTD. Los documentos se concibieron como una alternativa a los DTD, intentando superar sus puntos débiles y buscar nuevas capacidades a la hora de definir estructuras para documentos XML. El principal aporte de XML Schema es el gran número de tipos de datos que incorpora. De esta manera, XML Schema aumenta las posibilidades y funcionalidades de aplicaciones de procesado de datos, incluyendo tipos de datos complejos como fechas, números y strings. Los esquemas XML fueron diseñados completamente alrededor de namespaces [**54**] y soportan tipos de datos típicos de los lenguajes de programación, como también tipos personalizados simples y complejos. La programación en Schema XML se basa en namespaces o espacios de nombres, se puede trazar una analogía entre éstos y los packages en Java, cada namespace contiene elementos y atributos que están estrechamente relacionados entre sí.

La Validación XML es la comprobación de que un documento en lenguaje XML está bien formado y se ajusta a una estructura definida, es decir, que el documento sigue las reglas básicas de XML establecidas para el diseño de documentos y además respeta las normas dictadas por su DTD o esquema utilizado.

En primer lugar, los documentos XML deben basarse en la sintaxis definida en la especificación XML para ser correctos (documentos bien formados). Esta sintaxis impone cosas como la coincidencia de mayúsculas/minúsculas en los nombres de etiqueta, comillas obligatorias para los valores de atributo, etc. Sin embargo, para tener un control más preciso sobre el contenido de los documentos es necesario un proceso de análisis más exhaustivo. La validación es la parte más importante dentro de este análisis, ya que determina si un documento creado se ciñe a las restricciones descritas en el esquema utilizado para su construcción. Controlar el diseño de documentos a través de esquemas aumenta su grado de fiabilidad, consistencia y precisión, facilitando su intercambio entre aplicaciones y usuarios.

JSon

JSON (del inglés, JavaScript Object Notation) es un formato ligero de intercambio de datos, estandarizado mediante RFC-7159 [**55**] y ECMA-404 [**56**]. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo. Está basado en un subconjunto del Lenguaje de Programación JavaScript [**57**]. JSON es un formato de texto que es completamente independiente del lenguaje pero utiliza convenciones que son ampliamente conocidos por los programadores de la familia de lenguajes C, incluyendo C, C++, C#, Java, JavaScript, Perl, Python, y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos.

JSON está constituido por dos estructuras, una colección de pares de nombre/valor, que en varios lenguajes es conocida como un objeto, registro, estructura, diccionario, tabla hash, lista de claves o un arreglo asociativo, y una lista ordenada de valores que en la mayoría de los lenguajes, se implementa como arreglos, vectores, listas o secuencias.

Estas son estructuras universales; virtualmente todos los lenguajes de programación las soportan de una forma u otra. Es razonable que un formato de intercambio de datos que es independiente del lenguaje de programación se base en estas estructuras.

En JSON, un objeto es un conjunto desordenado de pares nombre/valor, comienza con { (llave de apertura) y termine con } (llave de cierre). Cada nombre es seguido por : (dos puntos) y los pares nombre/valor están separados por , (coma). Un arreglo es una colección de valores. Un arreglo comienza con [(corchete izquierdo) y termina con] (corchete derecho). Los valores se separan por , (coma). Un valor puede ser una cadena de caracteres con comillas dobles, o un número, o true o false o null, o un objeto o un arreglo, estas estructuras pueden anidarse. Una cadena de caracteres es una colección de cero o más caracteres Unicode, encerrados entre comillas dobles, usando barras divisorias invertidas como escape. Un carácter está representado por una cadena de caracteres de un único carácter. Una cadena de caracteres es parecida a una cadena de caracteres C o Java. Un número es similar a un número C o Java, excepto que no se usan los formatos octales y hexadecimales.

JQuery / JQuery UI

jQuery [**58**] es una biblioteca de JavaScript, creada inicialmente por John Resig, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX (del inglés, Asynchronous JavaScript And XML)a páginas web. Fue presentada el 14 de enero de 2006 en el BarCamp NYC. Es una de las bibliotecas de JavaScript más utilizadas.

Se basa en el modelo de software libre y de código abierto, posee un doble licenciamiento bajo la Licencia MIT y la Licencia Pública General de GNU v2, permitiendo su uso en proyectos libres y privados. Al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en JavaScript que de otra manera requerirían de mucho más código, es decir, con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y espacio.

Entre sus principales características se encuentran la selección de elementos DOM, la posibilidad de Interactuar y modificar el árbol DOM, incluyendo soporte para CSS3 y un plugin básico de XPath. También maneja eventos, permite la manipulación de las hojas de estilos CSS, el agregado de efectos y animaciones, incluyendo la creación de animaciones personalizadas. Pero sus principales virtudes radican en la simplificación de las técnicas involucradas en AJAX y su soporte para el añadido de extensiones de terceras partes. Además de las características enunciadas brinda utilidades varias como obtener información del navegador, operar con objetos y vectores, funciones para rutinas comunes, etc.

Es Compatible con los navegadores Mozilla Firefox 2.0+, Internet Explorer 6+, Safari 3+, Opera 10.6+ y Google Chrome 8+. brindando a los usuarios la capacidad de desarrollar contenido para distintos navegadores (cross-browser) utilizando solo una librería, simplificando el proceso de desarrollo de dichos contenidos y haciéndolos más mantenibles.

jQuery consiste en un único fichero JavaScript que contiene las funcionalidades comunes de DOM, eventos, efectos y AJAX. La característica principal de la biblioteca es que permite cambiar el contenido de una página web sin necesidad de recargarla, mediante la manipulación del árbol DOM y peticiones AJAX. Para ello utiliza las funciones $() o jQuery().La forma de interactuar con la página es mediante la función $(), un alias de jQuery(), que recibe como parámetro una expresión CSS o el nombre de una etiqueta HTML y devuelve todos los nodos que concuerden con la expresión. Esta expresión es denominada selector en la terminología de jQuery. Una vez obtenidos los nodos, se les puede aplicar cualquiera de las funciones que facilita la biblioteca o que el usuario haya definido.

jQuery UI [**59**] es una biblioteca de componentes para el framework jQuery que le añaden un conjunto de plug-ins, widgets y efectos visuales para la creación de aplicaciones web. Cada componente o módulo se desarrolla de acuerdo a la filosofía de jQuery, encuentra algo, manipúlalo.

La biblioteca se divide en cuatro módulos, el núcleo, que contiene las funciones básicas para el resto de los módulos, Las interacciones que añaden o simplifican comportamientos complejos a los elementos del DOM, los widgets, que son un conjunto de controles de UI que pueden ser configurados gracias a que poseen un conjunto de opciones definibles por el usuario y que son compatibles con CSS, y por ultimo están los efectos, que brindan una API que permite añadir efectos visuales complejos a los componentes.

La principal ventaja de la extensión radica en la cantidad de componentes disponibles para el usuario, su capacidad de funcionar en la mayoría de los navegadores y la adaptación a casi todos los frameworks HTML – JavaScript – CSS que posibilitan la creación de interfaces de usuario ricas.

Twitter Bootstrap

Es un framework o conjunto de herramientas de software libre para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como, extensiones de JavaScript opcionales adicionales.

Fue desarrollado por Mark Otto y Jacbod Thornton de Twitter, como un framework para fomentar la consistencia a través de herramientas internas. Antes de Bootstrap, se usaban varias librerías para el desarrollo de interfaces de usuario, las cuales guiaban a inconsistencias y a una carga de trabajo alta en su mantenimiento.

En agosto del 2011, Twitter liberó a Bootstrap [**60**] como código abierto. En febrero del 2012, se convirtió en el proyecto de desarrollo más popular de GitHub.

El framework es compatible con la mayoría de los navegadores web, la información básica de compatibilidad de sitios web o aplicaciones está disponible para todos los dispositivos y navegadores. Existe un concepto de compatibilidad parcial que hace disponible la información básica de un sitio web para todos los dispositivos y navegadores. Por ejemplo, las propiedades introducidas en CSS3 para las esquinas redondeadas, gradientes y sombras son usadas por Bootstrap a pesar de la falta de soporte de navegadores antiguos. Esto extiende la funcionalidad de la herramienta, pero no es requerida para su uso.

Desde la versión 2.0 también soporta diseños sensibles. Esto significa que el diseño gráfico de la página se ajusta dinámicamente, tomando en cuenta las características del dispositivo usado (Computadoras, tabletas, teléfonos móviles).

Bootstrap es modular y consiste esencialmente en una serie de hojas de estilo LESS (del inglés, stylesheet language) que implementan la variedad de componentes de la herramienta. Una hoja de estilo llamada bootstrap.less incluye los componentes de las hojas de estilo, se puede adaptar el mismo archivo de Bootstrap, seleccionando los componentes que deseen usar en cada proyecto.

Los ajustes son posibles en una medida limitada a través de una hoja de estilo de configuración central. Los cambios más profundos son posibles mediante las declaraciones LESS. El uso del lenguaje de hojas de estilo LESS permite el uso de variables, funciones y operadores, selectores anidados, así como clases mixin (clases que ofrecen cierta funcionalidad para ser heredada por una subclase, pero no están ideadas para ser autónomas.).

El framework viene con una disposición de cuadrilla estándar de 940 píxeles de ancho, alternativamente, el desarrollador puede usar un diseño de ancho variable, para ambos casos, la herramienta tiene cuatro variaciones para hacer uso de distintas resoluciones y tipos de dispositivos: teléfonos móviles, formato de retrato y paisaje, tabletas y computadoras con baja y alta resolución. Esto ajusta el ancho de las columnas automáticamente.

Bootstrap proporciona un conjunto de hojas de estilo que proveen definiciones básicas de estilo para todos los componentes de HTML. Esto otorga una uniformidad al navegador y al sistema de anchura, da una apariencia moderna para el formateo de los elementos de texto, tablas y formularios.

En adición a los elementos regulares de HTML, Bootstrap contiene otra interfaz de elementos comúnmente usados. Ésta incluye botones con características avanzadas (por ejemplo, grupo de botones o botones con opción de menú desplegable, listas de navegación, etiquetas horizontales y

verticales, ruta de navegación, paginación, etc.), etiquetas, capacidades avanzadas de miniaturas tipográficas, formatos para mensajes de alerta y barras de progreso.

Los componentes de JavaScript para Bootstrap están basados en la librería jQuery de JavaScript, los plug-ins se encuentran en la herramienta de plug-in de jQuery. Proveen elementos adicionales de interfaz de usuario como diálogos, tooltips y carruseles. También extienden la funcionalidad de algunos elementos de interfaz existentes, incluyendo por ejemplo una función de autocompletar para campos de entrada.

Para usar Bootstrap en una página HTML, el desarrollador solo debe descargar la hoja de estilo CSS y enlazarla en el archivo HTML. Luego basta con adaptar la maquetación de la página a la estructura propuesta por el framework

JavaScript

JavaScript es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas, aunque existe una forma de JavaScript del lado del servidor SSJS (del inglés, Server-side JavaScript). Su uso en aplicaciones externas a la web, por ejemplo en documentos PDF, aplicaciones de escritorio (mayoritariamente widgets) es también significativo.

JavaScript se diseñó con una sintaxis similar al C, aunque adopta nombres y convenciones del lenguaje de programación Java. Sin embargo Java y JavaScript no están relacionados y tienen semánticas y propósitos diferentes.

Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del DOM (del inglés, Document Object Model).

JavaScript fue desarrollado originalmente por Brendan Eich de Netscape con el nombre de Mocha, el cual fue renombrado posteriormente a LiveScript, para finalmente quedar como JavaScript. El cambio de nombre coincidió aproximadamente con el momento en que Netscape agregó soporte para la tecnología Java en su navegador web Netscape Navigator en la versión 2.002 en diciembre de 1995. La denominación produjo confusión, dando la impresión de que el lenguaje es una prolongación de Java, y se ha caracterizado por muchos como una estrategia de mercadotecnia de Netscape para obtener prestigio e innovar en lo que eran los nuevos lenguajes de programación web. JAVASCRIPT es una marca registrada de Oracle Corporation. Es usada con licencia por los productos creados por Netscape Communications y entidades actuales como la Fundación Mozilla. Microsoft dio como nombre a su dialecto de JavaScript JScript, para evitar problemas relacionados con la marca. JScript fue adoptado en la versión 3.0 de Internet Explorer, liberado en agosto de 1996, e incluyó compatibilidad con el Efecto 2000 en las funciones de fecha. Los dialectos pueden parecer tan similares que los términos JavaScript y JScript a menudo se utilizan indistintamente, pero la especificación de JScript es incompatible con la de ECMA en muchos aspectos.

Para evitar estas incompatibilidades, el W3C diseñó el estándar DOM, que incorporan Konqueror, las versiones 6 de Internet Explorer y Netscape Navigator, Opera la versión 7, Mozilla Application Suite y Mozilla Firefox desde su primera versión.

En 1997 los autores propusieron JavaScript para que fuera adoptado como estándar de la European Computer Manufacturers Association ECMA. En junio de 1997 fue adoptado como un estándar ECMA, con el nombre de ECMAScript [**61**]. Poco después también como un estándar ISO 16262:2011.

El uso más común de JavaScript es escribir funciones embebidas o incluidas en páginas HTML, que interactúan con el DOM de la página permitiendo cargar nuevo contenido para la página o enviar datos al servidor a través de AJAX sin necesidad de recargar la página, también se utiliza para incluir animación en los elementos de la página, hacerlos desaparecer, cambiar su tamaño y moverlos. Permite la creación de contenido interactivo, como juegos y reproducción de audio y vídeo. Mediante JavaScript, también, se puede implementar la validación de los valores de entrada de un formulario web para asegurarse de que son aceptables antes de ser enviado al servidor.

Dado que el código JavaScript puede ejecutarse localmente en el navegador del usuario, el navegador puede responder a las acciones del usuario con rapidez, haciendo una aplicación más sensible. Por otra parte, el código JavaScript puede detectar acciones de los usuarios que HTML por sí solo no puede, como por ejemplo las pulsaciones de teclado, permitiendo a las aplicaciones aprovechar esto para mejorar la interfaz de usuario.

Un motor de JavaScript (también conocido como intérprete de JavaScript o implementación JavaScript) interpreta el código fuente de JavaScript y ejecuta la secuencia de comandos en consecuencia. El primer motor de JavaScript fue creado por Brendan Eich en Netscape Communications Corporation, para el navegador web Netscape Navigator. El motor, denominado SpiderMonkey, está implementado en C. Desde entonces, ha sido actualizado para cumplir con el ECMA-262. El motor Rhino, creado principalmente por Norris Boyd (Google) es una implementación de JavaScript en Java. Rhino, como SpiderMonkey, es compatible con el ECMA-262 edición 3.

Un navegador web es el entorno de ejecución más común para JavaScript. Los navegadores web suelen crear objetos no nativos, dependientes del entorno de ejecución, para representar el DOM. El servidor web es otro entorno común de servicios. Un servidor web JavaScript suele exponer sus propios objetos para representar objetos de petición y respuesta HTTP, que un programa JavaScript podría entonces interrogar y manipular para generar dinámicamente páginas web.

Debido a que JavaScript es el único lenguaje por el que los navegadores web más populares comparten su apoyo, se ha convertido en un lenguaje al que muchos frameworks en otros lenguajes compilan, a pesar de que JavaScript no fue diseñado para tales propósitos. A pesar de las limitaciones de rendimiento inherentes a su naturaleza dinámica, el aumento de la velocidad de los motores de JavaScript ha hecho de este lenguaje un entorno factible para la compilación.

AJAX (del inglés, Asynchronous JavaScript And XML), es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (del inglés, Rich Internet Applications). Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, mejorando la interactividad, velocidad y usabilidad en las aplicaciones.

Es una tecnología asíncrona, en el sentido de que los datos adicionales se solicitan al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página. JavaScript es el lenguaje interpretado en el que normalmente se efectúan las funciones de llamada de AJAX mientras que el acceso a los datos se realiza mediante XMLHttpRequest, objeto disponible en los navegadores actuales. En cualquier caso, no es necesario que el contenido asíncrono esté formateado en XML. Esta es una técnica válida para múltiples plataformas y utilizable en muchos sistemas operativos y navegadores, dado que está basado en estándares abiertos como JavaScript y DOM.

AJAX es una combinación de cuatro tecnologías ya existentes XHTML o HTML y CSS para la información y el diseño que la acompaña, DOM accedido con un lenguaje de scripting por parte del usuario, para mostrar e interactuar dinámicamente con la información presentada, el objeto XMLHttpRequest para intercambiar datos de forma asíncrona con el servidor web y XML como el formato usado generalmente para la transferencia de datos solicitados al servidor, aunque cualquier formato puede funcionar, incluyendo HTML preformateado, texto plano, JSon, etc.

Java

Java [**62**] es un lenguaje de programación de propósito general, concurrente, orientado a objetos, que fue diseñado específicamente para tener la menor cantidad posible de dependencias de implementación. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo siguiendo el principio "write once, run anywhere", lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra.

El lenguaje de programación Java fue originalmente desarrollado por James Gosling de Sun Microsystems, adquirida por la compañía Oracle, y publicado en 1995 como un componente fundamental de la plataforma Java de Sun Microsystems. Su sintaxis deriva en gran medida de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de Java son generalmente compiladas a bytecode (clase Java) que puede ejecutarse en cualquier máquina virtual Java, JVM (del inglés, Java Virtual Machine), sin importar la arquitectura del hardware subyacente.

La compañía Sun desarrolló la implementación de referencia original para los compiladores de Java, máquinas virtuales, y librerías de clases en 1991 y las publicó por primera vez en 1995. A partir de mayo de 2007, en cumplimiento con las especificaciones del Proceso de la Comunidad Java, Sun volvió a licenciar la mayoría de sus tecnologías de Java bajo la Licencia Pública General de GNU. Otros también han desarrollado implementaciones alternas a estas tecnologías de Sun, tales como el Compilador de Java de GNU y el GNU Classpath.

Java se creó como una herramienta de programación para ser usada en un proyecto de set-top-box en una pequeña operación denominada the Green Project en Sun Microsystems en el año 1991. El equipo (Green Team), compuesto por trece personas y dirigido por James Gosling, trabajó durante 18 meses en Sand Hill Road en Menlo Park en su desarrollo. El lenguaje se denominó inicialmente Oak (por un roble que había fuera de la oficina de Gosling), luego pasó a denominarse Green tras descubrir que Oak era ya una marca comercial registrada para adaptadores de tarjetas gráficas y finalmente se renombró a Java.

Los objetivos de Gosling eran implementar una máquina virtual y un lenguaje con una estructura y sintaxis similar a C++. Entre junio y julio de 1994, tras una sesión maratoniana de tres días entre John Gaga, James Gosling, Patrick Naughton, Wayne Rosing y Eric Schmidt, el equipo reorientó la plataforma hacia la Web. Sintieron que la llegada del navegador web Mosaic, propiciaría que Internet se convirtiese en un medio interactivo, como el que pensaban era la televisión por cable. Naughton creó entonces un prototipo de navegador, WebRunner, que más tarde sería conocido como HotJava.

En 1994, se les hizo una demostración de HotJava y la plataforma Java a los ejecutivos de Sun. Java 1.0 pudo descargarse por primera vez en 1994, pero hubo que esperar al 23 de mayo de 1995, durante las conferencias de SunWorld, a que vieran la luz pública Java y HotJava, el navegador Web. El acontecimiento fue anunciado por John Gage, el Director Científico de Sun Microsystems. El acto estuvo acompañado por una pequeña sorpresa adicional, el anuncio por parte de Marc Andreessen, Vicepresidente Ejecutivo de Netscape, de que Java sería soportado en sus navegadores. El 9 de enero del año siguiente, 1996, Sun fundó el grupo empresarial JavaSoft para que se encargase del desarrollo tecnológico. Dos semanas más tarde la primera versión de Java fue publicada.

La promesa inicial de Gosling era Write Once, Run Anywhere (Escríbelo una vez, ejecútalo en cualquier lugar), proporcionando un lenguaje independiente de la plataforma y un entorno de ejecución (la JVM) ligero y gratuito para las plataformas más populares de forma que los binarios

(bytecode) de las aplicaciones Java pudiesen ejecutarse en cualquier plataforma. El entorno de ejecución era relativamente seguro y los principales navegadores web pronto incorporaron la posibilidad de ejecutar applets Java incrustadas en las páginas web.

Java ha experimentado numerosos cambios desde la versión primigenia, JDK 1.0, así como un enorme incremento en el número de clases y paquetes que componen la biblioteca estándar. Desde J2SE 1.4, la evolución del lenguaje ha sido regulada por el JCP (del inglés, Java Community Process), que usa Java JSRs (del inglés, Java Specification Requests) para proponer y especificar cambios en la plataforma Java. El lenguaje en sí mismo está especificado en la JLS (del inglés, Java Language Specification), Los cambios en los JLS son gestionados en JSR 901.

El lenguaje Java se creó con cinco objetivos principales, debía usar el paradigma de la programación orientada a objetos, debería permitir la ejecución de un mismo programa en múltiples sistemas operativos, debería incluir por defecto, soporte para trabajo en red, debería diseñarse para ejecutar código en sistemas remotos de forma segura y debería ser fácil de usar y tomar lo mejor de otros lenguajes orientados a objetos, como C++.

La primera característica, la orientación a objetos (“OO”) ofrece una base más estable para el diseño de un sistema software, permitiendo que grandes proyectos sean fáciles de gestionar y manejar, mejorando como consecuencia su calidad y reduciendo el número de proyectos fallidos. Por otro lado la orientación a objetos fomenta la reutilización del software entre proyectos, una de las premisas fundamentales de la Ingeniería del Software.

