UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

TÍTULO: Edición colaborativa de escenarios para la derivación de modelos

AUTORES: Mariángeles Hozikian

DIRECTOR: Leandro Antonelli

CODIRECTOR: Alejandro Fernández

ASESOR PROFESIONAL: -

CARRERA: Licenciatura en Sistemas

Resumen

En esta tesis se ha desarrollado una herramienta que permite capturar el conocimiento del dominio en etapas tempranas del desarrollo de software a través de Escenarios. Si bien los Escenarios se describen textualmente, la herramienta brinda un soporte semántico para los mismos. De esta forma, a través de queries semánticos, es posible derivar nuevos modelos. En esta tesis, a modo de ejemplo se implementan varios modelos y reglas para realizar la derivación. Además de los Escenarios se utiliza un glosario conocido como Léxico Extendido del Lenguaje para capturar el conocimiento del dominio. Luego, el modelo que se deriva es el de casos de prueba los cuales se especifican a través del Task Method Model. Finalmente se implementan reglas para derivar los casos de prueba.

Palabras Clave

Escenarios, elicitación de requerimientos, léxico extendido del lenguaje, derivación de modelos, soporte automático, reglas de derivación, trabajo colaborativo, casos de prueba, mediawiki, consultas semánticas.

Conclusiones

El desarrollo de software implica capturar el conocimiento del dominio y transformarlo sucesivamente a través de diferentes modelos hasta llevarlo al código. Esta tesis propone un acercamiento para resolver esta situación. Si bien hay ciertas restricciones como por ejemplo la utilización del modelo de Escenarios con un template especifico, la herramienta puede ser modificada y extendida para soportar otros modelos. En la tesis se describe la implementación del caso de estudio realizado, con el fin de que sirva de guía.

Trabajos Realizados

Se ha desarrollado una herramienta para la derivación automática de modelos. La herramienta se basa en la plataforma Media Wiki con lo cual, es una herramienta web de caracter colaborativo. Además, se incorporó en la misma funcionalidad para soporte semántico. La herramienta ha sido sometida a un test de usabilidad. Y se ha instanciado un caso concreto de derivación de modelos. Si bien los Escenarios son el elemento central de la herramienta, la misma ha sido extendida como caso de estudio con el Lexico Extendido del Lenguaje, Casos de Prueba y reglas para derivar estos últimos.

Trabajos Futuros

Se plantean dos ejes de trabajos futuros. Por un lado, profundizar la utilización de procesamiento de lenguaje natural con el fin de mejorar el reconocimiento de texto para transformarlo en conceptos ontológicos. Por otro lado, mejorar el mecanismo para extender los modelos y las reglas de derivación. Ofrecer una interface más cercana al usuario final y no tanto del desarrollador.

Fecha de la presentación: Febrero 2020

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

ÍNDICE DE LISTADOS

Capítulo 1 - INTRODUCCIÓN

En los inicios de las ciencias de la computación, los programas que se desarrollaban resolvían problemas muy simples, tales como operaciones matemáticas básicas. En aquellos comienzos este tipo de problemas no requerían mayor análisis. Tampoco se consideraba la posibilidad de que dicho desarrollo sufra cambios en el futuro, ni se preveía que otros informáticos pudieran incorporarse en etapas avanzadas para modificar lo ya desarrollado.

El rápido crecimiento de la demanda por software, su complejidad y la falta de técnicas para que el desarrollo de sistemas pueda ser validados, condujo a que en la década de 1960 ocurra la llamada "Crisis del Software". Ésta crisis se debió a varios motivos, tales como: los proyectos no terminaban a tiempo, no se ajustaban al presupuesto inicial, baja calidad en el software generado, el software no cumplía las especificaciones. Esto provocó la necesidad de formalizar los métodos para el proceso de desarrollo.

Es así que la Ingeniería del Software surgió para hacer frente a los problemas presentados. Según Pressman (Pressman 2002) es "*Una disciplina que integra métodos, herramientas y procedimientos para el desarrollo de software de computador".* Es decir, es una disciplina dentro de la informática que intenta focalizarse en el proceso de desarrollo de software. La misma se considera desde que nace la idea o necesidad, la puesta en marcha y el mantenimiento. Esta subdisciplina establece también las pautas a seguir para que durante el desarrollo se minimicen tiempo, esfuerzo y costo y se maximice la calidad del software (Dijkstra, 1972).

Existen distintos modelos de desarrollo que abarcan dos grandes grupos. El primero son los modelos estructurados en etapas. Algunos de estos modelos son: en cascada, en espiral, iterativo e incremental, etc. (Sommerville, 2005). Todos ellos tienen en común la obtención y análisis de requerimientos (requisitos) como el inicio de la planificación. El otro gran grupo son los modelos de desarrollo ágil o metodologías ágiles (Agile Manifiesto, 2019), las cuales son la tendencia actual, utilizan estrategias menos estructuradas. Esto se debe por poseer menos reglas, aunque la elicitación y análisis de requerimientos siguen siendo factores fundamentales para la producción de software.

Los Requerimientos fueron definidos por la IEEE como (IEEE, 1990):

1. Condición o capacidad requerida por el usuario para resolver un problema o alcanzar un objetivo
2. Condición o capacidad que debe satisfacer o poseer un sistema o una componente de un sistema para satisfacer un contrato, un estándar, una especificación u otro documento formalmente impuesto
3. Representación documentada de una condición o capacidad como en 1 o 2.

La ingeniería de requisitos es una de las primeras fases del ciclo de vida del software en la que se producen especificaciones a partir de ideas informales por parte de los stakeholders. La meta de la ingeniería de requerimientos es obtener una especificación de requerimientos de software correcta y completa.