La segunda característica, la independencia de la plataforma, significa que programas escritos en el lenguaje Java pueden ejecutarse igualmente en cualquier tipo de hardware, tal como reza el axioma de Java, "write once, run anywhere". Para ello, se compila el código fuente escrito en lenguaje Java, para generar un código conocido como bytecode, instrucciones máquina específicas de la plataforma Java, que están entre el código fuente y el código máquina que entiende el dispositivo destino. El bytecode es ejecutado entonces en la JVM, un programa escrito en código nativo de la plataforma destino, que interpreta y ejecuta el código. Además, se suministran bibliotecas adicionales para acceder a las características de cada dispositivo (como los gráficos, ejecución mediante threads, la interfaz de red, etc) de forma unificada. Se debe tener presente que, aunque hay una etapa explícita de compilación, el bytecode generado es interpretado o convertido a instrucciones máquina del código nativo por el compilador JIT (del inglés, Just In Time). Existen implementaciones del compilador de Java que convierten el código fuente directamente en código objeto nativo, como GCJ. Esto elimina la etapa intermedia donde se genera el bytecode, pero la salida de este tipo de compiladores sólo puede ejecutarse en un tipo de arquitectura.

La compilación JIT, convierte el bytecode a código nativo cuando se ejecuta la aplicación. Otras máquinas virtuales más sofisticadas usan una recompilación dinámica en la que la JVM es capaz de analizar el comportamiento del programa en ejecución y recompila y optimiza las partes críticas. La recompilación dinámica puede lograr mayor grado de optimización que la compilación tradicional o estática, ya que puede basar su trabajo en el conocimiento que de primera mano tiene sobre el entorno de ejecución y el conjunto de clases cargadas en memoria. La compilación JIT y la recompilación dinámica permiten a los programas Java aprovechar la velocidad de ejecución del código nativo sin por ello perder la ventaja de la portabilidad en ambos.

El concepto de independencia de la plataforma de Java cuenta con un gran éxito en las aplicaciones que se ejecutan en un entorno de servidor, como los Servicios Web, los Servlets, los Java Beans, así como en sistemas empotrados basados en OSGi, usando entornos Java empotrados.

En Java, el problema de fugas de memoria, se evita en gran medida gracias a la recolección de basura o automatic garbage collector. El programador determina cuándo se crean los objetos y el entorno en tiempo de ejecución de Java JRE (del inglés, Java Runtime Enviroment) es el responsable de gestionar el ciclo de vida de los objetos. El programa, u otros objetos pueden tener localizado un objeto mediante una referencia a éste. Cuando no quedan referencias a un objeto, el recolector de basura de Java borra el objeto, liberando así la memoria que ocupaba previniendo posibles fugas, el recolector de basura de Java permite una fácil creación y eliminación de objetos y mayor seguridad.

El diseño de Java, su robustez, el respaldo de la industria y su fácil portabilidad han hecho de Java uno de los lenguajes con un mayor crecimiento y amplitud de uso en distintos ámbitos de la industria de la informática. Esto se debe en gran medida a la creación de implementaciones específicas de la plataforma para cubrir necesidades propias de distintos entornos de ejecución, desde aplicaciones para móviles hasta aplicaciones empresariales, ejecutándose en grandes servidores.

La especificación J2ME (del inglés, Java 2 Platform, Micro Edition), una versión del entorno de ejecución Java reducido y altamente optimizado, especialmente desarrollada para el mercado de dispositivos electrónicos de consumo ha producido toda una revolución en lo que a la extensión de Java se refiere. Es posible encontrar microprocesadores diseñados para ejecutar bytecode Java y software Java para tarjetas inteligentes (JavaCard), teléfonos móviles, buscapersonas, set-top-boxes, sintonizadores de TV y otros pequeños electrodomésticos. El modelo de desarrollo de estas aplicaciones es muy semejante a las applets de los navegadores salvo que en este caso se denominan MIDlets.

Desde la primera versión de java existe la posibilidad de desarrollar pequeñas aplicaciones (Applets) en Java que luego pueden ser incrustadas en una página HTML para que sean descargadas y ejecutadas por el navegador web. Estas mini-aplicaciones se ejecutan en una JVM que el navegador tiene configurada como extensión (plug-in) en un contexto de seguridad restringido configurable para impedir la ejecución local de código potencialmente malicioso.

En la parte del servidor, Java es más popular que nunca, desde la aparición de la especificación de Servlets y JSP (Java Server Pages), hasta entonces, las aplicaciones web dinámicas de servidor que existían se basaban fundamentalmente en componentes CGI y lenguajes interpretados. Ambos tenían diversos inconvenientes. Los servlets y las JSPs supusieron un importante avance ya que cuentan con una API de programación muy sencilla, flexible y extensible.

Los servlets no son procesos independientes (como los CGIs) y por tanto se ejecutan dentro del mismo proceso que la JVM mejorando notablemente el rendimiento y reduciendo la carga computacional y de memoria requeridas.

Las JSPs son páginas que se compilan dinámicamente (o se pre-compilan previamente a su distribución) de modo que el código que se consigue tiene una ventaja en rendimiento substancial frente a muchos lenguajes interpretados.

La especificación de Servlets y JSPs define un API de programación y los requisitos para un contenedor (servidor) dentro del cual se puedan desplegar estos componentes para formar

aplicaciones web dinámicas completas. Hoy día existen multitud de contenedores, libres y comerciales, compatibles con estas especificaciones.

A partir de su expansión entre la comunidad de desarrolladores, estas tecnologías han dado paso a modelos de desarrollo mucho más elaborados con frameworks que se sobreponen sobre los servlets y las JSPs para conseguir un entorno de trabajo mucho más poderoso y segmentado en el que la especialización de roles sea posible y se facilite la reutilización y robustez de código. A pesar de todo ello, las tecnologías que subyacen (Servlets y JSPs) son substancialmente las mismas. Convirtiendo este modelo de trabajo en uno de los estándar de-facto para el desarrollo de aplicaciones web dinámicas de servidor.

Hoy en día existen multitud de aplicaciones gráficas de usuario basadas en Java. El entorno de ejecución Java (JRE) se ha convertido en un componente habitual en los PC de usuario de los sistemas operativos más usados en el mundo. Además, muchas aplicaciones Java lo incluyen dentro del propio paquete de la aplicación de modo que se ejecuten en cualquier PC.

En las primeras versiones de la plataforma Java existían importantes limitaciones en las APIs de desarrollo gráfico (AWT). Desde la aparición de la biblioteca Swing la situación mejoró substancialmente y posteriormente con la aparición de bibliotecas como SWT hacen que el desarrollo de aplicaciones de escritorio complejas y con gran dinamismo, usabilidad, etc. sea relativamente sencillo.

El JRE es el software necesario para ejecutar cualquier aplicación desarrollada para la plataforma Java. El usuario final usa el JRE como parte de paquetes software o plugins (o conectores) en un navegador Web. Se ofrece también el SDK de Java, o JDK (del inglés, Java Development Kit) en cuyo seno reside el JRE, e incluye herramientas como el compilador de Java, Javadoc para generar documentación o el depurador. Puede también obtenerse como un paquete independiente, y puede considerarse como el entorno necesario para ejecutar una aplicación Java, mientras que un desarrollador debe además contar con otras facilidades que ofrece el JDK.

Java incluye bibliotecas que ofrecen apoyo para el desarrollo. Algunos ejemplos de estas, son las bibliotecas centrales, que incluyen una colección de bibliotecas para implementar estructuras de datos como listas, arrays, árboles y conjuntos. También se proporcionan bibliotecas para análisis de XML, para seguridad, para internacionalización y localización, para integración, que permiten la comunicación con sistemas externos como por ejemplo la API para acceso a bases de datos JDBC (del inglés, Java DataBase Conectivity), la interfaz JNDI (del inglés, Java Naming and Directory Interface) para servicios de directorio, RMI (del inglés, Remote Method Invocation) y CORBA para el desarrollo de aplicaciones distribuidas. A su vez se incluyen bibliotecas para la interfaz de usuario como el conjunto de herramientas nativas AWT (del inglés, Abstract Window Toolkit), que ofrece componentes GUI, mecanismos para usarlos y manejar sus eventos asociados, las Bibliotecas de Swing, construidas sobre AWT que ofrecen implementaciones no nativas de los componentes de AWT, y APIs para la captura, procesamiento y reproducción de audio.

Java se define en tres plataformas en un intento por cubrir distintos entornos de aplicación. Java ME (del inglés, Java Platform Micro Edition), orientada a entornos de limitados recursos, como teléfonos móviles, PDAs (Personal Digital Assistant), etc. Java SE (del inglés, Standard Edition), para entornos de gama media y estaciones de trabajo y Java EE (del inglés, Enterprise Edition), orientada a entornos distribuidos empresariales o de Internet.

Spring Framework

Spring es un framework [**63**] para el desarrollo de aplicaciones y contenedor de inversión de control, de código abierto para la plataforma Java. La primera versión fue escrita por Rod Johnson, quien lo lanzó junto a la publicación de su libro Expert One-on-One J2EE Design and Development [**64**] El framework fue lanzado inicialmente bajo la licencia Apache 2.0 en junio de 2003. El primer gran lanzamiento fue la versión 1.0, que apareció en marzo de 2004 y fue seguida por otros hitos en septiembre de 2004 y marzo de 2005. La versión 1.2.6 de Spring Framework obtuvo reconocimientos Jolt Awards y Jax Innovation Awards en 2006. Spring Framework 2.0 fue lanzada en 2006, la versión 2.5 en noviembre de 2007, Spring 3.0 en diciembre de 2009, y Spring 3.1 dos años más tarde. El inicio del desarrollo de la versión 4.0 fue anunciado en enero de 2013.

Si bien las características fundamentales de Spring Framework pueden ser usadas en cualquier aplicación desarrollada en Java, existen variadas extensiones para la construcción de aplicaciones web sobre la plataforma Java EE. A pesar de que no impone ningún modelo de programación en particular, este framework se ha vuelto popular en la comunidad al ser considerado una alternativa, sustituto, e incluso un complemento al modelo EJB (del inglés, Enterprise JavaBean).

Spring Framework hizo que aquellas técnicas que resultaban desconocidas para la mayoría de programadores se volvieran populares en un periodo muy corto de tiempo. El ejemplo más notable es la inversión de control. En el año 2004, Spring disfrutó de unas altísimas tasas de adopción y al ofrecer su propio framework de programación orientada a aspectos AOP (del inglés, aspect-oriented programming) consiguió hacer más popular su paradigma de programación en la comunidad Java.

Spring Framework comprende diversos módulos que proveen un rango de servicios, el principal es el contenedor de inversión de control que permite la configuración de los componentes de aplicación y la administración del ciclo de vida de los objetos Java, se lleva a cabo principalmente a través de la inyección de dependencias.

Spring está estructurado en módulos configurables, los principales incluyen: El módulo de programación orientada a aspectos, que habilita la implementación de rutinas transversales, el módulo de acceso a datos que permite trabajar con RDBMS (del inglés, Relational DataBase Management System) en la plataforma java, usando JDBC y herramientas de mapeo objeto relacional con bases de datos NoSQL, el módulo de gestión de transacciones que unifica distintas APIs de gestión y coordina las transacciones para los objetos Java, el módulo de MVC (del inglés, Model View Controller) que aporta un framework basado en HTTP y servlets, que brinda herramientas para la extensión y personalización de aplicaciones web y servicios web REST, un módulo de acceso remoto que permite la importación y exportación estilo RPC, de objetos Java a través de redes que soporten RMI, CORBA y protocolos basados en HTTP incluyendo servicios web (SOAP), el módulo de procesamiento por lotes que permite la definición de procesamientos de tipo batch, el módulo de autenticación y autorización (spring security), que brinda procesos de seguridad configurables que soportan un rango de estándares, protocolos, herramientas y prácticas, el módulo de administración remota que permite la configuración de la visibilidad y la gestión de objetos Java de manera local o remota vía JMX, el módulo de mensajes que da soporte a distintas APIs de mensajería y el módulo de testing que brinda clases utiles para el desarrollo de test de unidades e integración.

Hibernate

Hibernate [**65**] es una herramienta de Mapeo objeto-relacional ORM (del inglés, Object Relational Mapping) para la plataforma Java, que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos XML o anotaciones en los beans de las entidades que permiten establecer estas relaciones. Es software libre, distribuido bajo los términos de la licencia GNU LGPL.

Como todas las herramientas de su tipo, Hibernate busca solucionar el problema de la diferencia entre los dos modelos de datos coexistentes en una aplicación, el usado en la memoria de la computadora (orientación a objetos) y el usado en las bases de datos (modelo relacional). Para lograr esto permite al desarrollador detallar cómo es su modelo de datos, qué relaciones existen y qué forma tienen. Con esta información Hibernate le permite a la aplicación manipular los datos en la base de datos operando sobre objetos, con todas las características de la POO. Hibernate convertirá los datos entre los tipos utilizados por Java y los definidos por SQL, genera las sentencias SQL y libera al desarrollador del manejo manual de los datos que resultan de la ejecución de dichas sentencias, manteniendo la portabilidad entre todos los motores de bases de datos con un ligero incremento en el tiempo de ejecución.

Hibernate está diseñado para ser flexible en cuanto al esquema de tablas utilizado, para poder adaptarse a su uso sobre una base de datos ya existente. También tiene la funcionalidad de crear la base de datos a partir de la información disponible. Ofrece también un lenguaje de consulta de datos llamado HQL (del inglés, Hibernate Query Language), al mismo tiempo que una API para construir las consultas programáticamente, conocida como criteria. La implementación para Java puede ser utilizada en aplicaciones Java independientes o en aplicaciones Java EE, mediante el componente Hibernate Annotations que implementa el estándar JPA, que es parte de esta plataforma.

Hibernate fue una iniciativa de un grupo de desarrolladores dispersos alrededor del mundo conducidos por Gavin King. Tiempo después, JBoss Inc. (empresa comprada por Red Hat) contrató a los principales desarrolladores de Hibernate y trabajó con ellos en brindar soporte al proyecto.

SQL

El lenguae de consulta estructurado o SQL (del inglés, Structured Query Language) es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en ellas. Una de sus características es el manejo del álgebra y el cálculo relacional que permiten efectuar consultas con el fin de recuperar de forma sencilla información de interés de bases de datos, así como hacer cambios en ellas.

Los orígenes del SQL están ligados a los de las bases de datos relacionales. En 1970 E. F. Codd propone el modelo relacional y asociado a este un sublenguaje de acceso a los datos basado en el cálculo de predicados. Basándose en estas ideas, los laboratorios de IBM definieron el lenguaje SEQUEL (del inglés, Structured English Query Language) que más tarde fue ampliamente implementado por el sistema de gestión de bases de datos experimental System R, desarrollado en 1977 también por IBM. Sin embargo, fue Oracle quien lo introdujo por primera vez en 1979 en un producto comercial. El SEQUEL terminó siendo el predecesor de SQL, que es una versión evolucionada del primero. El SQL pasa a ser el lenguaje por excelencia de los diversos sistemas de gestión de bases de datos relacionales surgidos en los años siguientes y fue por fin estandarizado en 1986 por el ANSI, dando lugar a la primera versión estándar de este lenguaje, el "SQL-86" o "SQL1". Al año siguiente este estándar es también adoptado por la ISO.

Sin embargo, este primer estándar no cubría todas las necesidades de los desarrolladores e incluía funcionalidades de definición de almacenamiento que se consideró suprimirlas. Así que, en 1992, se lanzó un nuevo estándar ampliado y revisado del SQL llamado "SQL-92" o "SQL2".

En la actualidad el SQL es el estándar de-facto de la inmensa mayoría de los DBMS comerciales. Y, aunque la diversidad de añadidos particulares que incluyen las distintas implementaciones comerciales del lenguaje es amplia, el soporte al estándar SQL-92 es general y muy amplio.

El ANSI SQL sufrió varias revisiones y agregados a lo largo del tiempo, en 1986, SQL-86 /SQL-87 fue la primera publicación hecha por ANSI. Confirmada por ISO en 1987, en 1989, SQL-89 consistió en una revisión menor. En 1992, SQL-92 / SQL2, fue una revisión mayor. En 1999, SQL: 1999 / SQL2000, agrego expresiones regulares, consultas recursivas (para relaciones jerárquicas), triggers y algunas características orientadas a objetos. En 2003, SQL: 2003, introduce algunas características de XML, cambios en las funciones, estandarización del objeto sequence y de las columnas autonumericas. En 2005, SQL: 2005, normalizado por ISO/IEC 9075-14:2005 define las maneras en las cuales el SQL se puede utilizar conjuntamente con XML. Define maneras de importar y guardar datos XML en una base de datos SQL, manipulándolos dentro de la base de datos y publicando el XML y los datos SQL convencionales en forma XML. Además, proporciona facilidades que permiten a las aplicaciones integrar dentro de su código SQL el uso de XQuery, lenguaje de consulta XML publicado por el W3C para acceso concurrente a datos ordinarios SQL y documentos XML. En 2008, SQL: 2008, permite el uso de la cláusula ORDER BY fuera de las definiciones de los cursores. Incluye los disparadores del tipo INSTEAD OF. Añade la sentencia TRUNCATE.

PostgreSQL

PostgreSQL [**66**] es un DBMS (del inglés, DataBase Managment System) relacional orientado a objetos y libre, publicado bajo la licencia BSD. Como muchos otros proyectos de código abierto, el desarrollo de PostgreSQL no es manejado por una empresa y/o persona, sino que es dirigido por una comunidad de desarrolladores que trabajan de forma desinteresada, altruista, libre y/o apoyada por organizaciones comerciales. Dicha comunidad es denominada el PGDG (del inglés, PostgreSQL Global Development Group).

PostgreSQL ha tenido una larga evolución, la cual se inicia en 1982 con el proyecto Ingres en la Universidad de Berkeley. Este proyecto, liderado por Michael Stonebraker, fue uno de los primeros intentos en implementar un motor de base de datos relacional. Después de haber trabajado un largo tiempo en Ingres y de haber tenido una experiencia comercial con el mismo, Michael decidió volver a la Universidad en 1985 para trabajar en un nuevo proyecto sobre la experiencia de Ingres, dicho proyecto fue llamado post-ingres o simplemente POSTGRES.

El proyecto post-ingres pretendía resolver los problemas con el modelo de base de datos relacional que habían sido aclarados a comienzos de los años 1980. El principal de estos problemas era la incapacidad del modelo relacional de comprender "tipos", es decir, combinaciones de datos simples que conforman una única unidad. Actualmente estos son llamados objetos. Se esforzaron en introducir la menor cantidad posible de funcionalidades para completar el soporte de tipos. Estas funcionalidades incluían la habilidad de definir tipos, pero también la habilidad de describir relaciones, las cuales hasta ese momento eran ampliamente utilizadas pero mantenidas completamente por el usuario. En Postgres la base de datos comprendía las relaciones y podía obtener información de tablas relacionadas utilizando reglas. Postgres usó muchas ideas de Ingres pero no su código.

En 1986 se publicaron varios papers que describían las bases del sistema, en 1988: ya se contaba con una versión utilizable. En 1989 el grupo publicaba la versión 1 para una pequeña comunidad de usuarios, en 1990: se publicaba la versión 2 la cual tenía prácticamente reescrito el sistema de reglas, 1991 se publicó de la versión 3 que añadía la capacidad de múltiples motores de almacenamiento. En 1993 se da un crecimiento importante de la comunidad de usuarios, la cual demandaba más características. En1994 después de la publicación de la versión 4, el proyecto terminó y el grupo se disolvió. Después de que el proyecto POSTGRES terminara, dos graduados de la universidad, Andrew Yu y Jolly Chen, comenzaron a trabajar sobre el código de POSTGRES, esto fue posible dado que POSTGRES estaba licenciado bajo la BSD, y lo primero que hicieron fue añadir soporte para el lenguaje SQL a POSTGRES, dado que anteriormente contaba con un intérprete del lenguaje de consultas QUEL (basado en Ingres), creando así el sistema al cual denominaron Postgres95.

Para el año 1996 se unieron al proyecto personas ajenas a la Universidad como Marc Fournier de Hub.Org Networking Services, Bruce Momjian y Vadim B. Mikheev quienes proporcionaron el primer servidor de desarrollo no universitario para el esfuerzo de desarrollo de código abierto y comenzaron a trabajar para estabilizar el código de Postgres95. En el año 1996 decidieron cambiar el nombre de Postgres95 de tal modo que refleje la característica del lenguaje SQL y lo terminaron llamando PostgreSQL, cuya primera versión de código abierto fue lanzada el 1 de agosto de 1996. La primera versión formal de PostgreSQL (6.0) fue liberada en enero de 1997. Desde entonces, muchos desarrolladores entusiastas de los motores de base de datos se unieron al proyecto, coordinaron vía Internet y entre todos comenzaron a incorporar muchas características al motor.

En 2000, ex inversionistas de Red Hat crearon la empresa Great Bridge para comercializar PostgreSQL y competir contra proveedores comerciales de bases de datos. Great Bridge auspició a varios desarrolladores de PostgreSQL y donó recursos de vuelta a la comunidad, pero a fines de 2001 cerró debido a la dura competencia de compañías como Red Hat y pobres condiciones del mercado.

En 2001, Command Prompt, Inc. lanzó Mammonth PostgreSQL, la más antigua distribución comercial de PostgreSQL. Continúa brindando soporte a la comunidad PostgreSQL a través del auspicio de desarrolladores y proyectos, incluyendo PL/Perl, PL/php y el alojamiento de proyectos de comunidades como PostgreSQL Build Farm.

En agosto de 2007 EnterpriseDB anunció el Postgres Resource Center y EnterpriseDB Postgres, diseñados para ser una completamente configurada distribución de PostgreSQL incluyendo muchos módulos contribuidos y agregados. EnterpriseDB Postgres fue renombrado Postgres Plus en marzo de 2008.

El proyecto PostgreSQL continúa haciendo lanzamientos principales anualmente y lanzamientos menores de reparación de bugs, todos disponibles bajo la licencia BSD, y basados en contribuciones de proveedores comerciales, empresas aportantes y programadores de código abierto mayormente.

Algunas de sus principales características son, entre otras, alta concurrencia, una amplia variedad de tipos nativos,

Mediante un sistema denominado MVCC (Acceso concurrente multiversión, por sus siglas en inglés) PostgreSQL permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos. Cada usuario obtiene una visión consistente de lo último a lo que se le hizo commit. Esta estrategia es superior al uso de bloqueos por tabla o por filas común en otras bases, eliminando la necesidad del uso de bloqueos explícitos.

PostgreSQL provee nativamente soporte para los siguientes tipos de datos: números de precisión arbitraria, texto de largo ilimitado, figuras geométricas (con una variedad de funciones asociadas), direcciones IP (IPv4 e IPv6), bloques de direcciones estilo CIDR, direcciones MAC, arrays. Adicionalmente los usuarios pueden crear sus propios tipos de datos, los que pueden ser por completo indexables gracias a la infraestructura GiST de PostgreSQL. Algunos ejemplos son los tipos de datos GIS creados por el proyecto PostGIS.

Tambien provee soporte para claves ajenas también denominadas Llaves ajenas o Claves Foráneas (foreign keys), disparadores (triggers), vistas, integridad transaccional, herencia de tablas, tipos de datos y operaciones geométricas.

A su vez brinda soporte para transacciones distribuidas que le permiten integrarse en un sistema distribuido formado por varios recursos, gestionado por un servidor de aplicaciones donde el éxito de la transacción global es el resultado del éxito de las transacciones locales.

Permite definir funciones que son bloques de código que se ejecutan en el servidor, pueden ser escritos en varios lenguajes, con la potencia que cada uno de ellos da, desde las operaciones básicas de programación, tales como bifurcaciones y bucles, hasta las complejidades de la programación orientada a objetos o la programación funcional. Ademas de un lenguaje propio llamado PL/PgSQL (similar al PL/SQL de oracle), Algunos de los lenguajes que se pueden usar son los siguientes, C, C++, Java PL/Java web, PL/Perl, plPHP, PL/Python, PL/Ruby, PL/sh, PL/Tcl, PL/Scheme y el lenguaje para aplicaciones estadísticas R por medio de PL/R. PostgreSQL soporta funciones que retornan filas,

donde la salida puede tratarse como un conjunto de valores que pueden ser tratados igual a una fila retornada por una consulta. Las funciones pueden ser definidas para ejecutarse con los derechos del usuario ejecutor o con los derechos de un usuario previamente definido.

Eclipse IDE

Eclipse [**67**] es una plataforma de desarrollo, diseñada para ser extendida de forma indefinida a través de plug-ins. Fue concebida desde sus orígenes para convertirse en una plataforma de integración de herramientas de desarrollo. No tiene en mente un lenguaje específico, sino que es un IDE genérico, aunque goza de mucha popularidad entre la comunidad de desarrolladores del lenguaje Java usando el plug-in JDT que viene incluido en la distribución estándar del IDE.

Proporciona herramientas para la gestión de espacios de trabajo, escribir, desplegar, ejecutar y depurar aplicaciones. Entre sus principales características se pueden destacar las perspectivas, editores y vistas. En Eclipse el concepto de trabajo está basado en las perspectivas, que no es otra cosa que una preconfiguración de ventanas y editores, relacionadas entre sí, y que permiten trabajar en un determinado entorno de trabajo de forma óptima. El desarrollo sobre Eclipse se basa en los proyectos, que son el conjunto de recursos relacionados entre sí, como puede ser el código fuente, documentación, ficheros configuración, árbol de directorios, etc. El IDE proporciona asistentes y ayudas para la creación de proyectos. Eclipse incluye un potente depurador, de uso fácil e intuitivo que ayuda a mejorar el código. Para ello sólo se debe ejecutar el programa en modo depuración. Existe una extensa colección de plug-ins disponibles, unos publicados por Eclipse, otros por terceros. Uno de los más importartent es el plug-in JDT que es el encargado del soporte del IDE al lenguaje Java y está incluido en la versión estándar de Eclipse.

La arquitectura de plug-ins de Eclipse permite, además de integrar diversos lenguajes sobre un mismo IDE, introducir otras aplicaciones accesorias que pueden resultar útiles durante el proceso de desarrollo como: herramientas UML, editores visuales de interfaces, ayuda en línea para librerías, etc.

Los orígenes de Eclipse los encontramos en su antecesor VisualAge de IBM, que desarrolló una máquina virtual dual para Java y Smaltalk. Cuando Java se comenzó a extender, y aumentó su popularidad, IBM decidió abandonar el proyecto de la máquina virtual dual y desarrollar una nueva plataforma basada en dicho lenguaje. De ahí, en el año 2001, nació junto con Borland, la Fundación Eclipse, sin ánimo de lucro, convirtiendo a Eclipse en un proyecto de código abierto bajo licencia Eclipse Public License. Esta fundación se ha ido enriqueciendo con la inclusión de importantes empresas del mundo del desarrollo como son Red Hat, Oracle, HP, etc.

Apache Maven

Maven [**68**] es una herramienta de software para la gestión y construcción de proyectos Java creada por Jason van Zyl, de Sonatype, en 2002. Es similar en funcionalidad a Apache Ant, pero tiene un modelo de configuración de construcción más simple, basado en un formato XML. Estuvo integrado inicialmente dentro del proyecto Jakarta pero ahora ya es un proyecto de nivel superior de la Apache Software Foundation.

Maven utiliza un POM (del inglés, Project Object Model) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado.

Una característica clave de Maven es que está listo para usar en red, el motor incluido en su núcleo puede dinámicamente descargar plugins de un repositorio, el mismo repositorio que provee acceso a muchas versiones de diferentes proyectos Open Source en Java, de Apache y otras organizaciones y desarrolladores. Provee soporte no sólo para obtener archivos de su repositorio, sino también para subir artefactos al repositorio al final de la construcción de la aplicación, dejándola al acceso de todos los usuarios. Una caché local de artefactos actúa como la primera fuente para sincronizar la salida de los proyectos a un sistema local.

Está construido usando una arquitectura basada en plugins que permite que utilice cualquier aplicación controlable a través de la entrada estándar. La idea de reutilización es un aspecto central, y más específicamente, la reutilización de la lógica de construcción. Como los proyectos generalmente se construyen en patrones similares, una elección lógica podría ser reutilizar los procesos de construcción. La principal idea es no reutilizar el código o funcionalidad, sino simplemente cambiar la configuración o también código escrito. Esa es la principal diferencia entre Apache Ant y Apache Maven: el primero es una librería de utilidades y funciones buenas y útiles, mientras que la otra es un framework configurable y extensible

Las partes del ciclo de vida principal del proyecto Maven son: compile, que genera los ficheros .class compilando los fuentes .java, test que ejecuta los test automáticos de JUnit existentes, abortando el proceso si alguno de ellos falla, package que genera el fichero .jar / .war con los .class compilados, install que Copia el fichero .jar/.war a un directorio del ordenador donde maven deja todos los .jar / .war, para que de esta forma esos .jar / .war pueden utilizarse en otros proyectos maven en el mismo ordenador, deploy que copia el fichero .jar / .war a un servidor remoto, poniéndolo disponible para cualquier proyecto maven con acceso a ese servidor remoto.

Cuando se ejecuta cualquiera de los comandos Maven, este irá verificando todas las fases del ciclo de vida desde la primera hasta la del comando, ejecutando sólo aquellas que no se hayan ejecutado previamente.

También existen algunas metas que están fuera del ciclo de vida que pueden ser llamadas, pero Maven asume que estas metas no son parte del ciclo de vida por defecto, es decir que no tienen que ser siempre realizadas. Estas metas son clean, que elimina todos los .class, .jar y .war generados, assembly:assembly que genera un fichero .zip con todo lo necesario para instalar nuestro programa java, site que genera un sitio web con la información del proyecto, site-deploy que sube el sitio web al servidor configurado, etc. estas metas pueden ser añadidas al ciclo de vida a través del POM.

SVN

SVN o Subversion [**69**] es una herramienta de control de versiones open source basada en un repositorio cuyo funcionamiento se asemeja enormemente al de un sistema de ficheros. Es software libre bajo una licencia de tipo Apache/BSD. Utiliza el concepto de revisión para guardar los cambios producidos en el repositorio, entre dos revisiones sólo guarda el conjunto de modificaciones (delta), optimizando así al máximo el uso de espacio en disco. Permite al usuario crear, copiar y borrar carpetas con la misma flexibilidad con la que lo haría si estuviese en su disco duro local. Dada su flexibilidad, es necesaria la aplicación de buenas prácticas para llevar a cabo una correcta gestión de las versiones del software generado.

Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintas computadoras, a cierto nivel, la posibilidad de que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración.

Entre sus principales características se destaca que permite seguir la historia de los archivos y directorios a través de copias y renombrados, que las modificaciones son atómicas, que la creación de ramas y etiquetas es una operación eficiente, que sólo se envían las diferencias en ambas direcciones, que puede ser servido mediante Apache, sobre WebDAV/DeltaV, permitiendo que clientes WebDAV utilicen Subversion de forma transparente. Además maneja eficientemente archivos binarios y permite selectivamente el bloqueo de archivos. Mediante la integracioncion con el servidor Web Apache permite utilizar todas las opciones de autentificación que este provee.

La estructura TTB se ha convertido en el estándar de-facto en los repositorios SVN, TTB son las iniciales de las tres carpetas que compondrán el primer nivel de directorios del repositorio: Trunk, Tags y Branches. Cada carpeta tiene su funcionalidad específica, pero Subversion, al igual que un disco duro, las tratará por igual y no limitará las operaciones a realizar sobre ellas. La carpeta Trunk es, en general donde se ubica la rama de desarrollo principal, la carpeta Tags se utiliza para la gestión de versiones y debería estar reservada para versiones cerradas y la carpeta Branches debería utilizarse para las ramas con evoluciones paralelas al Trunk.

Apache Tomcat

Apache Tomcat (también llamado Jakarta Tomcat o simplemente Tomcat) funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Implementa las especificaciones de servlets y de JavaServer Pages (JSP). Es mantenido y desarrollado por miembros de la Apache Software Foundation y voluntarios independientes, los usuarios disponen de libre acceso a su código fuente y a su forma binaria en los términos establecidos en la Apache Software License. Las primeras distribuciones de Tomcat fueron las versiones 3.0.x. Las versiones más recientes son las 7.x, que implementan las especificaciones de Servlet 3.0 y de JSP 2.2. A partir de la versión 4.0, Jakarta Tomcat utiliza el contenedor de servlets Catalina.

Tomcat es un servidor web con soporte de servlets y JSPs, no es un servidor de aplicaciones, como JBoss o JOnAS. Incluye el compilador Jasper, que compila JSPs convirtiéndolas en servlets. El motor de servlets de Tomcat a menudo se presenta en combinación con el servidor web Apache, Pero puede funcionar como servidor web por sí mismo. En sus inicios existió la percepción de que el uso de forma autónoma era sólo recomendable para entornos de desarrollo y entornos con requisitos mínimos de velocidad y gestión de transacciones. Hoy en día ya no existe esa percepción y Tomcat es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad. Dado que fue escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual Java. La versión 7.x Implementa las especificaciones de Servlet 3.0 JSP 2.2 y EL 2.2

Arquitectura y Diseño de la Aplicación

Esta sección comienza analizando el modelo que se propuso para implementar la herramienta, dando una visión global del mismo, para continuar con una descripción detallada de las capas que componen la aplicación, comenzado desde la interfaz de usuario hasta llegar a la base de datos, se detallan los aspectos técnicos del modelo de base de datos que determinaron la elección de un motor de base de datos específico y se detalla el formato de los archivos soportados por la aplicación. Por último se describen los principales problemas o aspectos críticos que surgieron durante el desarrollo de la aplicación.

Modelo de la Aplicación

LEL to UCP fue diseñada basándose en un modelo compuesto por capas que interactúan entre sí para llevar a cabo las funciones requeridas por la aplicación. Una primera aproximación al modelo propuesto puede verse en la siguiente figura, en donde se enumeran las capas mencionadas y se señala como interactúan entre sí para permitir que la información viaje desde la capa de presentación hasta su representación final en la base de datos y viceversa.

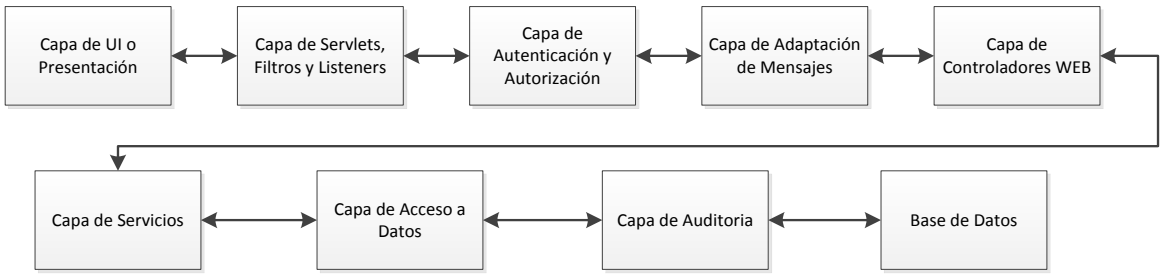

Ilustración 19 - Modelo de Capas de la Herramienta

A grandes rasgos el sistema cuenta con una capa de presentación que se encarga de presentan información y permitir la interacción con el usuario, una capa de servlets, filtros y listeners que acepta peticiones de la capa anterior y a la cual retorna los resultados de dichas peticiones, una capa de autenticación y autorización que se encarga de controlar que usuarios tiene derecho a hacer ciertas operaciones y de mantener la información de sesión de cada usuario del sistema. Luego se ubica una capa de adaptación de mensajes que actúa como traductor de las peticiones que viene desde el cliente y los resultados que se le envían. A continuación encontramos una capa de controladores web, que es la encargada de recibir las peticiones de la capa de presentación y delegar en la capa de servicios su procesamiento, retornando luego los resultados de dicho procesamiento. Seguida de esta se encuentra la capa de servicios en donde se ubica la lógica propia de la aplicación. Luego se encuentra la capa de acceso a datos que es la encargada de facilitarle dicho acceso a la capa de servicios, sirviendo como nexo entre la aplicación y la base de datos. Antes de llegar a la base de datos propiamente dicha se encuentra una capa de auditoria que se encarga de agregar metadatos a la información que se persiste para tener conocimiento de quien genera y/o modifica los datos que se almacenan en la base de datos, la que encontramos al final de nuestro modelo y en donde se almacenan todos los datos generados mediante la herramienta.

A continuación se detallan los componentes principales de cada capa y se analizan sus principales funciones, mecanismos de comunicación y el rol que desempeñan en el funcionamiento de la aplicación, para concluir con una descripción del modelo de datos subyacente y los formatos de archivos a utilizar en la herramienta.

Capa de UI o Presentación

La capa de UI o presentación es la encargada de interactuar con el usuario, de recibir sus indicaciones y de presentar los correspondientes resultados. Su principal característica visual es el diseño tabular que predomina a lo largo de las pantallas de la herramienta. Para maximizar la compatibilidad y optimizar los tiempos de repuesta, esta capa está construida enteramente en HTML, CSS y JavaScript, reduciendo de esta manera el tiempo de carga de las páginas y minimizando el overhead en el servidor.

Esta capa está basada en un modelo de acceso a los recursos de tipo REST (del inglés, Representational State Transfer) que esta implementado mediante la utilización de AJAX en conjunto con JSon, esto quiere decir que los datos se transfieren a las capas inferiores y son recibidos desde dichas capas en formato JSon, lo que facilita su procesamiento y permite realizar las acciones indicadas sin necesidad de recargar la página.

Para dar soporte a la interacción del usuario y a la transferencia de datos hacia y desde las demás capas, la aplicación incorpora JQuery, que brinda un conjunto de funciones que facilitan la programación de peticiones de tipo AJAX, el procesamiento de datos en formato JSon y brindan funciones útiles en el manejo del DOM. También se incorporó en esta capa JQueryUI que es una librería que brindan un conjunto de componentes, funciones y widgets que tienen por objetivo facilitar la presentación de datos y mejorar la interacción con el usuario.

Desde el punto de vista de la presentación de información la herramienta hace uso de Bootstrap, que brinda un conjunto de estilos y componentes de maquetación CSS que permite homogeneizar el diseño de la aplicación haciéndola más amena a la vista y brindando una interacción estándar, similar a la que la mayoría de los usuarios encuentran en otra aplicaciones web.

Con respecto a la interacción con las otras capas del sistema, la herramienta hace uso del envío de peticiones HTTP, mediante AJAX, las que en su mayoría portan datos codificados en formato JSon, cabe la aclaración de que las peticiones simples que involucran datos escalares no hacen uso de dicha codificación a fin de simplificar su implementación y optimizar su procesamiento. Al estar los contenidos de las peticiones codificados como lo indica JSon es responsabilidad de la capa de adaptación de mensajes decodificar dicho contenido, volviéndolo una representación comprensible en el modelo de objetos subyacente.

Capa de Servlets, Filtros y Listeners

Esta capa es la encargada de recibir las peticiones de la capa de UI y dirigir dichas peticiones hasta su destino final, así como también, es responsable de devolver los resultados de las peticiones a la capa superior. Está basada en tecnología estándar de servlets, filtros y listeners definidos por J2EE y en particular este es el punto donde se integran Spring Framework, Spring MVC y Spring Security que son los encargados de dar soporte a las demás capas de la aplicación.

En esta capa se dirigen los pedidos de la UI a los controladores, el encargado de definir qué mensaje debe ser dirigido a que método de que controlador, es el servlet nombrado como "dispatcher" que referencia a la clase "org.springframework.web.servlet.DispatcherServlet" quien guiándose por lo definido en las anotaciones que poseen los controladores, invoca el método correspondiente en el objeto definido por dichas anotaciones. Para que este servlet pueda cumplir con su objetivo se lo define en el descriptor de la aplicación, o web.xml, de la siguiente forma.

```
<context-param>
	<param-name>contextConfigLocation</param-name>
	<param-value>/WEB-INF/classes/applicationContext.xml</param-value>
</context-param>
<!-- Servlet para atender las peticiones de SpringFramework. -->
<servlet>
	<servlet-name>dispatcher</servlet-name>
	<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
	<init-param>
		<param-name>contextConfigLocation</param-name>
		<param-value>/WEB-INF/classes/dispatcher-servlet.xml</param-value>
	</init-param>
	<load-on-startup>1</load-on-startup>
</servlet>
```

Ilustración 20 - Definición de Dispatcher Servlet

Al dirigir un mensaje a su destino, el usuario que lo origino debe tener permiso para ejecutar la acción indicada, la capa encargada de controlar los aspectos de seguridad, es la de autenticación y autorización, y para que dicha capa esté al tanto de las peticiones que debe analizar, se define un filtro correspondiente a Spring Security que es el encargado de delegar las tareas correspondientes a la capa encargada de la seguridad de la herramienta. El filtro utilizado en la aplicación es "springSecurityFilterChain" que referencia a la clase "org.springframework.web.filter.DelegatingFilterProxy" y que define una cadena de filtros por los que el mensaje será conducido hasta llegar a su destino, este filtro como se puede ver en su definición intercepta todas las peticiones a la URL de la aplicación.

```
<filter>
	<filter-name>springSecurityFilterChain</filter-name>
	<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>

<filter-mapping>
	<filter-name>springSecurityFilterChain</filter-name>
	<url-pattern>/*</url-pattern>
</filter-mapping>
```

Ilustración 21 - Definición del Filtro de Seguridad

Además de las dos definiciones previas, esta capa es la encargada de definir el Listener de contexto de Spring, que es el encargado de gestionar el contexto de la aplicación que utilizaran las demás capas.

```
<listener>
        <listener-class>
            org.springframework.web.context.ContextLoaderListener
</listener-class>
</listener>
```

Ilustración 22 - Definición del Listener de Contexto de Spring

El componente principal de esta capa es el "DispatcherServlet" que se encarga de dirigir las peticiones que recibe la aplicación, pero debido a la organización de la arquitectura de J2EE y el diseño del framework Spring, es responsabilidad de esta capa declarar e inicializar los componentes que permite el funcionamiento de las capas que dependen de dicho framework, ejemplo de esto son las definiciones del listener de contexto y el filtro de Spring Security.

Capa de Autenticación y Autorización

La capa de Autenticación y Autorización es la encargada de identificar a los usuarios de la aplicación y de controlar que ejecuten solo las acciones que tiene permitido realizar. Para llevar a cabo esta tarea se basa en Spring Security que proporciona los componentes necesarios. En la capa anterior se definió el filtro por el que pasan las peticiones a la aplicación, en esta capa se definen los componente que permite el funcionamiento de dicho filtro. Por sus funciones esta capa es considerada transversal a la aplicación y sus componentes son referenciados y utilizados por casi todas las demás capas de la herramienta.

Los componentes más importantes de la capa son el "AuthenticationManager", el "UserDetailsService" y sus configuraciones. El primero es el encargado de autenticar a los usuarios, es decir, confirmar que estos sean quienes dicen ser, el segundo es el responsable de obtener la información de contexto y detalles de los usuarios autenticados y por ultimo las configuraciones de ambos definen la componente de autorización, que define qué acciones puede ejecutar cada usuario.