Si en la etapa de requerimientos, por ser una de las iniciales, se producen errores, las consecuencias son importantes. Esto es debido a que se arrastran al resto de los productos que se construyen a partir de los requerimientos. Una especificación incorrecta de requerimientos puede ocasionar errores ocultos dentro del sistema (Mizuno, 1983) que repercuten en efecto dominó sobre las fases posteriores del desarrollo de un sistema. Esto quiere decir que el problema está en el punto de partida, aunque puedan aparecer problemas posteriormente. Si la especificación es incorrecta, el diseño será incorrecto, porque sin importar que esté bien diseñado, será un diseño que resuelve un problema que no es real.

Mizuno los llama errores ocultos porque no se detectarán durante el ciclo de vida hasta que el software sea utilizado por el usuario y note que la aplicación no se ajusta a sus necesidades. En ese momento, se habrá realizado mucho trabajo (especificacion, diseño, codificacion, testing, manuales de usuario, etc.) y la corrección de estos errores ocultos implica realizar nuevamente todo el trabajo. Por el contrario, los errores surgidos en diseño o programación, sobre una especificación correcta, son corregibles, a traves de técnicas de inspección.

Según Boehm (Boehm y Papaccio, 1988), si se produce un error en la descripción de los requisitos y se corrige en mantenimiento, la corrección podría llegar a costar 200 veces más comparado con el costo de corregir la misma durante la etapa de requisitos. Este cálculo nace del re-trabajo que implica corregir requerimientos y los productos construidos a partir del mismo (diseño, código, pruebas, etc.). Como primer punto hubo recursos destinados a la elicitación que, por haber sido mal planteada, deben volver a analizarse. En segundo lugar, el tiempo y esfuerzo empleado para el desarrollo debe repetirse con la nueva especificación. Si bien no necesariamente se descarta todo el trabajo hecho, sí se requiere un re-trabajo con un costo relativo que puede llegar a crecer de 1 a 200 veces su esfuerzo. Como siguiente paso deben repetirse las pruebas ya realizadas, ahora con las nuevas especificaciones. Posiblemente sea necesario realizar una capacitación sobre la nueva funcionalidad, ya que la resuelta anteriormente no resolvía lo que se creía haber entendido. Estos pasos, son a grandes rasgos, los que deben replicarse descartando un trabajo anterior, en el que se pudo haber consumido un gran número de recursos, llámese personas, tiempo, dinero.

Los modelos de desarrollo estructurados se caracterizan por ser estrictos. Los mismos suelen estar compuestos por muchas etapas, fases bien definidas, fechas críticas, requisitos bien identificados y poco tolerantes a cambios. Esto podría provocar que el cumplimiento de tales pautas sea difícil de llevar a cabo en la práctica, produciendo, por ejemplo, retrasos en las entregas establecidas. Estos modelos intentan imponer disciplina al proceso de desarrollo de software para volverlos predecibles. Para alcanzarlo se basan en la planificación propia de otras ingenierías. La gran desventaja de esta característica es que el desarrollo de software es el producto de una actividad intelectual resuelta por seres humanos, por lo cual es que resulta difícil realizar estos procesos de forma predecible. A finales de los 90 aparecen las metodologías ágiles, con el intento de reducir la complejidad de las metodologías estructuradas. Este nuevo tipo de metodologías impulsan generalmente a una gestión de proyectos. Además de promover el trabajo en equipo, la organización y responsabilidad propia que cualquier metodología propone, le da un valor agregado: las metodologías ágiles son adaptativas (no predictivas) y orientadas a las personas (no a procesos). Es importante destacar que, si bien estas metodologías no son estructuradas, los requerimientos siguen siendo muy importantes. De esta forma se puede generar un grupo de buenas prácticas de

ingeniería de software que brindan una entrega rápida de software de alta calidad, que alinea el desarrollo con las necesidades del cliente y los objetivos de la empresa (Agile Manifiesto, 2019). Para cumplir con estas cualidades, en marzo de 2001, Kent Beck junto con otros ingenieros propuso el "Manifiesto Ágil", compuesto por cuatro valores. En la Tabla 1-1 se muestran estos cuatro valores comparando las metodologías tradicionales versus las metodologías ágiles.

Valoraciones de metodologías tradicionales	Valoraciones de metodologías ágiles
Procesos y herramientas	Individuos y sus interacciones
Documentación exhaustiva	El software funcionando
Negociación contractual	Colaboración con el cliente
Seguimiento de un plan	Respuesta al cambio

TABLA 1-1. VALORES DE METODOLOGÍAS TRADICIONALES VS. ÁGILES.

El ciclo de desarrollo en metodologías ágiles es iterativo e incremental permitiendo hacer pequeñas entregas llamadas "incrementos". Cada iteración se puede considerar como un sub-proyecto en el que las actividades de análisis de requerimientos, diseño, implementación y testing son llevadas a cabo con el fin de producir un subconjunto del sistema final. El proceso se repite varias veces produciendo un nuevo incremento en cada ciclo hasta que se elabora el producto completo.

No todas las metodologías ágiles proporcionan especificaciones de procesos, etapas y roles, pero por ejemplo Scrum y XP si lo hacen. Scrum tiene un artefacto llamado "Product Backlog" que recoge los requerimientos y estimaciones y los prioriza. Los desarrolladores sólo pueden implementar requerimientos que estén en el Product Backlog. Existen tres roles: Scrum Master quien gestiona los cambios, Product Owner que representar a los stakeholders y Team quienes llevan a cabo el desarrollo. En cada sprint (entre una y cuatro semanas de duración) se entrega un incremento utilizable, con las características definidas en el Product Backlog. Estas definiciones son determinadas en cada reunión de Sprint Planning, en las cuales el Product Owner identifica qué se ha resuelto, y qué se resolverá en el próximos Sprint.