La implementación del "AuthenticationManager"y el "UserDetailsService" confluyen en la clase "UsersServiceImpl" que implementa el servicio de usuarios de la aplicación, dicha clase es la encargada de determinar si las combinaciones de nombre de usuario y contraseña, suministradas a la aplicación, son válidas o no. En caso de que dichas combinaciones sean válidas también le cabe la responsabilidad de gestionar los detalles de los usuarios.

```
<authentication-manager>
	<authentication-provider user-service-ref="userService" />
</authentication-manager>
```

Ilustración 23 - Definición del Authentication Manager

Una vez autenticado un usuario, Spring Security mantiene información del mismo dentro del objeto "SecurityContext", el que es accedido dentro de la herramienta mediante el bean "securityContext" que referencia un objeto de tipo "SecurityContextFacadeImpl", que facilita el acceso a los datos de contexto que se conservan en la sesión de cada usuario.

```
<bean id="securityContext"
	class="ar.com.alanvido.common.context.impl.SecurityContextFacadeImpl">
```

Ilustración 24 – Definición de SecurityContextFacadeImpl

La definición de la configuración de autorización hace posible indicar a qué URLs es posible acceder sin estar autenticado (siendo un usuario anonimo) y a cuales solo es posible acceder estando autenticado, para ello se definen roles, que en el caso de LEL to UCP, solo se utilizan dos, el rol usuario y el rol anónimo, a continuación se muestra la definición del mapeo de autorización

```
<http pattern="/css/**" security="none" />
<http pattern="/js/**" security="none" />
<http pattern="/login.html*" security="none" />
<http pattern="/newUser.html*" security="none" />
<http pattern="/rest/users" security="none" />
<http pattern="/bootstrap/**" security="none" />
<http auto-config='true'>
       <intercept-url pattern="/secured/**" access="ROLE_USER" />
       <intercept-url pattern="/rest/**/secured/**" access="ROLE_USER" />
       <form-login login-page="/login.html" default-target-url='/secured/index.html' always-
use-default-target='true' />
       <logout logout-success-url="/login.html" />
</http>
```

Ilustración 25 - Definición del Mapeo de Autorización

En la definición anterior se permite el acceso anónimo a los recursos CSS, a los scritps JavaScript que son públicos, así como también, a las páginas y acciones necesarias para la autenticación y creación de nuevos usuarios, esto se hace definiendo los patrones correspondientes a sus URLs y especificando que no serán recursos securizados. Por otro lado se definen las acciones que requieren de un usuario autenticado y se especifica el rol usuario como necesario para ejecutarlas, en esta definición se incluyen dos líneas que indican que se debe volver a la página de inicio de sesión cuando se cierra la sesión, y cuando se intenta acceder a una URL securizada sin tener acceso, se redirige el cliente a la página de inicio de sesión de la aplicación.

Internamente cada rol definido tiene un conjunto de operaciones posibles que son invocadas desde la capa de presentación, se optó por especificar las acciones por cada rol, pese a que en la aplicación solo existe un rol con acciones asociadas, porque este diseño facilita la adición de acciones a dicho rol y la incorporación de nuevos roles con distintos conjuntos de acciones en cualquier extensión futura que se realice sobre la herramienta.

Como la aplicación está pensada para sea usada libremente, el objetivo final de la autenticación y autorización es mantener los datos generados por cada usuario asociados a dicho usuario y que solo él pueda accederlos, modificarlos o eliminarlos, LEL to UCP incluye una pantalla que brinda la funcionalidad necesaria para crear un usuario, en la que solo hay que proporcionar una dirección de correo electrónico y una contraseña. Todos los usuarios creados desde dicha funcionalidad poseen el rol usuario y están habilitados por ende, a realizar todas las acciones permitidas por la herramienta.

Capa de Adaptación de Mensajes

La capa de adaptación de mensajes tiene como responsabilidad adaptar los datos de las peticiones, al modelo de objetos de la aplicación y adaptar el contenido retornado por los controladores, a lo esperado por el cliente, la mayor parte de la lógica de esta capa esta implementada por los componentes provistos por Spring MVC y las tres responsabilidades básicas de esta capa son, la adaptación de las peticiones HTTP a sus correspondientes métodos en los controladores, la decodificación de los contenidos codificados mediante JSon y el manejo de la subida de contenidos multiparte.

La principal función, sin duda, es la adaptación de un pedido HTTP a su correspondiente método y la devolución de su respuesta a las capas superiores. Para realizar esta función se define un bean de tipo "org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter" que se basa en las anotaciones definidas en las clases que implementan los controladores y en sus métodos, a fin de determinar qué tipo de parámetros recibe y qué tipo de retorno posee el método a invocar, convirtiendo los datos en las peticiones HTTP a dichos tipos según lo indican las anotaciones mencionadas.

Existen métodos en la aplicación que aceptan como entrada, o producen como salida, objetos serializados mediante JSon, para adaptar dichos objetos de entrada, es necesario des-serializarlos para obtener los correspondientes objetos del modelo de la aplicación, y para el caso de los métodos que retornan objetos, es necesario serializar dichos objetos a fin de poder retornarlos a la capa de presentación en el formato correspondiente. El encargado de esta tarea es el bean "jacksonMessageConverter" de tipo "org.springframework.http.converter.json.Mapping JacksonHttpMessageConverter" que implementa la serialización / des-serialización explicada anteriormente.

```
<context:annotation-config></context:annotation-config>

<context:component-scan base-package="ar.com.alanvido"></context:component-scan>

<bean id="jacksonMessageConverter"
class="org.springframework.http.converter.json.MappingJacksonHttpMessageConverter"></bean>

<bean
class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">
       <property name="messageConverters">
       <list>
              <ref bean="jacksonMessageConverter" />
       </list>
       </property>
</bean>

<bean id="multipartResolver"
       class="org.springframework.web.multipart.commons.CommonsMultipartResolver" />
```

Ilustración 26 - Definición de la Adaptación de Mensajes

Otro aspecto importante de esta capa es la capacidad de manejar las subidas de archivos multiparte, funcionalidad que se utiliza en la importación de LEL y la cual es soportada mediante la funcionalidad proporcionada por el bean "multipartResolver" de tipo "org.springframework. web.multipart.commons.CommonsMultipartResolver" que se encarga de gestionar la subida del archivo desde el cliente al servidor de a partes o "chunks" las que se unen en el servido obteniendo así el archivo completo.

Capa de Controladores Web

Esta capa está construida basándose en Spring MVC y su modelo de controladores Web, es la encargada de definir las peticiones HTTP que aceptara la aplicación, de delegar la lógica correspondiente en la capa de servicios y de retornar los resultados generados a las capas superiores. LEL to UCP consiste en tres controladores principales que se encargar de las acciones relacionadas con los usuarios, de las acciones correspondientes a los archivos y de las acciones referidas a los proyectos.

El controlador de archivos es el encargado de llevar a cabo las subidas de los archivos que contienen los LEL a importar, este controlador responde a un solo método, “upload”, que esta implementado en la clase LELFileControler. Una vez subido un archivo, este controlador delega en la capa de servicios su parseo y procesamiento.

El controlador de usuarios es el encargado de intermediar en todo lo refente a los usuarios del sistema, sus responsabilidades incluyen, la creación de usuarios, mediante el método “post”, los cambios de contraseña mediante el método “changePass”, el retorno del usuario logueado mediante el método “current”, y por último la iniciación de la recuperación de contraseñas mediante el método “restore”. Las operaciones antes mencionadas están implementadas en la clase UserController, que al igual que el controlador anterior, delega la lógica de cada método en la capa de servicios y en el contexto de seguridad provisto por la capa de autenticación y autorización.

El controlador de proyectos, se encarga de las acciones referentes al manejo de proyectos en la herramienta, estas incluyen la apertura de proyecto, la eliminación de un proyecto, el listado de los proyectos del usuario y las actualizaciones correspondientes a los parámetros de UCP que están a cargo del usuario. El controlador esta implementado en la clase ProjectControler y ésta delega en las capas subyacentes las responsabilidades específicas de cada operación.

A fin de resumir las responsabilidades de cada clase y enumerar las operaciones expuestas por la capa de controladores web, las que son usadas por la capa de presentación a través de peticiones HTTP, se presenta la siguiente tabla en donde se enumeran dichas operaciones, las clases y los métodos que las implementan y si es necesario estar logueado al sistema para poder utilizarlas.

Operación	Método HTTP	Clase/ Método	Seguridad	Observaciones
/users/_current	GET	UserController.getActiveUser()	SI	Retorna el usuario logueado en formato JSon.
/users/secured/_changePass	POST	UserController. changePass()	SI	Los datos son enviados mediante POST y luego de realizar el cambio de contraseña envía un correo electrónico al propietario de la cuenta informado dicha acción. Cierra la sesión del usuario.
/users/_restore	POST	UserController.restore()	NO	Envía un mail al propietario de la cuenta con una contraseña para acceder a la aplicación.
/users	POST	UserController.addNew()	NO	Crea un nuevo usuario con los datos enviados en el cuerpo del mensaje.
/file/secured/lel/upload	POST	LELFileControler.upload()	SI	Sube un archive e inicia la importación de LEL a partir de dicho archivo.
/project/secured/{idProject}/ecf/{factor}	PUT	ProjectControler.putECF()	SI	Actualiza el factor {factor} ECF del proyecto con id={Id}.
/project/secured/{idProject}/tcf/{factor}	PUT	ProjectControler.putTCF()	SI	Actualiza el factor {factor} TCF del proyecto con id={Id}.
/project/secured/{id}/pf	PUT	ProjectControler.putPF()	SI	Actualiza el factor de productividad PF del proyecto con id={Id}.
/project/secured/uaw/{id}	PUT	ProjectControler.putUAW()	SI	Actualiza el UAW del Actor con id={Id}.
/project/secured/uucw/{id}	PUT	ProjectControler.putUUCW()	SI	Actualiza el UUCW del UC con id={Id}.
/project/secured/	GET	ProjectControler.list()	SI	Retorna todos los proyectos disponibles para el usuario en formato JSon.
/project/secured/{id}	GET	ProjectControler.get()	SI	Retorna el proyecto con id={id} en formato Json.
/project/secured/{id}	DELETE	ProjectControler. delete()	SI	Elimina el proyecto con id={id}.

Tabla 8 - Resumen de Operaciones Disponibles

Algunos de los métodos enunciados, principalmente los que utilizan JSon, retornan sus resultados haciendo uso de un elemento contenedor que se creó a tal fin y que tiene por objetivo encapsular el resultado de la petición e información de estado, como pueden ser posibles errores que se produjeron en capas subyacentes o falta de permisos para acceder a un determinado recurso. El objeto contenedor esta implementado por la clase JsonOperationResult y contiene los resultados propiamente dichos, un código de resultado, que indica el éxito o fracaso de la operación y un mensaje que se utiliza para describir situaciones anómalas. Es responsabilidad del invocador hacer uso de esta información de forma consistente, en la herramienta propuesta, dicha responsabilidad recae en la capa de presentación, que es la que invoca las operaciones expuestas por esta capa.

Capa de Servicios

Esta capa es la encargada de manejar la lógica de negocio de la aplicación, recibe las peticiones de las capas superiores, implementa lo necesario para llevar a cabo las acciones requeridas, haciendo uso de las abstracciones expuestas por la capa de acceso a datos y de la funcionalidad brindada por el contexto de seguridad.

Los principales componentes de esta capa son los servicios que implementan el parseo de LEL, la lógica de transformación de LEL en UC y el manejo de proyectos, cuyas implementaciones corresponden a las clases XMLToLELParseServiceImpl, LELtoUCServiceImpl y ProjectServiceImpl respectivamente. Junto a dichos servicios también se encuentran los correspondientes al manejo de usuarios, manejo de email y un servicio genérico que aglutina operaciones comunes, dichos servicios se encuentran implementados en las clases UsersServiceImpl, EmailServiceImpl y DefaultServiceImpl respectivamente.

El servicio de parseo de XML a LEL implementa, mediante el método parse, la lógica necesaria para construir el modelo de objetos correspondiente al LEL de un proyecto, partiendo de un archivo XML proporcionado por el usuario, para que esto sea posible dicho archivo debe especificar el LEL en formato XML respetando el Schema propuesto en la sección Modelo de DatosModelo de Datos.

El servicio de transformación de LEL a UC permite, por medio de la transformación descripta en el capítulo anterior, transformar el modelo de objetos que representa el LEL de un proyecto, en el conjunto de Casos de Uso correspondientes a la aplicación de dicha transformación. El método principal de dicho servicio es generarUCparaProyecto, este es el encargado de tomar el LEL de un proyecto y aplicar los pasos necesarios para obtener su correspondiente modelo de Casos de Uso. Además de implementar la transformación mencionada, este servicio también se encarga de inferir el número de clases necesarias para la implementación del UC y el número de entidades de base de datos a las cuales afectara el UC generado, todo esto siguiendo los métodos descriptos en el capítulo anterior. Para poder implementar la lógica necesaria, este servicio se basa en implementaciones de las heurísticas y procesos mencionados en los capítulos anteriores y cuyas implementaciones se analizan en detalle en la sección “Aspectos Críticos en el Desarrollo”.

El servicio de proyectos se encarga de implementar la lógica que da soporte a la administración de proyectos en la herramienta, brindando operaciones para la búsqueda, actualización, salvado, eliminación de proyectos y sus componentes, delega parte de sus responsabilidades en la capa de acceso a datos y en el contexto de seguridad.

El servicio genérico, o defaultService, implementa la lógica genérica de búsqueda, persistencia y eliminación de entidades, haciendo uso de la las capas subyacentes. El servicio de email se encarga de la lógica correspondientes al envío de correos electrónicos, la cual se utiliza para el recupero de las contraseñas de la herramienta, así como para dar la bienvenida a LEL to UCP a los nuevos usuarios o informarles sobre los cambios en sus cuentas de acceso. El servicio de usuarios da soporte a las operaciones de alta, baja, y modificación de usuarios así como también actúa en el proceso de recupero de contraseñas y colabora con la capa de autenticación y autorización.

Capa de Acceso a Datos

La capa de acceso a datos constituye el nexo entre el motor de base de datos y la aplicación, su principal función es abstraer a los servicios de las capas superiores de cómo se llevan a cabo las operaciones de búsqueda, actualización, creación y eliminación de datos que dichas capas requieren. Esta capa está compuesta por tres clases que brindan soporte de acceso a datos referente a los usuarios, a los proyectos y a las operaciones comunes. Dichas clases hacen uso de los elementos provistos por Hibernate y Spring Framework, así como también del contexto de seguridad.

Las clases que proveen el acceso a datos en LEL to UCP son GenericDaoImpl, que implementa las operaciones comunes de acceso a datos, como son las búsquedas, inicialización de objetos, persistencia de objetos, eliminación de objetos, etc. La clase UserDaoImpl, que provee las operaciones de acceso a datos especificas del manejo de usuarios, a saber, se encarga del login, de la recuperación de usuarios y de la recuperación de roles. Por último se encuentra ProjectDaoImpl que implementa al momento una única operación que es el listado de la información de proyectos asociados al usuario logueado.

Las clases involucradas en esta capa hacen uso, de la SessionFactory de Hibernate, del manejador de transacciones de Spring, de los pooles de conexión provistos por c3p0 y finalmente de la implementación de JDBC especifica de PostgreSQL, que configurados en conjunto permite el acceso a la base de datos a través de la implementación de Criterias y consultas HQL, de forma tal que la herramienta es prácticamente independiente del motor de base de datos elegido.

Capa de Auditoria

La capa de auditoria es la encargada de agregar metadatos a los objetos persistidos para identificar quien es el propietario de cada objeto, quien es su creador, cuál es su fecha de creación, si el objeto ha sido modificado y cuándo es que se ha realizado dicha modificación, así como también es la encargada de mantener el estado de los objetos, a saber, en LEL to UCP los objetos persistidos pueden encontrarse en alguno de los tres estados lógicos, Activos, que indica que es posible utilizarlos, Borrados, que indica que han sido eliminados de forma lógica, o pueden estar en un tercer estado que es En Proceso, que señala aquellos objetos que son usados internamente por la aplicación pero no están disponibles a través de los métodos de acceso a datos típicos, este estado se utiliza cuando se debe persistir parcialmente un objeto que luego completará su definición, y no debe ser expuesto hasta que se encuentre completo.

Esta capa esta implementada por la clase AbstractPersistentObjectInterceptor que mediante programación orientada a aspectos o AOP (del inglés, Aspect oriented Programming), intercepta el salvado de objetos que heredan de la clase AbstractPersistentObject, la que define los datos básicos de un objeto persistente en la herramienta, y añade la información correspondiente a la acción que se está realizando con dicho objeto, la mayor parte de la información se obtiene del contexto de seguridad proporcionado por la capa de autenticación y autorización.

Modelo de Objetos

En esta sección se presentan los diagramas de clases correspondientes a los objetos de la aplicación, a fin de mejorar su legibilidad se los dividió en seis diagramas. En el primer diagrama hace referencia a las entidades que dan soporte al resto de la aplicación, el segundo, expone el modelo propio de las entidades de la aplicación, como son proyecto, LEL, UC, etc. El tercer modelo presenta los componentes que dan soporte a la autenticación y autorización, el cuarto diagrama muestra la implementación de la capa de controladores, el quinto diagrama corresponde con la implementación de los servicios y el último exhibe la composición de la capa de acceso a datos.

En los diagramas que componen esta sección solo se incluyen las clases que fueron necesarias implementar y se excluyen las propias de los frameworks que dan soporte a dichas implementaciones o aquella que se utilizan mediante configuración de los mismos.

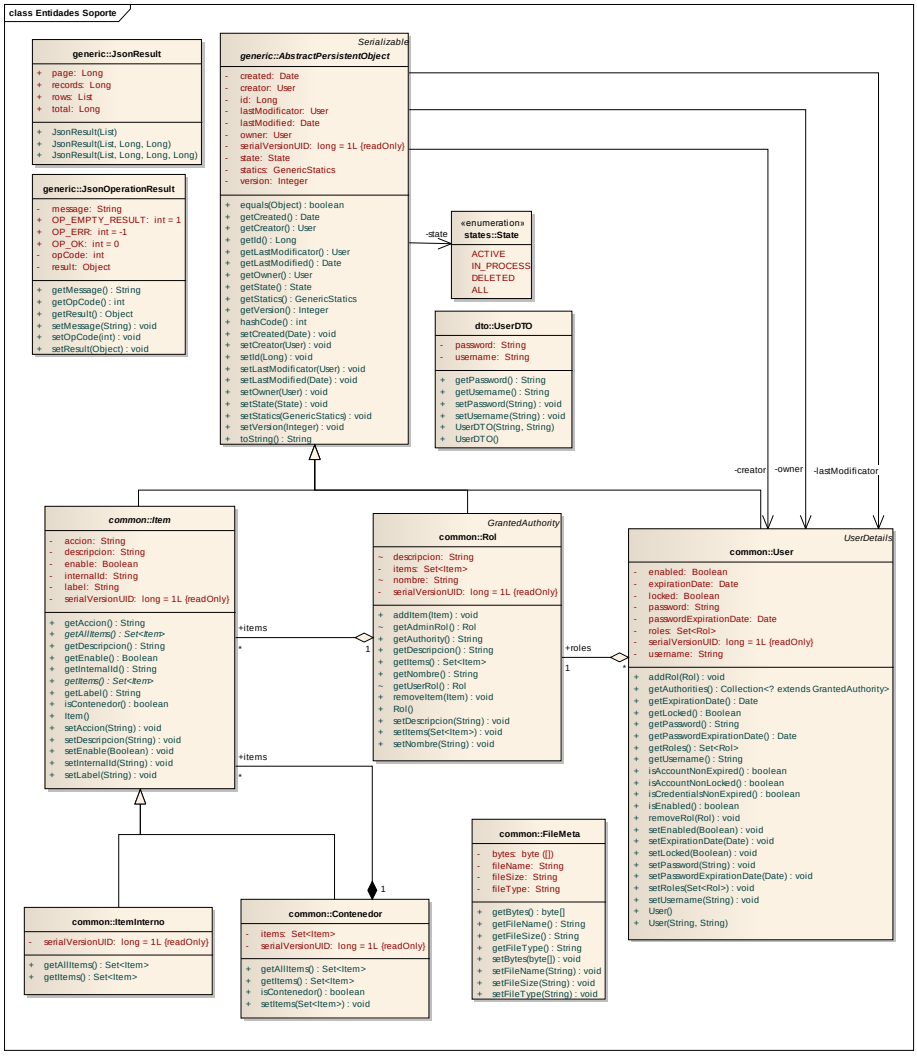

Ilustración 27 - Modelo de Objetos - Entidades de Servicio

En el diagrama anterior se presentan las clases correspondientes a las entidades que dan soporte a la aplicación, a continuación se enumeran las principales y sus responsabilidades, la clase JsonResult es la encargada de almacenar listados de resultados en formato Json. La clase JsonOperationResult es la responsable de almacenar los resultados de la invocaciones a operaciones que deben retornar sus resultados en formato JSon, sus instancias encapsulan el resultado de la operación, un código que indica el éxito o fracaso de la invocación y un mensaje que se utiliza para describir las causas de los errores o situaciones anómalas.

La clase AbstractPersistentObject es la clase base de la cual heredan todas las clases que representan entidades persistentes en el sistema, esta clase contiene un conjunto de propiedades útiles a todas sus hijas, como son el identificador y la versión, utilizada por Hibernate, y las propiedades de auditoria como el creador, el ultimo usuario en modificar la entidad, el dueño de la entidad, la fecha de creación, su última fecha de modificación y su estado. Además se incluyen en este diagrama las clases que representan a los usuarios y sus roles, la enumeración que representa los estados de las entidades del sistema, la clase FileMeta que se utiliza como abstracción de un archivo y se utiliza en la importación de LEL a la herramienta.

El siguiente diagrama presenta las clases que modelan los datos propios de la herramienta cuya clase principal es Proyecto, que representa el proyecto de un usuario, esta clase aglutina el LEL importado, los Casos de Uso generados y las métricas correspondientes al método de estimación propuesto por este trabajo. La clase Termino se utiliza para modelar los términos pertenecientes al LEL importado, el cual se compone de una noción, una colección de impactos y otra de sinónimos. La enumeración TipoTermino permite diferenciar los tipos de términos que constituyen el LEL.

Los impactos, sinónimos y nociones se modelan con las clases Impacto, Sinonimo, Nocion respectivamente, estas se diseñaron formando parte de una jerarquía para simplificar su persistencia, considerando que, a nivel datos su única función es almacenar una cadena de caracteres que representa su significado, omitiendo intencionalmente la noción de que a nivel dominio representan cosas completamente diferentes.