Más allá de la técnica de elicitación de requerimientos, no tendría sentido tanto trabajo si el resultado no fuera volcado a una especificación de requerimientos. Se debe proporcionar una representación del software a desarrollar.

Se entiende que la elicitación de requerimientos es el proceso de adquirir todo el conocimiento relevante necesario para producir un modelo de requerimientos de un dominio del problema (Loucopoulos y Karakostas, 1995). Según estos autores, además de los usuarios finales, se pueden considerar otras fuentes de información, como literatura sobre el dominio o estándares nacionales o internacionales. Entre las principales técnicas de elicitación propuestas por ellos, se pueden mencionar:

- Originadas en el usuario.
- Análisis de objetivo y meta.
- Escenarios.
- Análisis de formularios.
- Lenguaje natural.
- Reúso de requerimientos.
- Análisis de tareas.

Sommerville (Sommerville, 2005) por su parte manifiesta que "*la meta del proceso de ingeniería de requerimientos es crear y mantener un documento de requerimientos del sistema*". Este autor define como parte del proceso general de la elicitación de requerimientos, cuatro subprocesos: estudio de viabilidad, obtención y análisis de requerimientos, especificación de requerimientos y validación de requerimientos.

Independientemente del modelo de procesos utilizado, la meta siempre será la especificación, que puede ser tan formal como el standard de la IEEE830 1998 o algo más simple, como pueden ser la User Stories usadas en metodologías ágiles (SCRUM).

Hoy en día podemos encontrar avances informáticos en la mayoría de los aspectos de nuestras vidas y a distintos niveles. En algunos casos suele ser indispensable y en otros pueden ser simplemente una ayuda para gestionar o trazar procesos de la vida cotidiana. Lo que es indiscutible es que estos avances siempre están vinculados a algún dominio de aplicación específico y tienen un objetivo en particular.

Es por ello que un analista o un ingeniero además de ser competente en su especialidad, necesita comprender dicho dominio al mismo nivel que un usuario. No deben descartarse los riesgos de mal interpretar determinados léxicos ya que ambas partes pueden darle su propia connotación. Es por ello también que el proceso de elicitación de requerimientos se considera una tarea social. Esto quiere decir que debe alimentarse de la colaboración y participación de los diferentes stakeholders quienes pueden aportar de manera constructiva una comprensión unificada del léxico del dominio de aplicación independientemente de sus propias perspectivas (Antonelli, Rossi y Oliveros, 2016).

La elicitación de requerimientos es clave para la comprensión del problema a resolver y los stakeholders deben manejar un lenguaje natural. Por un lado, los miembros del equipo de desarrollo necesitan descripciones más precisas y formales, ya que resulta muy difícil lograr especificaciones que sean adecuadas y útiles para las dos partes. El lenguaje natural carece de la precisión y formalidad necesaria, mientras que los usuarios no están familiarizados con los lenguajes formales. Por un lado, hablaremos de LEL (Léxico Extendido del Lenguaje), que permite describir el lenguaje del dominio de la aplicación utilizando el lenguaje natural. Por otro lado, hablamos de Escenarios que introducen al contexto de la solución que se va a desarrollar, además de definir el problema previendo su solución dentro de los objetivos de negocio.

Sea cual fuera el modelo de desarrollo que se vaya a implementar, es fundamental que, para poder planificar una entrevista, un cuestionario, un taller, entre otros; se debe conocer qué es lo que se preguntará, analizará o estudiará. Además, se debe identificar a los actores clave, que pueden aportar al

desarrollo del proyecto, por ello es importante realizar estas actividades preliminares (Antonelli, Rossi, Leite y Oliveros, 2013).

El Método-V es un modelo que vincula las etapas iniciales del ciclo de vida del proceso de desarrollo con etapas de prueba. Resume los pasos principales que hay que tomar en las correspondientes entregas de los sistemas de validación.

La parte izquierda de la V representa la corriente donde se definen las especificaciones del sistema. La parte derecha de la V representa la corriente donde se comprueba el sistema (contra las especificaciones definidas en la parte izquierda). El vértice representa la corriente de desarrollo como se muestra en la Ilustración 1-1.

La corriente de especificación consiste principalmente de:

- Especificaciones de requerimiento de usuario
- Especificaciones funcionales
- Especificaciones de diseño

La corriente de pruebas, por su parte, suele consistir de:

- Calificación de instalación
- Calificación operacional
- Calificación de rendimiento

La etapa de prueba es la variable de ajuste cuando los tiempos y el presupuesto de un proyecto están en crisis. Lograr que las pruebas, total o parcialmente, se realicen después de cada modificación asegura la calidad de los requisitos del software. Para ello es necesario tener bien definida la especificación de requerimientos o productos iniciales. Si bien existen muchas herramientas que dan soporte a los procesos de ingeniería de software, también es importante contar con herramientas que automaticen el trabajo. Habiendo estudiado las bondades de cada modelo, sus ventajas, la utilidad fundamentada en cada una de las etapas de la vida del software ha surgido la necesidad de crear una herramienta que integre estos conceptos. La forma que se plantea para la integración de modelos es a través de una derivación de los mismos. Esto es importante ya que a partir de la elicitación de requerimientos se obtiene un primer modelo del software. Con este conocimiento se puede plantear el diseño, a partir de éste el código y a partir de éste último las pruebas. En cada etapa es necesario poner atención en distintos aspectos. Cada una de ellas recibe como entrada la información procesada en la etapa anterior y tiene como salida la información transformada para la etapa siguiente. Es decir que cada modelo transforma la información dependiendo de la etapa del ciclo de vida del software y que cada modelo se construye a partir del anterior. Por ejemplo, Antonelli (Antonelli, Rossi, Leite y Oliveros, 2012) ofrece técnicas para obtener especificaciones de requerimientos a partir de conceptos de dominio, símbolos definidos en el diccionario LEL. Estas técnicas consisten en la derivación de LEL a User Stories y de LEL a Casos de Usos.

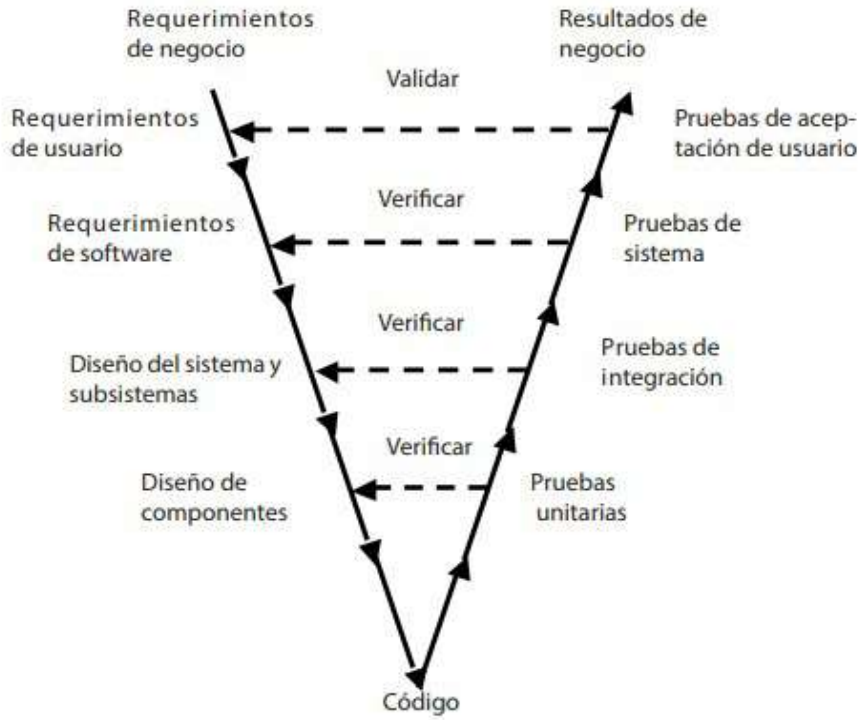


ILUSTRACIÓN 1-1. MODELO DE CICLO DE VIDA EN "V".

Una vez elegidos los modelos de desarrollo, establecidas las técnicas de elicitación de requerimientos y la formalidad que se usará en cada una, es importante elegir herramientas que den soporte a dichos procesos. Dada la importancia que tiene la buena especificación de requerimientos en etapas tempranas del ciclo de vida del proceso de software, consideramos como modelo inicial los Escenarios, a partir de los cuales se podrán derivar otros modelos.

El objetivo de esta tesis es desarrollar una herramienta que, a partir de un modelo permite obtener otros modelos. Dada la importancia de los requerimientos y de capturar el conocimiento de las etapas iniciales, la herramienta utiliza como modelo principal a los Escenarios. Si bien los Escenarios se describen textualmente, la herramienta brinda un soporte semántico para los mismos. De esta forma, a través de queries semánticos, la herramienta brinda la posibilidad de derivar nuevos modelos. En esta tesis, se definio el modelo de casos de pruebas implementados a traves del Task/Method Model. Además, se definieron queries semánticos para derivar los casos de prueba a partir de los Escenarios. Cabe señalar que tanto los casos de prueba como las reglas de derivación, se definieron a modo de ejemplo, y la herramienta es adaptable para definir y derivar otros modelos.

La tesis está organizada de la siguiente manera: El capítulo 2 describe el background. El capítulo 3 describe la arquitectura de la herramienta, la cual es extensible para otros modelos. El capítulo 4 describe el diseño, es decir, cómo se implementaron los casos de pruebas y la derivación de los mismos. El capítulo 5 muestra el manual de uso, como así también una prueba de usabilidad. El capítulo 6 discute las conclusiones.

Capítulo 2 - BACKGROUND

En esta sección se describen los conceptos que serán los puntos de partida para la derivación de modelos. Utilizamos Escenarios como modelo a partir del cual se derivarán otros modelos. Otro concepto que es necesario ahondar es el de Glosario. Utilizamos particularmente los Léxicos Extendidos del Lenguaje, este modelo requiere un análisis más profundo del universo de estudio (dominio de aplicación). Se da una definición del mismo y se explica mediante ejemplos. Seguidamente se desarrolla el concepto de Casos de Prueba, su importancia y sus ventajas y se especifica particularmente el modelo de Task/Method, el cual se utiliza para especificar los casos de prueba. Por último, que describe la importancia del trabajo colaborativo y sus ventajas. Esto es asi ya que la herramienta en si, provee capacidades colaborativas.

ESCENARIOS

Existen distintos modelos para representar los requerimientos funcionales de un sistema. El principal objetivo en esta etapa de elicitación es recabar información de lo que el cliente final espera o necesita del sistema. Además, es necesario asegurarse un buen entendimiento y colaboración entre todos los involucrados a la definición de requerimientos.

Los analistas entienden, modelan y analizan el dominio de aplicación. Los clientes (y usuarios finales, quienes podrían ser los mismos o no) validan si los puntos de vista de los analistas son los correctos (Leite, Rossi, Balaguer, Maiorana, Kaplan, Hadad y Oliveros, 1997).