La clase CasoDeUso hace lo propio modelando los UC generados por la herramienta y estos a su vez contienen una colección de actores, los que son representados por la clase Actor. Por último se definen dos enumeraciones que se utilizan para describir los factores de complejidad técnica y ambiental que afectan al proyecto.

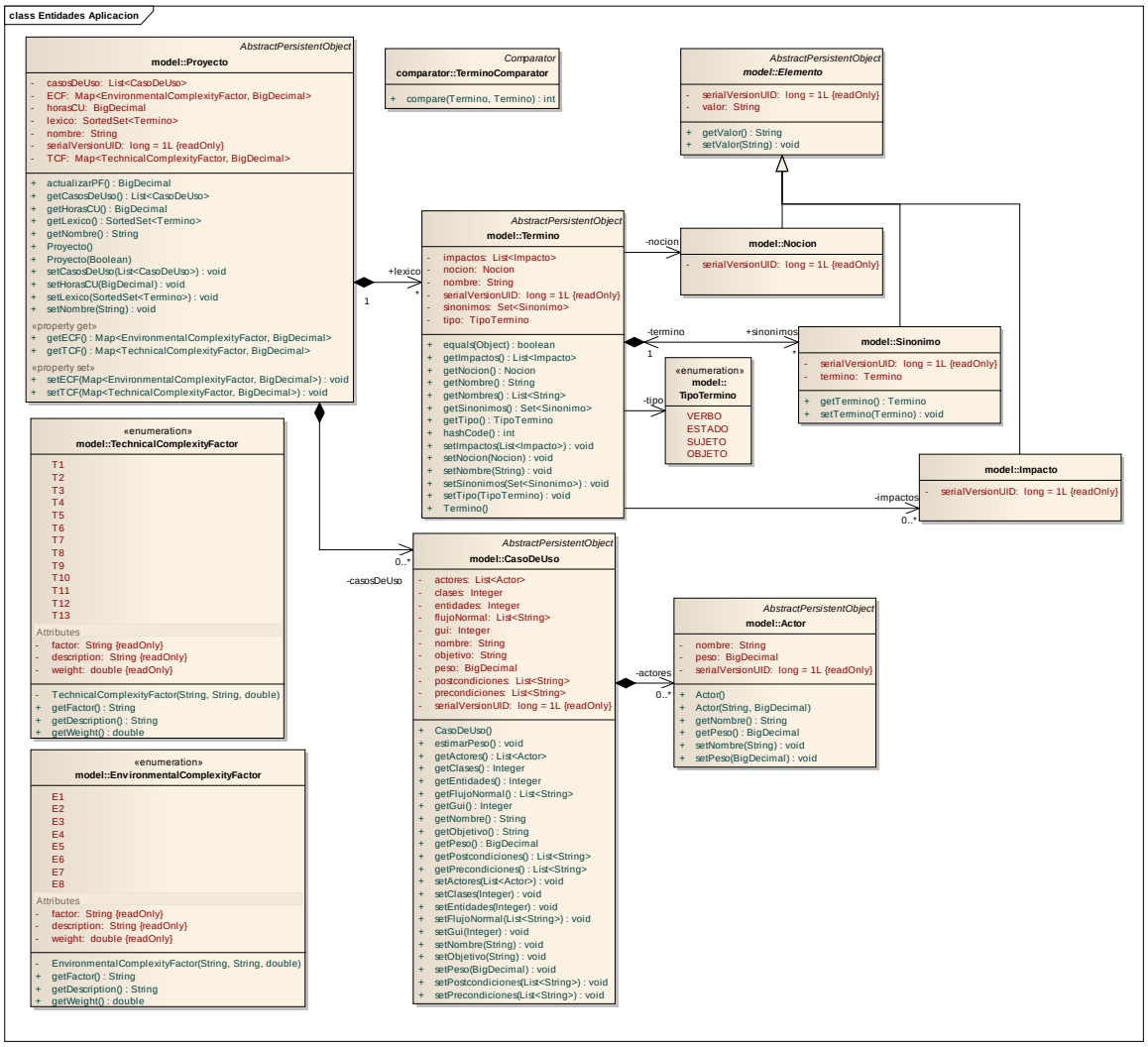

Ilustración 28 - Modelo de Objetos - Entidades de la Herramienta

A continuación se presenta el diagrama correspondiente a las clases que colaboran en la capa de autenticación y autorización. La clase principal de este diagrama es la implementación de SecurityContextFacade, la cual es responsable de proveer una fachada de acceso al contexto de seguridad de la aplicación y es utilizada por las demás capas del sistema. También se presenta la clase UserServiceImp, que implementa las operaciones relativas a los usuarios y los métodos declarados por las interfaces AuthenticationManager y UserDetailService los que son utilizados por los beans de Spring Security para proporcionar soporte de autenticación y autorización.

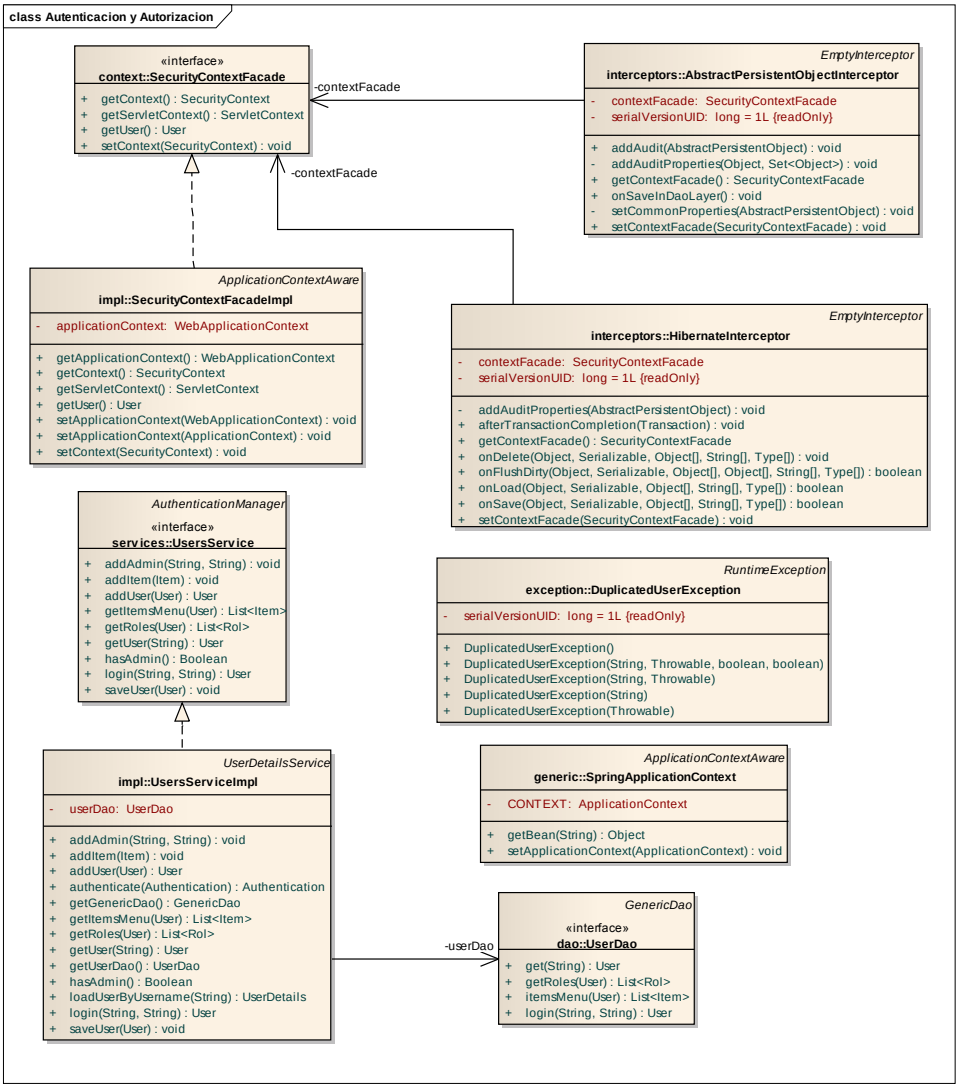

Ilustración 29 - Modelo de Objetos - Autenticación y Autorización

El próximo diagrama presenta las clases que interactúan en la capa de controladores, en donde se encuentran las clases UserController, ProjectController y LELFileController, que se encargan de exponer las operaciones a la capa de presentación. La clase UserController define las operación de interacción con los usuarios de la aplicación, la clase LELFileController es la responsable de la subida de archivos a LEL to UCP mediante el método upload, que implementa la lógica necesaria para subir archivos multiparte. Por último la clase más importante es ProjectController que implementa toda la lógica relacionada con el manejo de los proyectos de los usuarios de la aplicación.

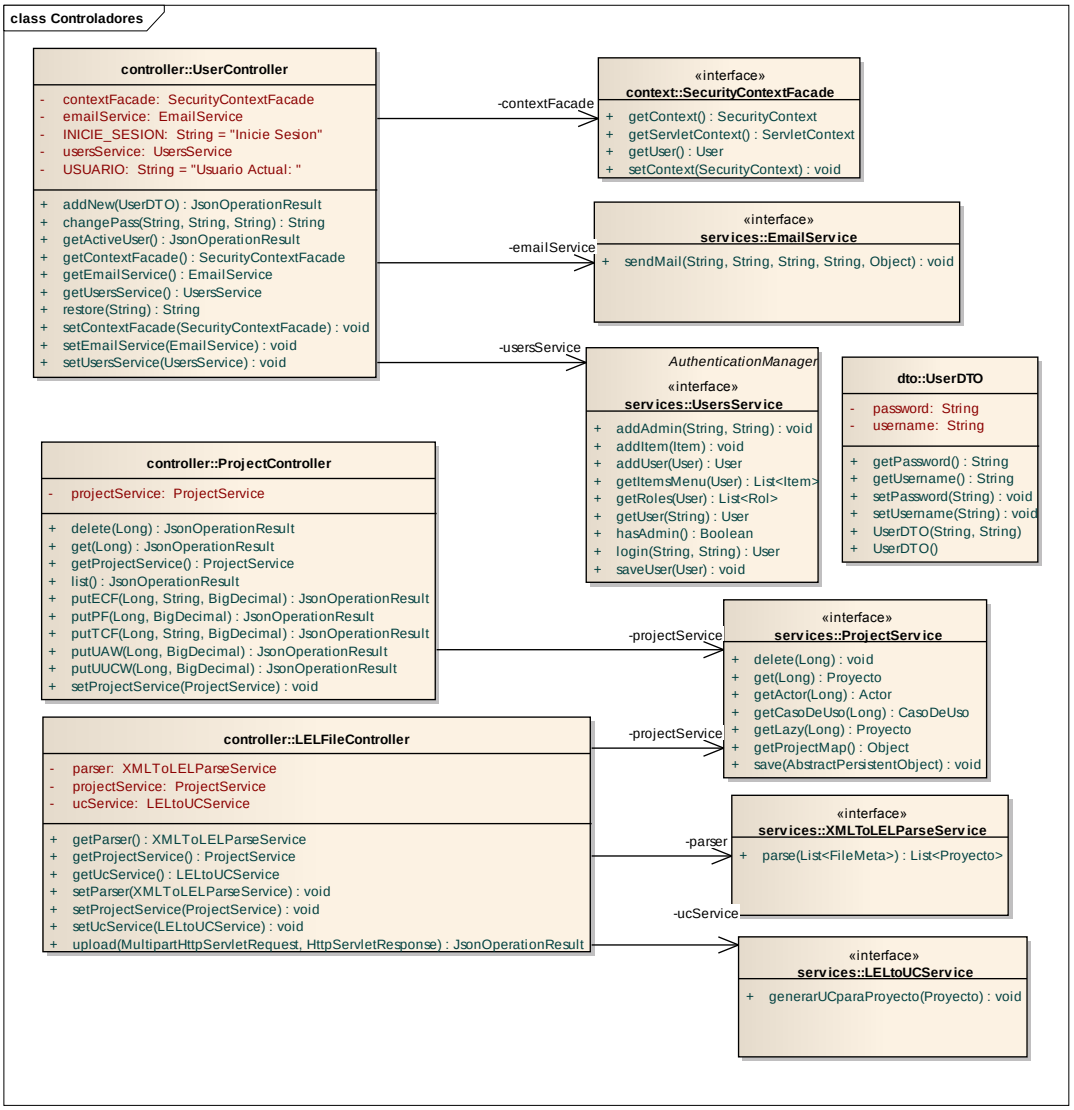

Ilustración 30 - Modelo de Objetos - Controladores

El siguiente diagrama presenta las clases que componen la capa de servicios, las principales clases expuestas corresponden a las implementaciones de las interfaces DefaultService, ProjectService, UserService y LELToUCService. La clase DefaultServiceImpl provee a la aplicación de la lógica genérica usada para interactuar con la capa de acceso a datos. La clase ProjectServiceImpl provee toda la lógica correspondiente al manejo de proyectos en la aplicación. La clase UserServiceImpl brinda soporte para las operaciones propias del manejo de usuarios en LEL to UCP. Por último la clase LELToUCServiceImpl implementa la lógica de negocio que hace posible transformar un LEL en el correspondiente conjunto de Casos de Uso, así como también lleva a cabo las operaciones de inferencia de clases necesarias para la construcción de cada UC y la inferencia de entidades de base de datos afectadas por cada UC.

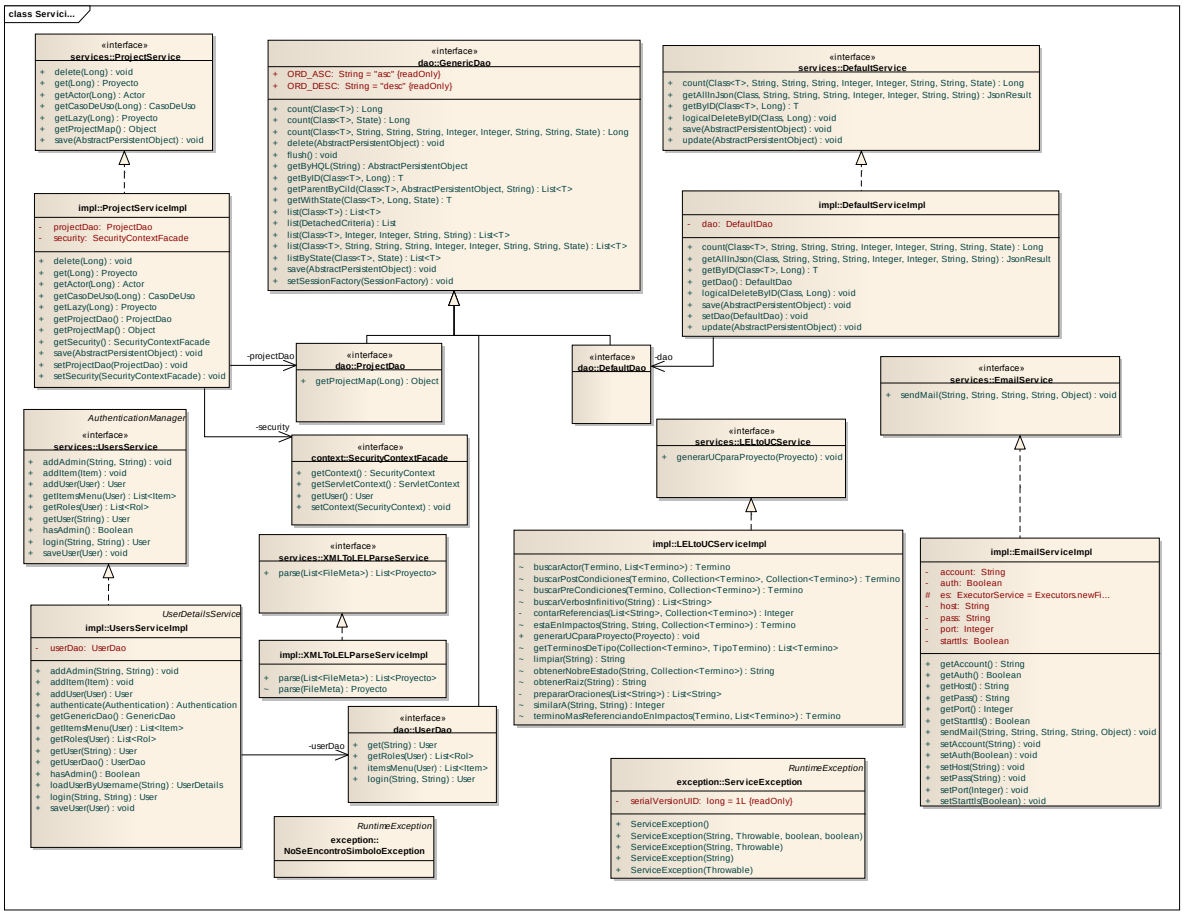

Ilustración 31 - Modelo de Objetos - Servicios

Por último se presenta el diagrama correspondiente a la capa de acceso a datos, en donde se exponen las principales clases que la integran. Su clase principal el GenericDaoImpl, la que define un conjunto genérico de operaciones de acceso a datos que heredan el resto de las clases que implementan operaciones de acceso a datos. La clase UserDaoImpl es la encargada de exponer las abstracciones necesarias para las operaciones específicas del manejo de usuarios. La clase ProjectDaoImpl implementa la lógica necesaria para los listados de proyectos por usuario. La clase DefaultDaoImpl se encuentra sin implementación actual, pero se incluyó en el diagrama ya que su diseño original la propone como el lugar donde implementar las operaciones genéricas que no deben ser heredadas por todos los DAOs de la aplicación. También se incluyen las clases que implementan los interceptores de auditoria, los que tiene definida la lógica para incluir los metadatos correspondientes a la operación que se está realizando con cada entidad persistente del sistema.

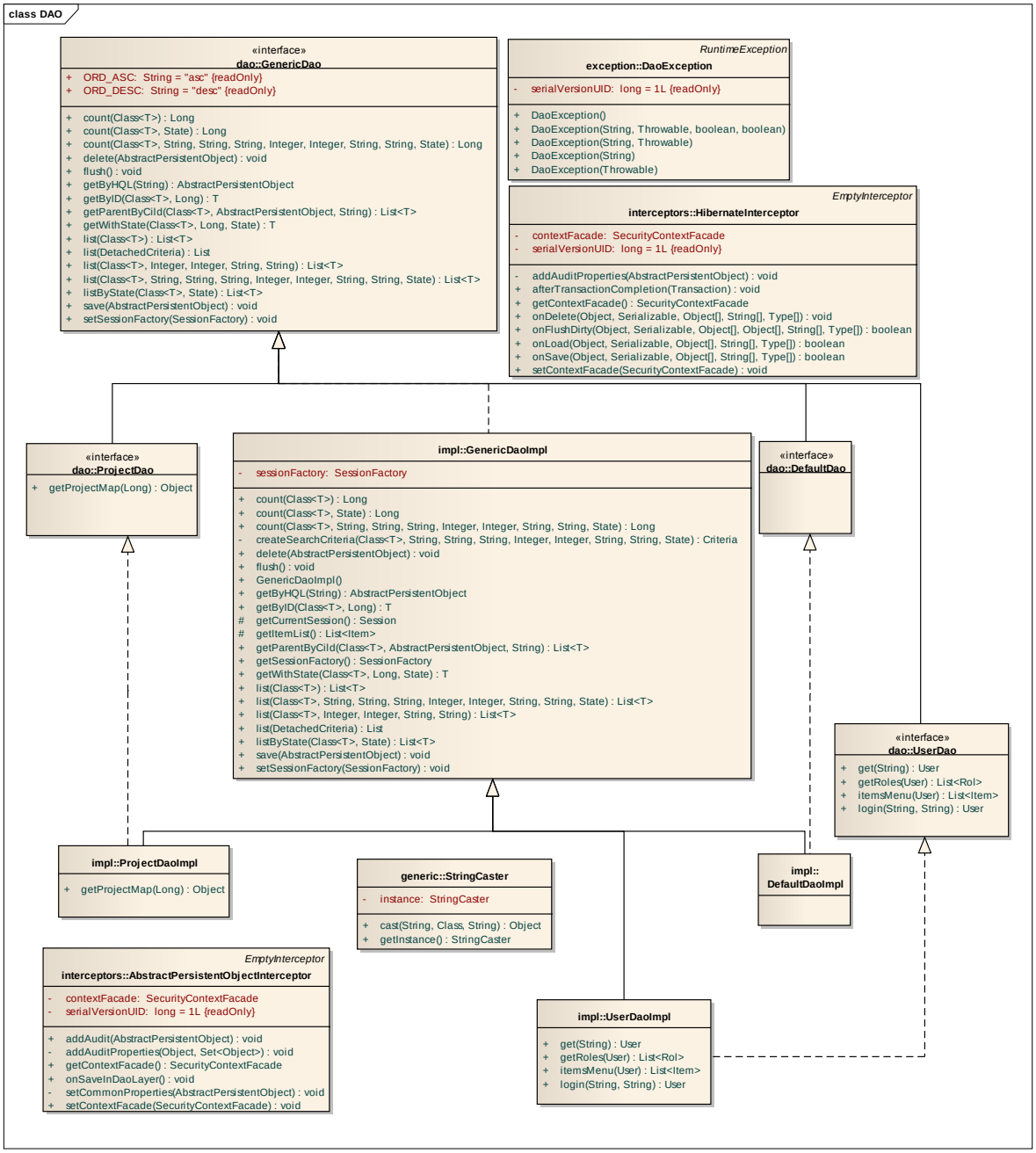

Ilustración 32 - Modelo de Objetos - DAOs

En la correspondiente sección se analizó el modelo de LEL to UCP desde una visión del diseño de los objetos que interactúan para llevar a cabo el propósito de la herramienta. En la siguiente sección se prosigue con este análisis con un enfoque orientado a los datos que consume y genera la herramienta.

Modelo de Datos

En esta sección se describe el formato de archivos utilizados por la aplicación y el modelo de base de datos que da soporte a los objetos con los que trabaja la herramienta. Como se mencionó anteriormente, los archivos utilizados por la aplicación deben estar en formato XML y ser archivos bien formados y validos con respecto al XML Schema que se presenta a continuación.