Suelen elegirse modelos gráficos para lograr esta comunicación, que utilizan diagramas y dibujos para describir las interacciones. Uno de los ejemplos más comunes es la utilización de Casos de Uso, definidos por lenguaje UML, sin embargo, son descripciones poco específicas, para producir un modelo adecuado, sería necesario describir la funcionalidad con más detalle.

La ingeniería de requerimientos comienza detallando la generación de requerimientos y su dependencia con aspectos sociales que abarcan la construcción del software. Los escenarios se usan a lo largo de todo el proceso de desarrollo de software, evolucionando hasta las etapas de testeo.

Además, nos proveen un medio de comunicación entre todos los involucrados en el proyecto de desarrollo de software más certero y específico, sin perder de vista el manejo de un lenguaje común entre ellos y de fácil entendimiento. Es decir que sirven de interface entre el medio ambiente y el sistema. También podemos decir que sirven para entender la aplicación y su funcionalidad. Cada escenario describe una situación específica de la aplicación centrando la atención en su comportamiento (Leite, Hadad, Doorn y Kaplan, 2000). También son útiles a la hora de conocer el problema, unificar criterios, ganar compromiso con clientes y usuarios, organizar los detalles y entrenar a nuevos participantes.

Un escenario es una descripción parcial y concreta del comportamiento de un sistema en una determinada situación. Se dice que es parcial, porque no necesita describir todas las características de las entidades involucradas, sólo se describe aquello que está relacionado con un comportamiento particular del sistema analizado. A pesar de estar acotados a un determinado comportamiento, describen todo el

contexto que involucra a esa actividad: recursos del sistema, objetivos de los usuarios, contexto social en que se desarrolla, entidades involucradas.

Un método muy útil y aplicado en este trabajo es basarse en el vocabulario propio del universo de discurso. El mismo refleja las palabras más utilizadas en el lenguaje del dominio de la aplicación mediante los llamados Léxicos Extendidos del Lenguaje. Estos léxicos son un tipo de Glosario creado con el fin de registrar ese vocabulario específico y su semántica dejando de lado para una fase posterior la resolución de los problemas en sí.

Dependiendo de la etapa en la que se utilicen, puede tener diferentes objetivos. Se destacan tres objetivos principales. El primero es capturar los requerimientos, el segundo es proveer un medio de comunicación entre los stakeholders, y por último proveer un soporte para la trazabilidad.

Se habla de etapas del ciclo de vida del proceso de desarrollo de software, y se hace énfasis en que los escenarios nacen a partir de la elicitación de requerimientos (que sería una de las primeras etapas). Así mismo cabe destacar que la documentación sufre modificaciones y ajustes en todo el ciclo de vida. Esto es porque en la medida que el desarrollo avanza, los stakeholders se reformulan y re-ven los requerimientos. Esta característica es propia de procesos de desarrollo iterativas e incrementales.

En el proceso de elicitación de requerimientos, el ingeniero de software se puede enfrentar a dificultades. La primera de ellas es poder establecer una buena comunicación con el usuario. De ser esto posible, los errores y omisiones pueden ser detectados a tiempo evitando la “catarata de errores” (Mizuno, 1983) y su propagación durante el resto del proceso de desarrollo. La segunda dificultad se basa poder garantizar el seguimiento de los requerimientos durante todo el ciclo de vida del software (Leite y Oliveira, 1995). Esto quiere decir que se puedan mantener los orígenes de los requerimientos, para su posterior creación de casos de prueba a partir de los requerimientos.

Cada uno de los escenarios describen un momento específico de la aplicación. Esto permite comprender el problema de manera refinada y poder analizarlo parcialmente. Otra ventaja de esta herramienta es que se describe con un lenguaje natural, esto permite una fácil validación con el usuario final. Si bien cada escenario es una descripción parcial del problema, existe relación semántica con el resto de los escenarios (Booch, 1994).

Los escenarios pueden escribirse de diferentes maneras, dependiendo del autor. Estas definiciones son descripciones informales, ya que cada autor puede darle su impronta. Algunos ejemplos pueden ser mediante narrativa textual, videos, prototipos o maquetas. El grado de importancia va a depender del estado de avance del proceso de desarrollo o de la importancia que le dé el usuario al problema.

Según (Leite, Rossi, Balaguer, Maiorana, Kaplan, Hadad y Oliveros 1997), un escenario se puede definir con los siguientes atributos: un "título" que identifica el escenario, un "objetivo" o un objetivo a alcanzar, un "contexto" donde el objetivo podría alcanzarse. El contexto indica un punto de inicio para el escenario. Uno o más "recursos" (objetos físicos relevantes o información que debe estar disponible en el escenario), "actores" que son los agentes que realizan las acciones que se describen como un conjunto de "episodios". Los actores podrían ser personas que interactúan con el sistema, como así también otros sistemas. Esta definición es la que tomamos como modelo para nuestro desarrollo.

Para ejemplificar la integración y derivación de modelos se toma un dominio de aplicación en particular. El dominio elegido en cuestión es de agricultura, la producción de tomates bajo invernadero. En

la Tabla 2-1 se muestra un ejemplo de escenario en el dominio de aplicación especificado. El mismo es "Decidir el tipo de labor cultural". Esta decisión debe tomarse ya que, al comienzo de la temporada de producción de tomate, el agricultor debe tomar múltiples decisiones importantes. Algunas de ellas son, cuándo comenzar, qué variedad plantar y densidad de plantación. Éstas pueden mostrar variabilidad en el tiempo. Por otro lado, hay decisiones que reflejan el estilo de trabajo, por ejemplo, el sistema de capacitación, estilo de conducción y poda de tallos, que muestran menos variabilidad en el tiempo.