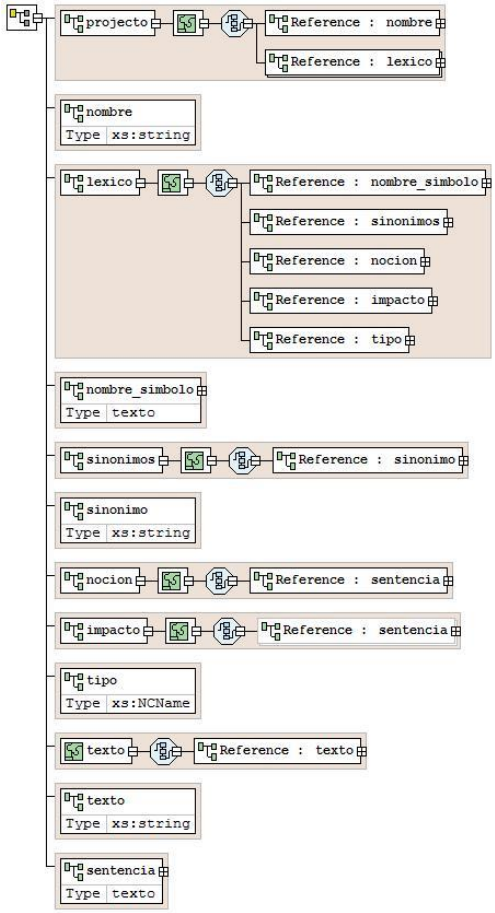

Ilustración 33 - Modelo de XML Schema Soportado por LEL to UCP

El diagrama precedente puede ser traducido a una definición de XML Schema que se enuncia en la siguiente página. El diseño de Schema presentado, fue influenciado por la necesidad de poder ser transformado desde los documentos generados por herramientas como C&L [**25**], por lo cual se tomó el formato de archivo generado por dicha aplicación y se lo optimizó para facilitar su procesamiento, pero teniendo siempre como premisa no complejizar en exceso el proceso de transformación, procurando obtener un diseño balanceado entre la facilidad de procesamiento y la facilidad de escritura de dichos archivos.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="projecto">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="nombre"/>
        <xs:element maxOccurs="unbounded" ref="lexico"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="nombre" type="xs:string"/>
  <xs:element name="lexico">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="nombre_simbolo"/>
        <xs:element ref="sinonimos"/>
        <xs:element ref="nocion"/>
        <xs:element ref="impacto"/>
        <xs:element ref="tipo"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="nombre_simbolo" type="texto"/>
  <xs:element name="sinonimos">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="sinonimo"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="sinonimo" type="xs:string"/>
  <xs:element name="nocion">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="sentencia"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="impacto">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="sentencia"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="tipo" type="xs:NCName"/>
  <xs:complexType name="texto">
    <xs:sequence>
      <xs:element ref="texto"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="texto" type="xs:string"/>
  <xs:element name="sentencia" type="texto"/>
</xs:schema>
```

Ilustración 34 - XML Schema Utilizado por LEL to UCP

A continuación se introduce el modelo de base de datos que da soporte a la herramienta, para mejorar su legibilidad, se divide dicho modelo en dos diagramas, el primero incluye las entidades que dan soporte a la aplicación y el segundo se encarga de representar las entidades propias del modelo de negocio de la herramienta.

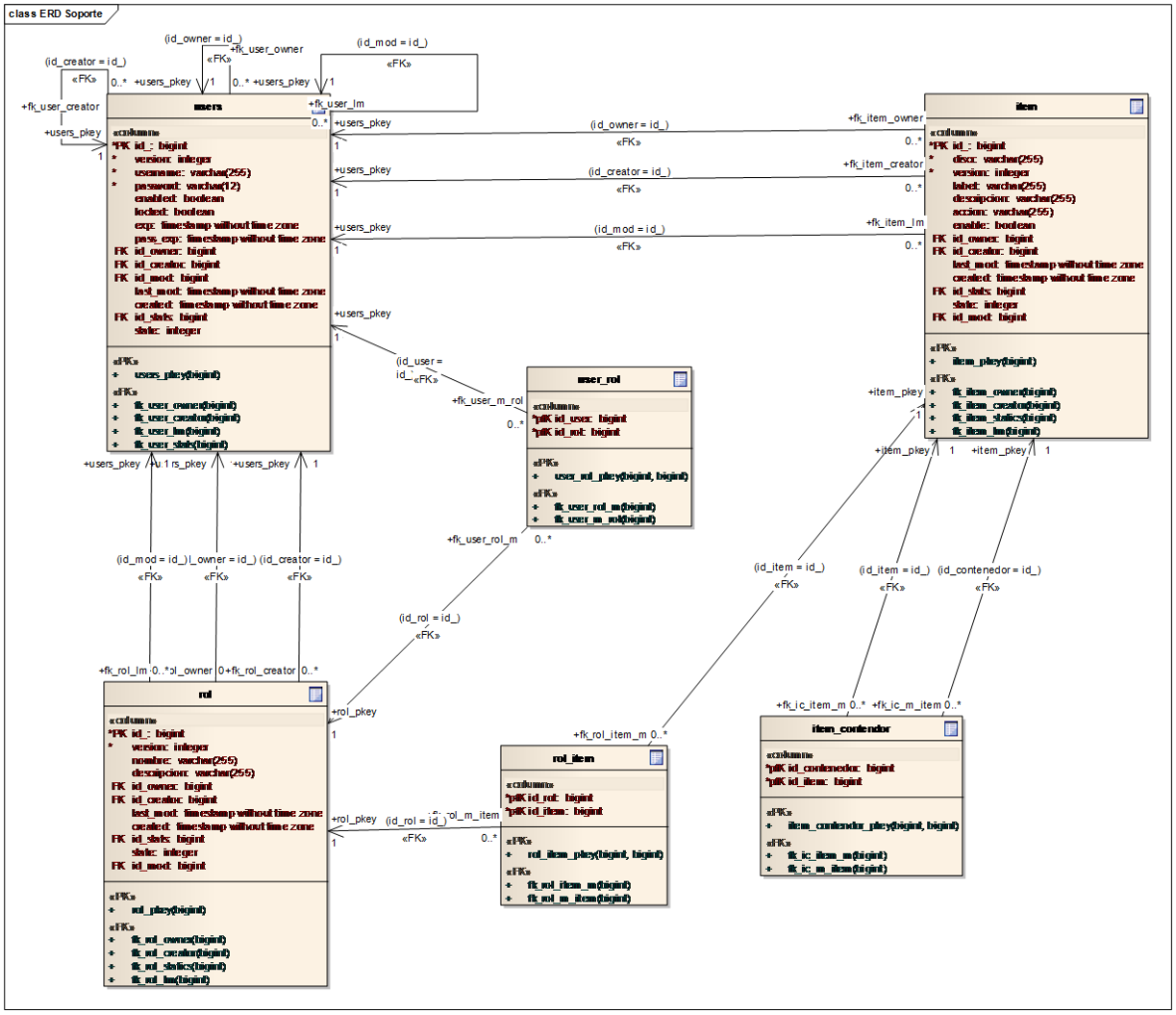

Ilustración 35 - Modelo de Base de Datos - Entidades de Soporte

En el diagrama anterior se representan las tablas que dan soporte al manejo de usuarios de la aplicación, como se enunció a lo largo del trabajo, los usuarios tiene roles y dichos roles los habilitan a realizar acciones en el sistema, la información necesaria para modelar esta situación se almacena en las tablas correspondientes a los roles e ítems, los que representan las acciones.

El diagrama que se presenta a continuación es el correspondiente a las entidades que son propias del modelo de negocio de la herramienta construida, a saber se presentan las tablas que se encargan de almacenar la información correspondiente a los Casos de Uso, los Proyectos, los Términos de los LEL y toda la información necesarias para calcular las estimaciones propuestas en el presente trabajo.

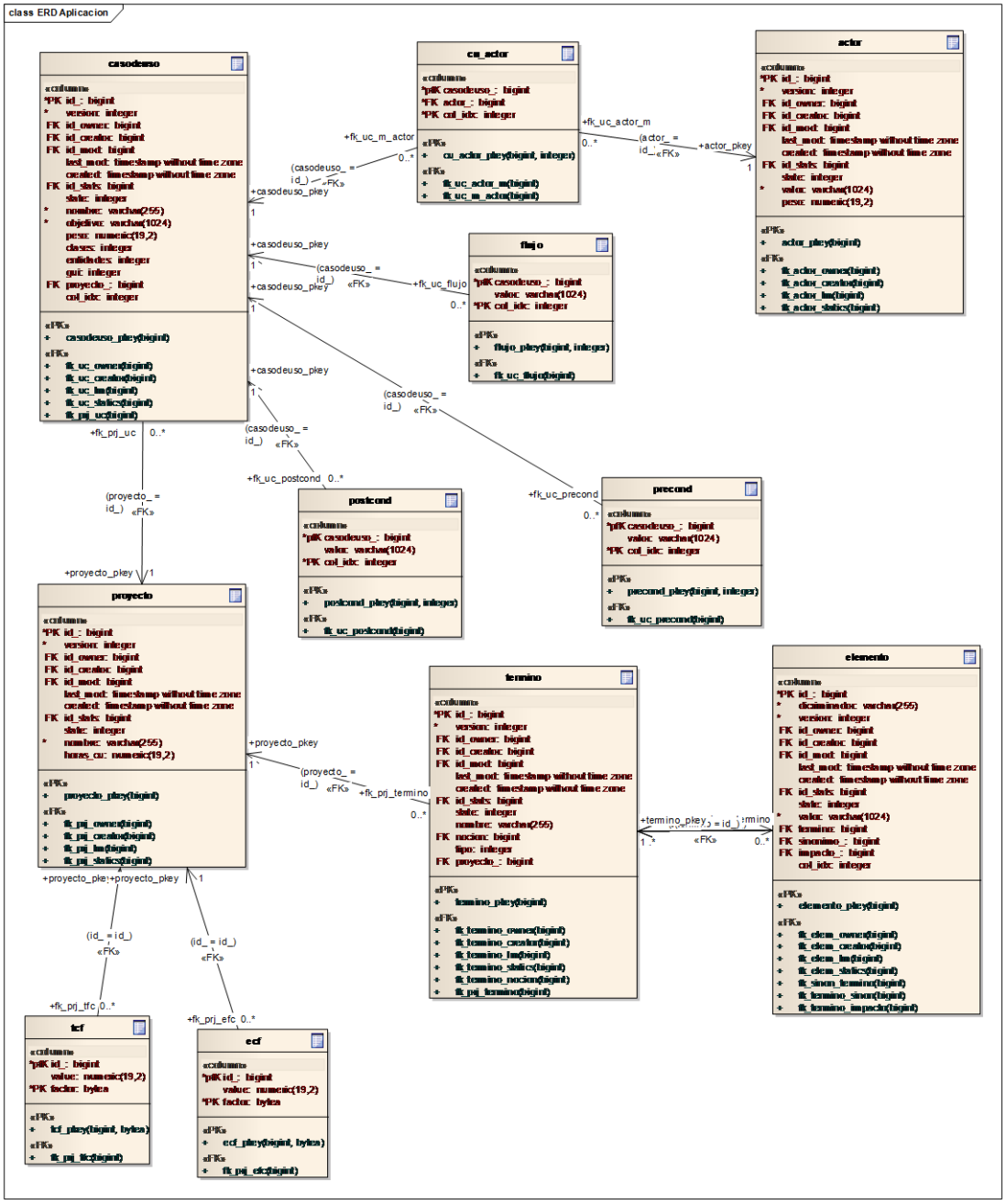

Ilustración 36 - Modelo de Base de Datos - Entidades de Dominio

Aspectos Críticos en el Desarrollo

Bajo este título se presentan los algoritmos y decisiones de implementación que se consideran importantes en el desarrollo de este trabajo, se explican los factores que motivaron dichas decisiones, y como éstas afectan a la herramienta construida.

De acuerdo con el método planteado en el presente trabajo, las tres tareas más complejas a nivel algorítmico son: la trasformación del LEL en un conjunto de Casos de Uso, la inferencia del número de clases necesarias para implementar un UC y la inferencia del número de entidades afectadas por un Caso de Uso. Este conjunto de tareas comparte cierta lógica, principalmente en el procesamiento de cadenas de caracteres que conforman oraciones, las que describen los símbolos del LEL, su noción, impacto y sus sinónimos. En los siguientes párrafos se explicarán los principales algoritmos utilizados para construir las heurísticas aplicadas en las tareas mencionadas.

El principal problema encontrado en la implementación de la transformación de LEL a UC utilizada en el presente trabajo, viene dado por la necesidad de buscar referencias a un término T en los impactos de otros términos, a modo de ejemplo sea T el término de tipo Verbo “Crear cuenta de usuario”, sea S un término de tipo Sujeto con nombre “Administrador” y sea I una definición de impacto del termino S, de modo “El administrador crea una cuenta de usuario”. A simple vista el lector puede apreciar que el termino S referencia al término T mediante su impacto I, Pero a nivel algorítmico no es una decisión lineal ya que si comparamos la cadena expresada por I con el nombre del termino T, en adelante Nombre(T), I no contiene a Nombre(T), sino que contiene una conjugación del verbo empleado en Nombre(T).

El problema planteado en el párrafo anterior aplica a la definición del algoritmo usado para la búsqueda de actores de un Caso de Uso y en las búsquedas de precondiciones de los UC. En respuesta a esto se decidió que el enfoque más balaceado entre la complejidad algorítmica y la exactitud en la búsqueda de las correspondencias del tipo expuesto, estaba dada por la aplicación de una heurística que hiciera uso de la información disponible en las oraciones para obtener los verbos en infinitivo que esta contienen, para luego proceder a la descomposición de la oración y del verbo en su raíz o lexema y su desinencia o morfema. De esta manera podemos ver a I como el lexema del verbo, su correspondiente morfema y un conjunto de palabras (no verbos) que contextualizan al verbo anterior, de esta manera y para nuestra heurística:

$$I = \text{El administrador crea una cuenta de usuario}$$

Es similar a

$$I = I_l + I_m + I_r$$

En donde I_l corresponde al lexema del verbo crear (cre), I_m corresponde al morfema del verbo (a) y por ultimo I_r en un conjunto ordenado de las palabras que contextualizan el verbo anterior (el, administrador, una, cuenta, de, usuario)

Y por otro lado si aplicamos el mismo procedimiento al nombre del término T tenemos:

$$Nombre(T) = \text{Crear cuenta de usuario}$$

Entonces Nombre(T)_l está dado por el lexema del verbo crear (cre), Nombre (T)_m corresponde al morfema (ar) y Nombre(T)_r se compone con (cuenta, de, usuario). De esta manera podemos descartar los morfemas y alivianar el concepto de contención de forma tal que I contiene a

Nombre(T), si: $I_l = Nombre(T)_l$ y I_r *contiene la mayoria de las palabras en* $Nombre(T)_r$ en donde la cantidad de palabras que configuran "la mayoría de la palabras" permite balancear la eficiencia y la eficacia de la heurística.

Es menester aclarar que esta heurística se basa en que la definición de los términos del LEL respeta lo propuesto en la sección "Buenas prácticas en la escritura de un Léxico Extendido del L" y que no presta especial atención a los verbos irregulares, ya que lo que se busca es optimizar la comparación y no hacer un análisis lingüístico de la oración en cuestión. Cabe la aclaración que los casos más habituales de verbos irregulares, como puede ser el verbo "dar" y su conjugación "da", la heurística planteada produce un resultado satisfactorio tomando como lexema "d" y morfema "ar" / "a" siendo estos incorrectos en términos lingüísticos pero sirviendo perfectamente a los fines propuestos.

Siguiendo con la estrategia de simplificar las operaciones entre oraciones, en casos como el conteo de clases necesarias para la implementación de un Caso de Uso, o el conteo de entidades de bases de datos afectadas por un UC, se simplifican las palabras que las componen eliminado las terminaciones que indican plurales, quitando los símbolos que indican acentuaciones o puntuaciones, y convirtiendo todas las oraciones a caracteres en minúsculas. Obteniendo de esta manera un resultado aceptable y una mayor eficiencia en su procesamiento.

Por otro lado, al momento de definir la estrategia de persistencia de las entidades utilizadas en la aplicación, se optó por mantener una sincronización de los cambios producidos por el usuario y los datos en la base de datos, de forma tal que al realizar un cambio en alguna entidad, la aplicación persista dicho cambio en la base de datos, en vez de requerir que el usuario indique que la acción debe llevarse a cabo. Se escogió este modelo teniendo en mente que de ésta manera se le da la posibilidad al usuario de abandonar un proyecto en cualquier momento, con la seguridad de que cuando retome su actividad dicho proyecto se encontrará exactamente igual a como lo dejó. Es necesario aclarar que la aplicación depende de muchos componentes como son: redes, motores de bases de datos, servidores Web, etc. Y que una falla que provoque una interrupción en la conexión de dichos componentes puede comprometer el correcto funcionamiento de la misma.

Instalación de la Herramienta

En esta sección se describe el proceso de instalación de la herramienta, para ello, se comienza por enunciar los requisitos de software que habilitan la tarea en cuestión, para luego enumerar los pasos a seguir en la instalación de LEL to UCP, finalizando con las operaciones sugeridas para validar el funcionamiento de la instalación realizada.

Requisitos Mínimos

LEL to UCP es una aplicación Web que hace uso de un motor de base de datos PostgreSQL, la herramienta está desarrollada para ejecutarse en un contenedor de aplicaciones Web.

A continuación se listan los requisitos de software necesarios para llevar a cabo la instalación de la herramienta.

1. Java SE 7
2. Apache Tomcat 7.x
3. PostgreSQL 9.0 o superior, en donde exista una base de datos con un esquema para alojar la aplicación y un usuario con permisos sobre dichos elementos.

Proceso de Instalación

El proceso de instalación de LEL to UCP consta de una serie de pasos que deben seguirse de forma estricta a fin de garantizar el correcto funcionamiento de la aplicación, dichos pasos se listan a continuación

1. Apagar el contenedor de aplicaciones Apache Tomcat.
2. Creación del archivo de configuración: en primer lugar se debe crear un archivo de propiedades llamado “leltoucp.properties” en el directorio “conf” de la instancia del servidor Web a utilizar. Dicho archivo debe tener el siguiente formato:

```
jdbc.url=jdbc\:postgresql\://host_name\:port_number/data_base_name
jdbc.user=user_name
jdbc.pass=password
hibernate.default_schema=schema_name
hibernate.hbm2ddl.auto = create
mail.account=leltoucp@mail.com
mail.pass=Pa55W0r4
mail.smtp.host=smtp.mail.com
mail.auth=true
mail.start.tls=true
mail.smtp.port=587
```

Ilustración 37 - Modelo de Archivo de Configuración

En donde “host_name” se debe reemplazar por el nombre del host donde se encuentra el servidor de base de datos, “port” corresponde al puerto del servidor de base de datos, “data_base_name” corresponde al nombre asignado a la base de datos que alojará los datos de la herramienta, “schema_name” corresponde al esquema de base de datos a utilizar, “user_name” hace referencia al nombre de usuario asignado para la herramienta, “password” corresponde a la contraseña del usuario y la propiedad “hibernate.hbm2ddl.auto” se crea inicialmente con el valor “create”. Las propiedades referentes al envío de correos electrónicos corresponden a la cuenta desde donde se enviarán dichos correos, ésta se configura mediante la propiedad “mail.account”, la contraseña correspondiente a la cuenta se configura mediante la propiedad “mail.pass”, el servidor smtp se indica mediante la propiedad “mail.smtp.host”, “mail.auth” y “mail.start.tls”

corresponden a la seguridad del servidor y por ultimo “mail.smtp.port” hace referencia al puerto en el que escucha el servidor smtp.

3. Copiar WAR de la herramienta “leltoucp.war” a la carpeta “webapps” de la instancia del servidor web a utilizar.
4. Encender el contenedor de aplicaciones Apache Tomcat y validar el despliegue de la aplicación.
5. Apagar el contenedor de aplicaciones Apache Tomcat.
6. Editar el archivo creado en el punto 1 de tal modo que a la propiedad “hibernate.hbm2ddl.auto” se le asigne el valor “none”.
7. Encender el contenedor de aplicaciones Apache Tomcat y validar el despliegue de la aplicación.

Luego de la ejecución de los pasos mencionados, la aplicación debería encontrarse en funcionamiento, para validar esta condición se enumera una serie de operaciones y sus resultados esperados a fin constatar el funcionamiento de la herramienta.

Pruebas Mínimas de Funcionalidad

Para validar el funcionamiento de una nueva instalación de LEL to UCP se sugiere realizar un conjunto de operaciones básicas a fin de comprobar la disponibilidad de la herramienta. Para realizar las operaciones mencionadas se sugiere leer la sección correspondiente al manual de usuario a fin de evacuar dudas de cómo realizar las operaciones que se enuncian a continuación.

1. Acceso a la aplicación mediante un navegador Web dirigido a la URL http://servidor:puerto/leltoucp, en donde “servidor” se refiere a la dirección IP o nombre del servidor donde se encuentra instalada la herramienta y “puerto” corresponde al número de puerto en el que el servidor escucha peticiones. Esta operación debería presentar al usuario la página principal de la herramienta.
2. Creación de un nuevo usuario. Mediante la funcionalidad provista por la herramienta, crear un nuevo usuario, proporcionando una dirección de correo electrónico válida. Como resultado de esta operación, el sistema debería enviar un correo electrónico de bienvenida a la cuenta provista y redirigir el navegador a la pantalla principal de la aplicación.
3. Inicio de sesión. Desde la pantalla principal, iniciar sesión con el usuario creado anteriormente. Esta operación debería dirigir al usuario a la pantalla de importación de LEL.
4. Importación de un LEL. Descargar uno de los archivos de ejemplo provistos por la aplicación e importarlo. Esta operación debería mostrar al usuario el LEL importado.

Validadas estas operaciones se puede considerar correctamente instalada la herramienta y habilitar su uso a terceros.

Manual de Usuario

En esta sección se brinda un manual que introduce al usuario en el funcionamiento de la herramienta propuesta en el presente trabajo, para ello se presenta la interfaz utilizada en LEL to UCP a fin de familiarizar al lector con dicha interfaz y luego se prosigue abordando cada funcionalidad de la herramienta. Las explicaciones en este manual hacen énfasis en tres factores principales respectivos a cada funcionalidad, siendo estos el objetivo de dicha funcionalidad, como accederla y que resultados esperar de la funcionalidad indicada.

Organización de la Interfaz

La herramienta desarrollada a lo largo de este trabajo presenta una interfaz tabular, en donde es posible acceder a sus principales funciones, que son la importación de LEL, la visualización del LEL importado, la visualización de los Casos de Uso generados, la categorización de UC y Actores, la especificación de los factores técnicos, la especificación de los factores ambientales, el cálculo de UCP y el cálculo de horas-hombre. A su vez la herramienta tiene la capacidad de gestionar cuentas de usuario permitiendo que cada uno administre sus propios proyectos. Por último, la herramienta provee ayuda contextual y sugerencias en un recuadro ubicado a la derecha de la pantalla.

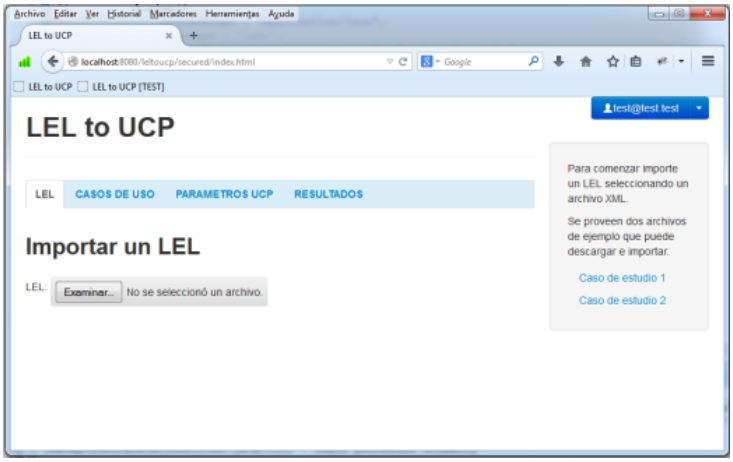

Ilustración 38 - Interfaz de Usuario de LEL to UCP

Funcionalidad Disponible en LEL to UCP

En esta sección se aborda la descripción de la funcionalidad provista por la herramienta, como acceder a dicha funcionalidad y que resultados esperar al ejecutar un acción determinada en el sistema.

Nuevo Usuario

Para poder interactuar con la mayor parte de la aplicación es necesario poseer una cuenta de usuario, dicha cuenta se crea desde la pantalla de creación de usuarios, para acceder a esta pantalla desde la página inicial de la aplicación se debe seguir el enlace “Nuevo Usuario”.

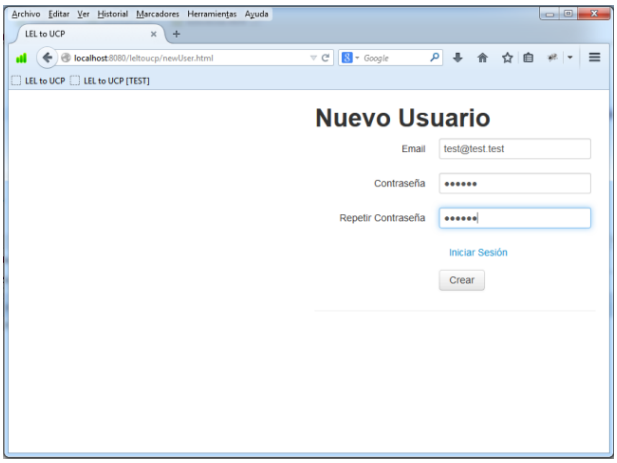

Ilustración 39 - Pantalla de Creación de Usuarios

Una vez en la pantalla de creación de usuario, se deben completar los campos, proporcionando una dirección de correo electrónico, y una contraseña, que debe estar compuesta por al menos seis caracteres. Luego de esto se debe presionar el botón “Crear”. Si los datos son correctos, el sistema redirigirá el navegado a la página de inicio de la aplicación. Si no es así, el sistema presentara los mensajes correspondientes el error encontrado. Los errores más comunes están dados por una longitud errónea de la contraseña o una dirección de correo electrónico invalida.

Una vez creado el usuario, la aplicación envía un correo electrónico de bienvenida a la dirección proporcionada por el usuario.

Inicio de Sesión

Para acceder al sistema se debe iniciar sesión completando los datos solicitados en la pantalla de inicio de sesión, estos datos corresponden a la cuenta de correo utilizada en la creación del usuario y la contraseña escogida.

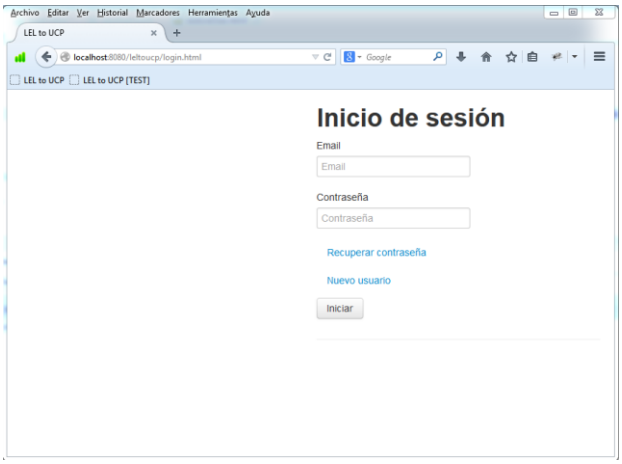

Ilustración 40 - Pantalla de Inicio de Sesión

Una vez ingresados dichos datos se debe presionar el botón “Iniciar”. Si los datos son correctos el sistema redirigirá el navegador a la página principal de la herramienta, en cambio si los datos proporcionados no corresponden con una cuenta de la aplicación, esta informara el error.

Recuperar Contraseña

La aplicación brinda la posibilidad de recuperar la contraseña proporcionada para una cuenta, para ello, desde la pantalla inicial de la aplicación se debe navegar el enlace “Recuperar contraseña”.

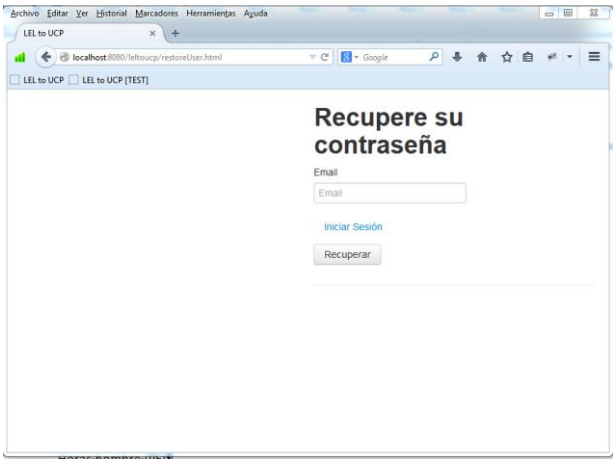

Ilustración 41 - Pantalla de Recuperación de Contraseña

Luego en la página de recuperación de contraseña se debe introducir la dirección de correo electrónico utilizada en la creación del usuario. Si existe una cuenta de usuario asociada a dicha dirección, la aplicación enviara un correo electrónico informando la contraseña perdida.

Cambiar Contraseña

El sistema permite modificar la contraseña de una cuenta de usuario, para ello, desde la pantalla principal, se debe acceder al menú y a la opción “cambiar contraseña”

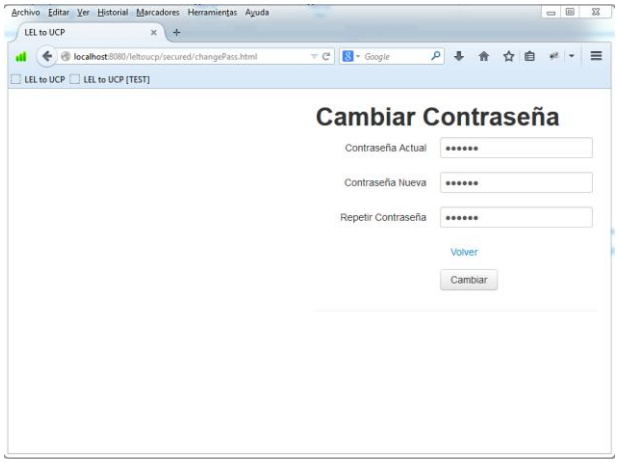

Ilustración 42 - Pantalla de Cambio de Contraseña

Luego en la página de cambio de contraseña se debe introducir la contraseña actual y la nueva contraseña, la que debe tener al menos seis caracteres. Si la operación se lleva a cabo de forma correcta, la aplicación redirigirá el navegador a la página de inicio de sesión y enviara un correo electrónico notificando del cambio realizado. Caso contrario el sistema informa los errores encontrados.

Importar un Léxico Extendido del Lenguaje

Una vez iniciada la sesión en el sistema, la pantalla principal permite importar un LEL al sistema, dicho LEL debe estar contenido en un archivo XML y respetar lo especificado en la sección Modelo de Datos. Para realiza la importación se debe clicar en el botón “Examinar…” y luego seleccionar un archivo a importar.

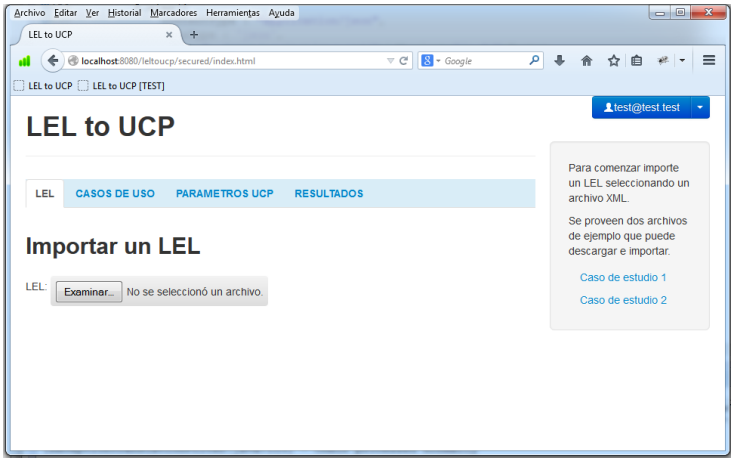

Ilustración 43 - Pantalla de Importación de LEL

Si la importación se lleva a cabo correctamente, la herramienta derivará los Casos de Uso partiendo desde LEL, inferirá las clases y entidades de base de datos involucradas en cada Caso de Uso, realizará una estimación inicial del peso del UC y presentará la pantalla de visualización de LEL en donde el sistema expone los términos importados. En caso contrario el sistema presenta los errores encontrados.

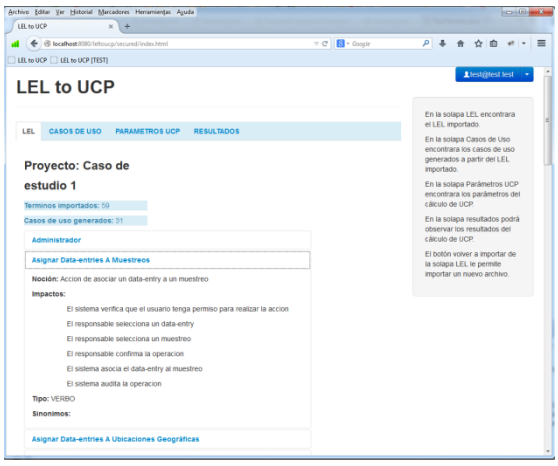

Ilustración 44 - Pantalla de Visualización de LEL

La herramienta proporciona dos archivos de ejemplo, a fin de poder ser evaluada por el usuario y que éste pueda adaptarse a su funcionamiento, diagramación, etc. Estos archivos están disponibles a través del recuadro de ayuda contextual ubicado a la derecha de la pantalla.

Una vez importado un LEL, este conformará un proyecto que estará disponible para su consulta y edición hasta que el usuario lo elimine, cabe destacar que los proyectos generados por un usuario solo son visibles, editables y eliminables por él mismo.

Ante cualquier error o equivocación del usuario, que requiera volver a importar el archivo procesado u otro archivo, en el menú ubicado en la esquina superior derecha se encuentra la opción “Volver a importar” que permite seleccionar un nuevo archivo.

Visualizar Casos de Uso Generados

Una vez importado un LEL, la aplicación genera los UC derivándolos a partir del LEL importado, la solapa “CASOS DE USO” presenta los Casos de Uso generados partiendo del LEL importado por el usuario, presentando la información correspondiente al nombre, objetivo, precondiciones, postcondiciones, actor principal y flujo normal de cada UC derivado.

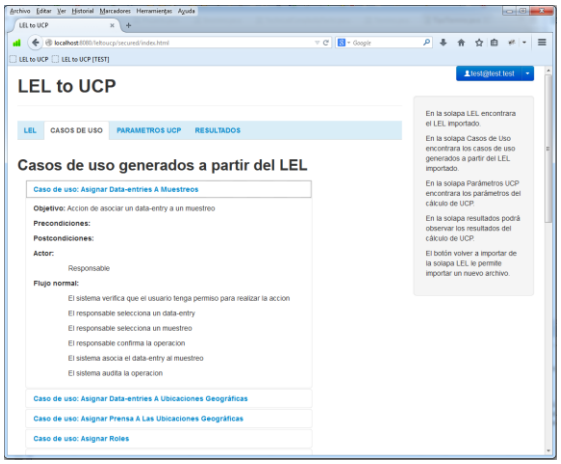

Ilustración 45 - Pantalla Casos de Uso

Configurar los Parámetros de Use Case Points

Una vez importado un LEL, la aplicación genera los UC derivándolos a partir del LEL importado, calcula el número de clases necesarias para implementar el UC y las entidades de base de datos que serán afectadas por el mismo. Partiendo de estos datos realiza una estimación de la complejidad mínima que tendrá la implementación de cada Caso de Uso. Además de esto, extrae los actores involucrados en los UC generados.

La información referente a los parámetros de UCP se presenta en la solapa “PARAMETROS DE UCP”, que debido a la cantidad de información disponible, se divide en cuatro subsecciones, a saber “UUCW”, “UAW”, “TFC” y “EFC”

La solapa “UUCW”, que corresponde con el cálculo de peso desajustado de los Casos de Uso, lista todos los Casos de Uso generados y por cada uno, informa el número de clases involucradas en su implementación y el número de entidades de base de datos afectadas por dicho UC. Correspondiente con lo mencionado y asociado a cada Caso de Uso, la herramienta presenta un combo box cuya finalidad es indicar la complejidad del Caso de Uso.

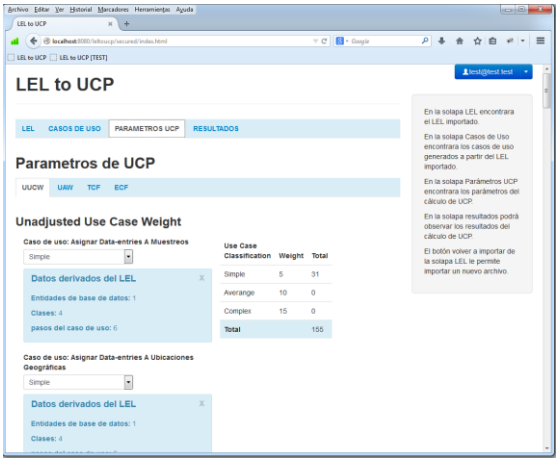

Ilustración 46 - Pantalla Parámetros de UUCW

El combo asociado a la complejidad del UC tiene por valor inicial la complejidad mínima estimada por la herramienta, pero debido a que ésta no puede inferir la complejidad de la UI necesaria en el Caso de Uso, es responsabilidad del usuario ajustar el valor inicial propuesto por la herramienta al valor que el usuario considere correcto. También y de considerarlo necesario, el usuario puede descartar el UC para su procesamiento posterior, seleccionado la opción “Discard”.

Al tiempo que los valores son editados, se lleva a cabo el recalculo del UUCW en la tabla que se encuentra a la derecha de la lista de UC permitiendo ver como las decisiones tomadas en un Caso de Uso afectan el valor total.

La solapa “UAW” hace lo propio con los pesos asignados a los actores involucrados en los Casos de Uso, para ello lista los actores y mediante un combo box correspondiente a cada uno permite indicar su complejidad.

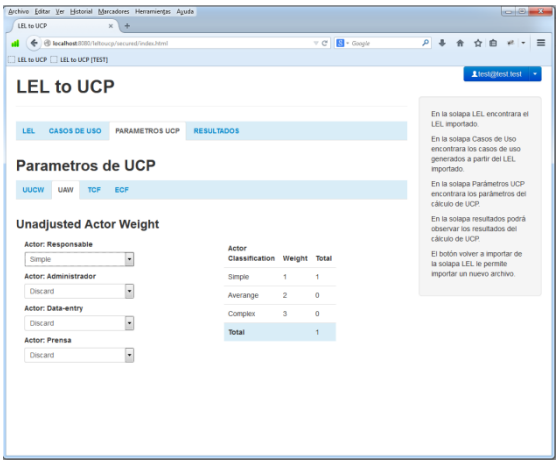

Ilustración 47 - Pantalla Parámetros UAW

Al igual que la solapa correspondiente al UUCW, ésta presenta una tabla en donde se totalizan los pesos asignados a los actores.

La solapa “TCF” permite indicar los valores correspondientes a los factores de complejidad técnica, totalizando el factor total “Total(TF)” y calculando e informando el “TCF” a medida que se cambian los factores que lo componen.

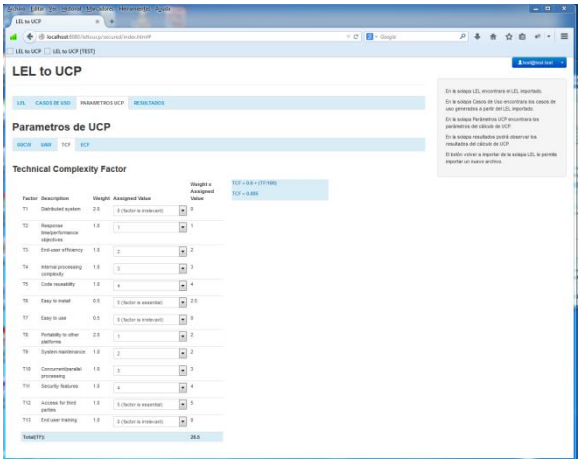

Ilustración 48 - Pantalla Parámetros TCF

La solapa "ECF" permite indicar los valores correspondientes a los factores de complejidad ambiental, totalizando el factor total "Total(EF)" y calculando e informando el "ECF" a medida que se cambian los factores que lo componen.

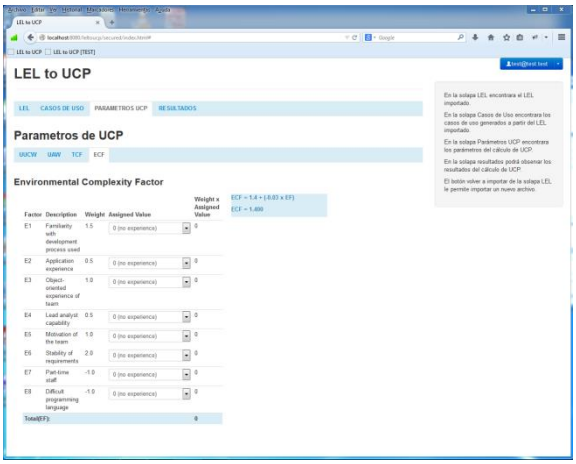

Ilustración 49 - Pantalla Parámetros ECF

Visualización de Use Case Points y Cálculo de Horas - Hombre

La ultima solapa, llamada "RESULTADOS", permite visualizar el cálculo de UCP obtenido de la información derivada del LEL y la provista por el usuario, además de esto presenta la posibilidad de calcular las Horas-Hombre necesarias, seleccionando el correspondiente factor de productividad "PF" en el combo box ubicado a la derecha de la pantalla.

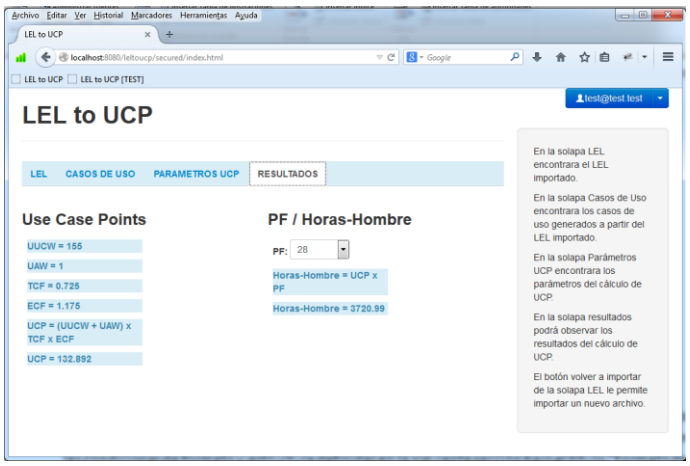

Ilustración 50 - Pantalla Resultados

Para facilitar el cálculo de Horas-Hombre el sistema realiza una estimación inicial del factor de productividad basándose en los factores de complejidad ambiental seleccionados, e inicializa el combo box correspondiente con dicho valor. Es responsabilidad del usuario conservar o redefinir este valor según su criterio.

Manejo de Proyectos

LEL to UCP brinda la funcionalidad para mantener cada LEL importado dentro de un proyecto de estimación accesible y eliminable por su creador. Para acceder a esta funcionalidad se debe clicar en la opción “Administrar proyectos” ubicada en el menú que se encuentra en la esquina superior derecha. Seguido de esto, la aplicación despliega un listado que contiene todos los proyectos generados por el usuario. Por cada elemento de la lista anterior, el sistema da la posibilidad de abrirlo mediante el botón con una carpeta como icono, o de eliminarlo con el botón que tiene una equis como icono.

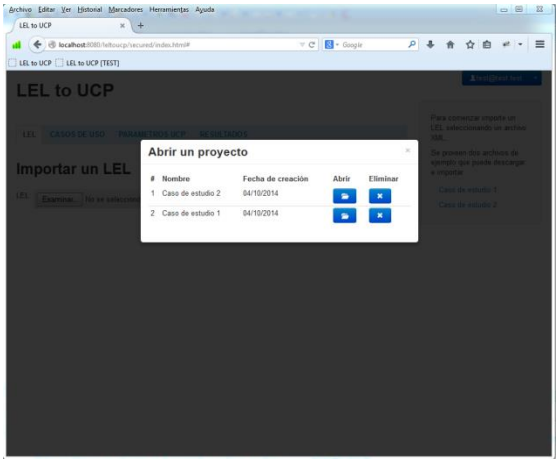

Ilustración 51 - Pantalla Administración de Proyectos

Como la aplicación persiste el estado de los proyectos en cada modificación realizada a los mismos, al abrir un proyecto, el usuario lo encontrara en el estado en que lo dejó la última vez que trabajó en dicho proyecto.

Palabras Finales

En este capítulo se introdujeron las tecnologías principales involucradas en el desarrollo de la herramienta LEL to UCP, para luego analizar su diseño arquitectónico, su modelo de objetos y datos, haciendo lugar a un apartado en donde se analizan las heurísticas involucradas en el procesamiento de un Léxico Extendido del Lenguaje. El resto del capítulo se dedicó a la guía de instalación de la herramienta y a su manual de usuario.

Al comienzo del presente capítulo se hizo una reseña de las principales tecnologías involucradas en la aplicación desarrollada, teniendo en cuenta las utilizadas en todas las capas de la misma, desde la capa de presentación hasta la capa de datos, dando lugar a las tecnologías de soporte al desarrollo y de contenedores de aplicaciones donde la herramienta se ejecuta.

En la sección dedicada a la arquitectura y diseño de la aplicación se presentó el modelo multicapas que se utilizó para implementar la herramienta, se evidenciaron los componentes principales de cada capa y como estos interactúan para brindar la comunicación necesaria entre las distintas capas. Luego se presentó el modelo de objetos de la aplicación, el modelo de datos donde se persiste toda la información generada por la herramienta y se introdujo el esquema XML que deberán respetar los archivos que contengan un Léxico Extendido del Lenguaje a importar. Para finalizar esta sección se analizó desde un punto de vista técnico las heurísticas descriptas en el capítulo anterior, haciendo énfasis en la criticidad que tienen en el proceso de inferencia de datos.

El capítulo prosigue con la guía de instalación de LEL to UCP, la que detalla el procedimiento para realizar la instalación de la herramienta, y a su vez brinda una guía para comprobar la correcta instalación y funcionamiento de la misma.

Por último se dedicó el final del capítulo al manual de usuario de la herramienta construida, a fin de facilitar su uso e invitar a nuevos usuarios a utilizarla, el manual consta de dos partes bien definidas, en la primera se introdujo la organización de la interfaz de usuario y en la segunda se explicó cada funcionalidad, como llegar a ella dentro del sistema, y que resultado esperar al utilizarla.

Conclusión