Título	Decidir tipo de labor cultural.
Contexto	Producción en invernadero. Producción de tomate. La decisión se realiza una vez que la planta ha alcanzado el tamaño adecuado para comenzar a realizar labores culturales.
Objetivo	Decidir el tipo de labor cultural, en particular sistema de conducción, sistema de capacitación y poda.
Actores	Agricultor, Líder de producción.
Recursos	Técnica de capacitación, sistema de conducción, labor cultural, medida, estrategia, políticas de poda.
Episodios	Agricultor eligen un sistema de conducción. Agricultor deciden técnica de capacitación. Líder establece las políticas de poda a aplicar. Líder escribe un procedimiento de acuerdo al estándar que describe el sistema de conducción, la técnica de capacitación y las políticas de poda.

TABLA 2-1. ESCENARIO: DECIDIR TIPO DE LABOR CULTURAL.

LÉXICO EXTENDIDO DEL LENGUAJE

Durante el proceso elicitación de requerimientos es indispensable la comunicación entre los distintos stakeholders. El ingeniero de requerimientos maneja un vocabulario más bien formal y técnico. Por otro lado, el usuario final maneja un lenguaje natural, el lenguaje habitual de su medio ambiente. Si la comunicación entre los distintos actores se expresa con diferentes léxicos, la probabilidad de fracaso aumenta. Este fracaso puede significar requerimientos incorrectos e incompletos, en consecuencia, un alto costo de reingeniería, sumado a clientes insatisfechos. La especificación de requerimientos debe resultar igualmente de útil para ambas partes.

Con el objetivo de establecer una comunicación clara y eficaz, el ingeniero de software debe homogeneizar el vocabulario. Una forma de lograr esto entre todos los actores es utilizando "Language Extended Lexicon" (LEL) (Leite, 1989).

Los LEL pueden construirse paralelamente al proceso de construcción de escenarios, y un modelo puede alimentarse con el contenido del otro. Ambos en conjunto ayudan a comprender el dominio de aplicación. A su vez, mientras los escenarios van creciendo en contenido, los LEL deben ser actualizados para mantener la consistencia e ir mejorándolos.

Los LEL no sirven para describir la especificación de requerimientos, sino que son un tipo glosario que definen terminología propia de un dominio de aplicación. LEL se define como un modelo que permite representar y documentar, con tecnología hipertextual un conjunto de símbolos que representan el lenguaje de la aplicación, sin necesidad de entender el problema a resolver con el sistema. Es decir que consiste en "*una descripción de los términos significativos del macrosistema, acotando el lenguaje externo con el uso de símbolos definidos en el mismo LEL y a su vez, minimiza el uso de símbolos externos al lenguaje de la aplicación*" (Leite y Franco, 1990). Un LEL está formado por un conjunto de símbolos, los cuales pueden ser palabras o frases que identifican el lenguaje particular de la aplicación y que el usuario utiliza con más frecuencia o que son relevantes para el dominio del problema.

Para definir un LEL, como primer paso se genera una lista con todos los símbolos identificados. Durante la elicitación, el ingeniero de software intenta entender y definir el significado de cada símbolo. La semántica de cada uno se representa con una o más nociones y cero o más impactos. La noción indica qué es el símbolo, puede ser una frase o un único párrafo, si se expande más en esta descripción, probablemente no esté bien definido tal símbolo y sea necesario desdoblarlo. El otro campo es el impacto, que indica cómo repercute el símbolo en el sistema, es decir los efectos de su uso u ocurrencia en la aplicación. El impacto suele usarse como medida del efecto de un incidente o requerimiento sobre el sistema, lo que permite establecer una prioridad, junto con otros parámetros que no son motivo de estudio en este trabajo.

Por lo tanto, cada símbolo tiene un "nombre" que lo identifica, una "noción" (la denotación del símbolo) y un "impacto" (connotación del símbolo) que lo describen. En la descripción de las nociones e impactos existen dos principios (Leite y Franco, 1993) (Hadad, Kaplan, Oliveros y Leite, 1997), que se deben cumplir simultáneamente:

-El principio de vocabulario mínimo indica que, al describir una noción o un impacto, esta descripción debe minimizar el uso de símbolos externos al lenguaje de la aplicación.

-El principio de circularidad define que las nociones e impactos deben ser descritos usando símbolos del propio lenguaje.

Una vez identificados todos los símbolos y descritas las nociones e impactos de cada uno siguiendo estos principios, deben clasificarse en cuatro grupos: Sujetos, Objetos, Verbos y Estados.

En Tabla 2-2, Tabla 2-3 y Tabla 2-4 se muestran ejemplos de representación de símbolos LEL que pertenecen al dominio de aplicación elegido para este trabajo.

El proceso de construcción de LEL consta de seis actividades que pueden desarrollarse paralelamente. Las mismas son: identificar fuentes de información; identificar símbolos; clasificar símbolos; describir símbolos; verificar el LEL y validar el LEL. Las cuatro primeras se refieren a la construcción del LEL

propiamente dicho, las últimas dos generan la retroalimentación de las demás para permitir las correcciones necesarias. Estas validaciones son necesarias para evitar ciertos errores comunes, que no suelen ser identificados en una primera observación. Algunos de ellos pueden ser símbolos mal descritos, símbolos incompletos, clasificación incorrecta, símbolos omitidos, falta de referencia a otros símbolos o mal uso de símbolos (Leite, Hadad, Doorn y Kaplan, 2000).

Símbolo	Sujeto: Agricultor
Noción	Persona que cultiva tomates en un lote.
Impacto	El agricultor participa en la determinación de las labores culturales. El agricultor siembra los tomates. El agricultor cultiva los tomates. El agricultor controla la temperatura del invernadero.

TABLA 2-2. SUJETO AGRICULTOR.

Símbolo	Objeto: Invernadero
Noción	Lugar donde crecen los tomates en ambiente controlado.
Impacto	El agricultor planta tomates en un invernadero. El agricultor controla la temperatura del invernadero.

TABLA 2-3. OBJETO INVERNADERO.

Símbolo	Verbo: Control de temperatura en el invernadero
Noción	Acción de monitorizar la temperatura de manera de mantenerla dentro de cierto rango de valores.
Impacto	El agricultor monitorea la temperatura. El agricultor ventila el invernadero para descender la temperatura. El agricultor cierra las ventanas del invernadero para incrementar la temperatura.

TABLA 2-4. VERBO CONTROL DE TEMPERATURA EN EL INVERNADERO.

Casos de Prueba

Dado que no todos los requisitos pueden ser cubiertos con los escenarios ni los casos de uso, es que existen los casos de prueba, los cuales describen el conjunto de actividades, que al ejecutarse validará la funcionalidad de la aplicación. Es decir que los casos de prueba validan los requerimientos funcionales del sistema, con pruebas llamadas "pruebas de caja negra" ya que validan *qué* debe probarse y no *cómo*. Basado en el modelo-v podemos decir que lo ideal es comenzar a realizar las pruebas lo antes posible, abarcando todas las etapas del desarrollo (inicio, elaboración, construcción). Sin embargo, en lo cotidiano resulta difícil determinar la línea que delimita el comienzo de las pruebas.

En el momento de definir las normas de modelado es necesario reconocer dos actividades que requieren diferentes modelos, uno basado en la planificación y otro basado en la ejecución. La principal diferencia radica en que la planificación construye estados hipotéticos del mundo, por lo tanto, es posible cancelar la aplicación de tareas. Por otro lado, la ejecución modifica los estados del mundo de aplicación por lo tanto no pueden ignorarse los estados finales de las tareas, más aún los estados no deseados.

Entonces el objetivo principal de la especificación de los casos de prueba para la derivación es que a partir de los modelos de planificación se podrá representar un modelo de ejecución. La definición formal de casos de prueba se puede decir que es un conjunto de condiciones que bajo el criterio del analista quien determinará que los requisitos de las aplicaciones se cumplen de manera parcial o completamente exitoso. Es preciso definir para determinar este criterio, entradas conocidas y salidas esperadas establecidas previamente a la ejecución de las pruebas, conocidas como pre-condiciones y post-condiciones respectivamente.

Debido a que los casos de prueba validan los requerimientos funcionales de una aplicación, debe existir al menos un caso de prueba por requerimiento. A su vez podría haber más de uno para el mismo requerimiento. Para un mismo caso de uso podrían definirse dos casos de prueba, para distintos escenarios posibles. Por ejemplo, un caso de uso especificado para que un usuario se loguee en un sitio. A partir del mismo se desprenden dos escenarios posibles, con lo cual podrían existir dos o más casos de prueba, uno para aquel usuario que tiene cuenta registrada en dicho sitio y otro cuando no tiene cuenta registrada. Ambos casos de prueba se derivan de un mismo caso de uso, sin embargo, prueban cosas diferentes desde el punto de vista de los requerimientos funcionales. Podrían estar probando el mismo código desde una perspectiva distinta que internamente no importa, ya que son pruebas de caja negra.

Cada caso de prueba está compuesto por un conjunto de pasos a seguir, según la complejidad del programa a probar, teniendo en cuenta todas las funcionalidades que el mismo es capaz de realizar. Cada uno de estos pasos se compone de una acción y de un resultado esperado. Para que un caso de prueba sea exitoso, todos sus pasos deben haber sido exitosos. Basta que un paso no se cumpla para que el resultado general del caso de prueba sea fallido. Debe conocerse el paso que falló, esto servirá a los desarrolladores y analistas a detectar el error para corregirlo y generar una nueva versión de la aplicación con los fallos arreglados.

Se dice que un caso de prueba tiene un buen diseño si se eligen casos con mayor probabilidad de encontrar errores, con el menor esfuerzo posible. Algunas condiciones que deben cumplirse para diseñar un buen caso de prueba son:

- Especificar entradas.
- Especificar todas las salidas posibles y sus características.
- Especificar requisitos especiales de los procedimientos.
- Dependencias entre distintos casos de pruebas.
- No deben ser redundantes.

Para realizar las pruebas existen varias formas, podrían ser manuales o automatizadas. Para el primer caso, la prueba depende exclusivamente del accionar de un humano. Debido a que muchas veces la complejidad de la funcionalidad aumenta, es que se crearon la prueba automática. Estas brindan un margen de error menor cuando se prueban funcionalidades muy precisas además de ahorrar mucho tiempo en esta etapa de la vida del software. Sin embargo, en este tipo de pruebas, si bien se gana en tiempo y exactitud, se pierde la visión del usuario final, que no es un factor menor, ya que éste será el destinatario de la aplicación que se desarrolla.

Esta disyunción no hace referencia a que un método sea mejor que el otro, sino que dependiendo lo que se espera de las pruebas, es la estrategia que se elige utilizar dentro del mismo proyecto. En la Tabla 2-5 se hace una comparación entre estos tipos de pruebas destacando las ventajas y desventajas de cada método.

TASK METHOD

En este trabajo se abarca una estrategia particular de abordar la generación de casos de prueba mediante un modelo llamado Task/Method.