Partiendo de la idea de obtener una estimación del esfuerzo necesario para el desarrollo de un proyecto de software en las etapas tempranas de la elicitación de requisitos, se analizaron los elementos técnicos que dan soporte a la disciplina, buscando un conjunto de técnicas y conocimientos que se amalgamen en pos del objetivo mencionado, producto de este análisis, se definió un método, basado en un proceso de tipo pipeline, mediante el cual es posible obtener una estimación temprana del esfuerzo necesario para el desarrollo de un proyecto de software y se construyó una herramienta que da soporte a dicho método.

El método propuesto en el presente trabajo parte de la información recabada en la construcción del Léxico Extendido del Lenguaje de una aplicación, el que representa la abstracción del conocimiento que se tiene del dominio del problema que se intenta resolver, y cuya construcción ocurre en etapas tempranas de la elicitación de requisitos, contribuyendo, mediante su validación con las partes involucradas en el proyecto, al correcto entendimiento entre las mismas, por medio de la unificación de los conceptos que describen el dominio analizado.

Mediante la aplicación de transformaciones e inferencia de datos basada en heurísticas, se deriva del Léxico Extendido del Lenguaje, el conjunto de Casos de Uso posibles que definirán la aplicación a construir, los actores que los llevaran a cabo, el número de clases necesarias para implementar cada uno de ellos y el número de entidades de base de datos que serán afectadas por dichos Casos de Uso, elementos que se utilizan como entrada al cálculo de Use Case Points.

Teniendo en cuenta que la estimación que se está llevando a cabo por el método, se basa en información a priori volátil, tanto por la etapa del proceso en la que se encuentra, como por ser una tarea de análisis iterativa, y que existe información que no es capturada por el Léxico Extendido del Lenguaje, como es la complejidad técnica del proyecto, la composición del equipo de desarrollo que intervendrá en éste, los factores ambientales que influyen en dicho equipo, la complejidad de la interfaz de usuario de un determinado Caso de Uso. El método propuesto deja a criterio del usuario la definición de estos factores así como la inclusión o no de los actores y Casos de Usos generados en la etapa mencionada en el párrafo anterior, aumentando su flexibilidad y capacidad de expresión de una determinada situación.

Una vez definidos los parámetros a cargo del usuario, se utiliza la técnica de Use Case Points para estimar el esfuerzo necesario para el desarrollo de la aplicación en cuestión, técnica que se escogió a conciencia, ya que es ampliamente utilizada en la industria del software y la mayoría de los profesiones han tenido contacto con ella.

Por último el método propone una técnica de estimación del factor de productividad basada en los factores de complejidad ambiental, permitiendo de esta manera, aplicar dicho factor de productividad a los Use Case Points obtenidos, para lograr una estimación del esfuerzo requerido para el desarrollo de la aplicación en Horas-Hombre. Siguiendo los lineamientos anteriores en pos de la flexibilidad, el usuario puede modificar el factor de productividad si su criterio así lo determina.

Para facilitar la aplicación del método enunciado, se construyó una herramienta que le da soporte, se la nombró LEL to UCP, cuyo origen es evidente, y está desarrollada sobre una plataforma Web, para facilitar su acceso. La herramienta permite la importación de Léxico Extendido del Lenguaje por medio de un archivo XML que lo contenga, realiza todo el procesamiento necesario para derivar los

Casos de Uso, actores, numero de clases necesarias para la implementación de los Casos de Uso y número de entidades de base de datos afectadas por un Caso de Uso.

Una vez realizada la inferencia y transformación de datos, la herramienta presenta al usuario el Léxico Extendido del Lenguaje que se ha importado, los Casos de Uso que se han generado, los actores que llevan a cabo los Casos de Uso y los parámetros que se han inferido. En este punto entra en juego el criterio del usuario, que introduce los parámetros que no pueden inferirse, descarta Casos de Uso o actores, si esto fuera necesario, determina la complejidad ambiental y técnica que influirán sobre la estimación final flexibilizando la aplicación del método, e incluyendo el conocimiento adquirido por el usuario en la estimación del proyecto que está llevando a cabo.

Por último y con la información aportada por el usuario, la herramienta lleva a cabo el cálculo de Use Case Points y sugiere un factor de productividad, el que puede ser modificado por el usuario, obteniendo de esta manera la estimación del esfuerzo necesario para el desarrollo de la aplicación en Use Case Points y Horas-Hombre. Cabe destacar que la herramienta proporciona gestión de cuentas de usuario y que cada usuario tiene la capacidad de gestionar los datos importados y las estimaciones realizadas, lo que en el dominio de LEL to UCP se denominan proyectos, y pertenecen a su creador. Dichos proyectos se mantiene actualizados con cada modificación realizada por el usuario hasta que éste los elimine.

De ésta manera los profesionales que utilicen el Léxico Extendido del Lenguaje en su proceso de elicitación de requisitos, son grandes candidatos a la utilización del presente método, considerando la importancia que tiene una correcta estimación del esfuerzo necesario para el desarrollo de una aplicación y como esto impacta en la planificación del proyecto y el proceso en el que se enmarcará su construcción. Brindando a dichos profesionales un marco de trabajo que les permite obtener estimaciones del esfuerzo necesario para el desarrollo de una aplicación en etapas tempranas de la elicitación de requisitos, y observar cómo éstas varían dependiendo del grado de conocimiento que se tiene sobre el domino del problema que la aplicación viene a resolver.

La principal ventaja asociada al método propuesto y la herramienta que le da soporte, radica en la prontitud con la que se puede obtener una estimación del esfuerzo necesario para el desarrollo de una aplicación, sumado a que dicha estimación está basada en tecnologías ampliamente utilizadas, se logra reducir el esfuerzo necesario por parte del profesional que está realizando la estimación, al requerido para adaptar el Léxico Extendido del Lenguaje al formato de archivo requerido por LEL to UCP.

La facilidad de aplicación del método, lograda por medio de la herramienta construida, permite aprovechar el carácter iterativo de la elicitación de requisitos para la construcción de métricas basadas en las estimaciones obtenidas en cada paso de dicha iteración. Éstas métricas serán aplicables, durante el proceso de elicitación de requisitos, a la planificación del proyecto, ya que tener una estimación del esfuerzo necesario, es vital para la calendarización de las tareas del proyecto en cuestión así como también lo es para definir su viabilidad como proyecto de software dentro de una organización determinada.

La principal desventaja encontrada en el método desarrollado, está dada por la volatilidad de los datos utilizado como entrada, y que debido a que la inferencia de datos, depende de las propiedades constructivas del Léxico Extendido del Lenguaje, el que puede representar, o no, el dominio de una aplicación y puede, o no, apegarse a las reglas constructivas en las que se basa el método al momento de transformar e inferir datos, produciendo una gran volatilidad de las estimaciones obtenidas por resultado. Debido a esto es importante que la utilización del presente método se lleve a cabo con criterio y conciencia de parte del profesional y que se respeten los lineamientos indicados en éste trabajo a fin de minimizar las deficiencias expuestas.

Para continuar la línea de investigación y desarrollo seguida por el presente trabajo, se debería hacer un análisis estadístico de las estimaciones obtenidas a lo largo de distintos proyectos y los valores reales relativos al esfuerzo, a fin de mejorar los parámetros y factores de ajuste que utilizan las estimaciones en el método desarrollado, logrando de esta manera, estimaciones más precisas. Por otro lado también se debería amalgamar lo contenido en este trabajo con otras técnicas de estimación y/o datos de entrada, permitiendo la utilización de la herramienta en etapas avanzadas de la elicitación de requisitos, desarrollo y mantenimiento del producto, brindando de esta manera una herramienta integral, que saque provecho de todo el conocimiento acumulado a lo largo del proyecto, y lo utilice para mejorar su gestión.

Bibliografía

[1] J.C.S.P. Leite, *Application Languages: A Product of Requirements Analysis*. Rio de Janeiro, Brazil: Departamento de Informática, PUC-/RJ, 1989.

[2] J.C.S.P. Leite and A.P.M. Franco, "O Uso de Hipertexto na Elicitaçao de Linguagens da Aplicaçao," in *Anais de IV Simpósio Brasilero de Engenharia de Software*, 1990, pp. 134-149.

[3] K. Ribu, "Estimating Object-Oriented Software Projects with Use Cases," University of Oslo Department of Informatics, Master of Science Thesis 2001.

[4] G. Karner, "Metrics for Objectory," Linkoping University (LiTH-IDA-. Ex-9344:21), Linkoping, Sweden, Master Thesis 1993.

[5] L.M. Cysneiros and J.C.S.P. Leite, "Using the Language Extended Lexicon to Support Non-Functional Requirements Elicitation," in *Poceedings of the Workshops de Engenharia de Requisitos*, Buenos Aires, Argentina, 2001.

[6] D. Gil, D. A. Figueroa, and A. Oliveros, "Producción del LEL en un Dominio Técnico. Informe de un caso," in *Proceedings of the Workshops de Engenharia de Requisitos*, Rio de Janeiro, Brazil, 2000.

[7] IEEE Computer Society, *610.12-1990 IEEE Standard Glossary of Software Engineering Terminology*.: IEEE Computer Society, 1990.

[8] Roger S. Pressman, *Ingeniería del Software, un enfoque practico*, Quinta ed.: Mc Graw Hill, 2002.

[9] Ian Sommerville, *Ingeniería del software*, Séptima ed. Madrid: Pearson Educación S.A., 2005.

[10] Standish Group. (2013) The Chaos Report. [Online]. http://blog.standishgroup.com/

[11] Fred Brooks, *The Mythical Man-Month: Essays on Software Engineering*, 2nd ed.: Addison-Wesley Professional, 1995.

[12] Steve McConnell, *Software Estimation: Demystifying the Black Art*.: Microsoft Press, 2006.

[13] ISO/IEC, "ISO/IEC 14598-1 Parte 1: Visión general," ISO/IEC, Norma 1999.

[14] Stephen H. Kan, *Metrics and models in software quality engineering*, 2nd ed.: Adisson wesley, 2002.

[15] Ivar Hjalmar Jacobson, Patrik Jonsson, Magnus Christerson, and Gunnar Overgaard, *Ingeniería de Software Orientada a Objetos - Un acercamiento a través de los casos de uso.* Upper Saddle River, N.J.: Addison Wesley Longman, 1992.

[16] Alistair Cockburn, *Writing Effective Use Cases*.: Addison-Wesley, 2000.

[17] Gustav Karner, *Resource Estimation for Objectory Projects*.: Objective Systems SF AB, 1993.

[18] Allan J. Albrecht, "Measuring Application Development Productivity," *Proc. of IBM Applications Development Symposium*, pp. 14-17, Octubre 1979.

[19] Ivar Hjalmar Jacobson, G. Booch, and J. Rumbaugh, *The Objectory Development Process*.: Addison-Wesley, 1998.

[20] J.C.S.P. Leite and A.P.M. Franco, "A Strategy for Conceptual Model Acquisition," *Proceedings of the First IEEE International Symposium on Requirements Engineering*, pp. 243-246, 1993.

[21] Leandro Antonelli, "Identificación temprana de características transversales en el lenguaje de la aplicación capturado con el Léxico Extendido del Lenguaje," Facultad de Informática, Universidad Nacional de La Plata, La Plata, BA, Tesis Doctoral 2012.

[22] K.K. Breitman and J.C.S.P. Leite, "Ontology as a Requirements Engineering Product," in *Proceedings of the 11th IEEE International Conference on Requirements Engineering (RE)* , Monterey Bay, California, USA, 2003.

[23] T.R. Gruber, "A translation approach to portable ontology specifications," *Knowledge acquisition journal*, vol. volume 5, pp. 199-220, June 1993.

[24] G. Kaplan, G. Hadad, J. Doorn, and J.C.S.P. Leite, "Inspeccion del Lexico Extendido del Lenguaje," in *proceedings of the Workshops de Engenharia de Requisitos*, Rio de Janeiro, Brazil, 2000.

[25] Cenários e Léxicos. (2014, Enero) C&L Tool -. [Online]. http://pes.inf.puc-rio.br/cel/index_old.htm

[26] J.C.S.P. Leite, L.F. Silva, and K.K. Breitman, "C&L: Uma Ferramenta de Apoio à Engenharia de Requisitos," *Revista de Informática Teórica e Aplicada (RITA)*, vol. XII, no. 1, pp. 23-46, Junho 2005.

[27] Leandro Antonelli, Gustavo Rossi, Julio Cesar Sampaio do Prado Leite, and Alejandro Oliveros, "Buenas prácticas en la especificación del dominio de una aplicación," Lifia, Fac. de Informática, UNLP, La Plata, BA, Argentina, Paper 2013.

[28] R. Wirfs-Brock, B. Wilkerson, and L. Wiener, *Designing Object-Oriented Software*.: Prentice Hall, 1990.

[29] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns CD: Elements of Reusable Object-Oriented Software*.: Addison-Wesley Professional, 1994.

[30] IEEE, "Recommended Practice for Software Requirements Specifications - IEEE Std 830-1998," IEEE Computer Society, 1998.

[31] B. L. Kovitz, *Practical software requirements: A manual of content and style*. Greenwich: Manning, 1999.

[32] David Chelimsky et al., *The RSpec Book: Behaviour-Driven Development with RSpec, Cucumber, and Friends*.: Pragmatic Bookshelf, 2010.

[33] K. Wiegers, *Software requirements*.: Miscrosoft Press, 1999.

[34] L. Rosenberg, *Methodology for Writing High Quality Requirement Specifications and for Evaluating Existing Ones*. Greenbelt, MD, USA: Software Assurance Technology Center, NASA Goddard Space Flight Center, 1998.

[35] K. A Gunter and J. C. Mitchell, *Theoretical Aspects of Object-Oriented Programming*.: The MIT Press, 1994.

[36] B. Liskov and J. Wing, "A behavioral notion of subtyping," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 16, pp. 1811-1841, 1994.

[37] D. Bäumer, D. Riehle, W. Siberski, and M. Wulf, "The Role Object Pattern," in *Proceedings of PLOP 97*, Monticello, Illinois, USA, 1997.

[38] Leandro Antonelli, Gustavo Rossi, Julio Cesar Sampaio do Prado Leite, and Alejandro Oliveros, "Deriving requirements specifications from the application domain language captured by Language Extended Lexicon," , Buenos Aires, Argentina, 2012.

[39] The Eclipse Foundation. (2014, Junio) ATL a model transformation technology. [Online]. http://eclipse.org/atl/

[40] Leandro Antonelli, Gustavo Rossi, Julio Cesar Sampaio do Prado Leite, and Alejandro Oliveros, "Language Extended Lexicon Points: Estimating the Size of an Application using Its Language," in *22nd IEEE Requirements Engineering Conference*, Karlskrona, Sweden, 2014, pp. 25-29.

[41] Tim Berners-Lee, "Information Management: A Proposal," CERN, 1989.

[42] R. Fielding et al. (1999, Junio) ITEF.org. [Online]. http://www.ietf.org/rfc/rfc2616.txt

[43] W3C (World Wide Web Consortium). (2014, Septiembre) http://www.w3.org/. [Online]. http://www.w3.org/Protocols/

[44] The World Wide Web Consortium (W3C). (2014, Septiembre) Cascading Style Sheets home page. [Online]. http://www.w3.org/Style/CSS/

[45] The World Wide Web Consortium (W3C). (1996, Diciembre) Cascading Style Sheets, level 1. [Online]. http://www.w3.org/TR/CSS1/

[46] The World Wide Web Consortium (W3C). (Mayo, 1998) Cascading Style Sheets, level 2. [Online]. http://www.w3.org/TR/2008/REC-CSS2-20080411/

[47] The World Wide Web Consortium (W3C). (2011, Junio) Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. [Online]. http://www.w3.org/TR/2011/REC-CSS2-20110607/

[48] The World Wide Web Consortium (W3C). (2014, Septirmbre) CSS current work. [Online]. http://www.w3.org/Style/CSS/current-work

[49] The World Wide Web Consortium (W3C). (2014, Septiembre) Extensible Markup Language (XML). [Online]. http://www.w3.org/XML/

[50] The World Wide Web Consortium (W3C). (2008, Noviembre) Extensible Markup Language (XML) 1.0 (Fifth Edition). [Online]. http://www.w3.org/TR/REC-xml/

[51] The World Wide Web Consortium (W3C). (2004, Octubre) XML Schema Part 0: Primer Second Edition. [Online]. http://www.w3.org/TR/xmlschema-0/

[52] The World Wide Web Consortium (W3C). (2004, Octubre) XML Schema Part 1: Structures Second Edition. [Online]. http://www.w3.org/TR/xmlschema-1/

[53] The World Wide Web Consortium (W3C). (2004, Octubre) XML Schema Part 2: Datatypes Second Edition. [Online]. http://www.w3.org/TR/xmlschema-2/

[54] The World Wide Web Consortium (W3C). (2009, Diciembre) Namespaces in XML 1.0 (Third Edition). [Online]. http://www.w3.org/TR/REC-xml-names/

[55] Internet Engineering Task Force (IETF). (2014, Marzo) RFC - 7159. [Online]. http://tools.ietf.org/html/rfc7159

[56] ECMA. (2013, Octubre) ECMA-404 - The JSON Data Interchange Format. [Online]. http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

[57] ECMA. (1999, Diciembre) Standar ECMA-262. [Online]. http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

[58] The jQuery Foundation. (2014) jQuery. [Online]. http://jquery.com/

[59] The jQuery Foundation. (2014) jQuery UI. [Online]. http://jqueryui.com/

[60] Bootstrap. (2014) Bootstrap. [Online]. http://getbootstrap.com/

[61] ECMA. (2011, Junio) ECMAScript Language Specification. [Online]. http://www.ecma-international.org/publications/standards/Ecma-262.htm

[62] Oracle. (2014, Septiembre) Java. [Online]. http://www.java.com/

[63] Pivotal Software. (2014, Septiembre) Spring IO. [Online]. http://spring.io/

[64] Rod Johnson,.: Wrox Press, 2002, p. 750.

[65] Red Hat. (2014, Septiembre) Hibernate. [Online]. http://hibernate.org/

[66] The PostgreSQL Global Development Group. (2014, Septiembre) PostgreSQL. [Online]. http://www.postgresql.org/

[67] The Eclipse Foundation. (2014, Septiembre) The Eclipse Foundation. [Online]. http://eclipse.org/

[68] The Apache Software Foundation. (2014, Septiembre) Apache Maven Project. [Online]. http://maven.apache.org/

[69] Apache Software Foundation. (2014, Septiembre) Apache Subversion. [Online]. http://subversion.apache.org/