Este paradigma de modelado se basa en considerar al conocimiento como tareas y proviene de la rama de la inteligencia artificial. Más precisamente de la adquisición y el modelado de conocimientos, sistemas expertos, sistemas basados en conocimiento entre otros (Trichet y Tchounikine, 1999).

La principal ventaja de este paradigma es tener una forma declarativa para expresar el conocimiento. Esto permite que pueda ser procesado con facilidad por herramientas de procesamiento de tareas, etc. (Camilleri, Soubie y Zalaket, 2003).

El modelo de task/method a su vez se compone de dos submodelos: el modelo de dominio y el modelo de razonamiento. El modelo de dominio describe los objetos del mundo utilizados (directa o indirectamente) por el modelo de razonamiento.

	Pruebas Manuales	Pruebas automáticas
Ventajas	Permite conocer para qué sirve el software.	Útil cuando el caso de prueba va a ejecutarse muchas veces.
	Útil cuando se ejecuta una o dos veces.	
	Identifica qué problemas soluciona la aplicación.	Resultados rápidos.
	Le da una visión a la aplicación desde el punto de vista del usuario final.	Eficaz para comprobar partes del software muy específicas y de alta granularidad.
	Un ser humano podrá encontrarse con situaciones inesperadas.	Evita errores humanos cuando las pruebas son repetitivas y tediosas, como comparación de datos y chequeos repetitivos.
	Permite reproducir situaciones compatibles con el mundo real.	Para pruebas de alta fiabilidad, son capaces de detectar errores invisibles al ojo humano.
Desventajas	Errores humanos pueden hacer perder la fiabilidad de los resultados.	No considera factores de usabilidad o amigabilidad con el usuario.
	Ineficiente para tareas repetitivas.	Requiere invertir tiempo en el diseño y desarrollo de las pruebas, por lo que no es eficiente si no van a ejecutarse varias veces.

TABLA 2-5. COMPARACIÓN DE PRUEBAS MANUALES VS. AUTOMÁTICAS.

El modelo de razonamiento describe cómo se realiza una tarea y utiliza dos primitivas de modelado: Tareas (Task): es una transición entre dos estados y se describe:

- *Name*: Nombre de la tarea.
- *Par*: Lista de parámetros tipificados en el encabezado de la tarea.
- *Objective*: Estado esperado de la tarea.
- *Methods*: Lista de métodos para llevar a cabo la tarea.

Método (Method): describe la forma de realizar una tarea. Está compuesto por los siguientes campos:

- *Heading*: Tarea realizada por el método.
- *Prec*: Precondiciones que deben cumplirse para poder llevar a cabo la tarea.

- *Effects*: Efectos generados por la ejecución exitosa del método. Los objetivos de las tareas necesariamente pertenecen a los efectos de los métodos. Cada objetivo de una tarea genera un estado del mundo esperado, mientras que los efectos pueden no hacerlo.
- *Control*: orden en que deben ejecutarse las subtareas. Es decir, la secuencia de instrucciones, los condicionales "if… then" e "if...then ...else" y las iteraciones "while".
- *Subtask*: conjunto de subtareas.

Siguiendo el ejemplo del caso de uso descrito anteriormente, la traducción de escenario a caso de prueba utilizando Task/Method Model quedaría definido como muestra laTabla 2-6. Si bien a priori pareciera una traducción sintactica la definición en Task/Method Model permite la ejecución del cÓdigo traducido. De esta forma, con la ejecución de cada paso es posible obtener dos resultados posibles, que el paso se ejecute satisfactoriamente o que el paso falle. La combinación de todas las fallas posibles, dan lugar, a los distintos casos de prueba. Por ejemplo, una situación podria ser que directamente falle el primer paso: *elegir(agricultor, sistema_conduccion)*. Otra situación sería que el primer paso resulte exitoso, sin embargo, que falle el segundo paso *decidir(agricultor, tecnica_capacitacion)*. Otra situación podría ser que el primer y segundo paso sean exitosos y que falle el tercero *establecer(lider, politica_de_poda).* Y asi siguiendo.

Método: M1
Tarea: elegir(labor_cultural)
Control:
elegir(agricultor, sistema_conduccion);
decidir(agricultor, tecnica_capacitacion);
establecer(lider, politica_de_poda);
escribir(lider, procedimiento_completo_de_decision);

TABLA 2-6. CASO DE PRUEBA ESPECIFICADO CON TASK/METHOD MODEL.

TRABAJO COLABORATIVO

Desde la concepción de la web de la mano de Tim Berners-Lee quien la definió: "La Web es más una creación social que técnica. La diseñé para un efecto social —para ayudar a las personas que trabajan juntas— y no como un juguete técnico. El objetivo último de la Web es apoyar y mejorar nuestra existencia en la telaraña mundial. Nos agrupamos en familias, asociaciones y empresas. Desarrollamos la confianza a grandes distancias, y la desconfianza a la vuelta de la esquina" (Berners-Lee y Fischetti, 2001).

El trabajo colaborativo es una práctica grupal con el fin de reducir trabajo utilizando el esfuerzo conjunto de varios individuos que tienen un mismo objetivo. Dividir el trabajo entre varias personas que colaboran en lugar de depender de una sola reduce el esfuerzo de concretar la tarea y enriquece los resultados (Lowry A. y Lowry M., 2004). Es decir que la construcción del conocimiento nace como un proceso individual y luego se incorpora en la sociedad mediante mecanismos cíclicos.

Cuando las personas utilizan el conocimiento o las ideas de los demás, cuando se basan en el trabajo de los demás, cuando comparten sus ideas o cuando contribuyen con parte de un proyecto, no se

considera la colaboración como tal. Para que sea considerado trabajo colaborativo es fundamental que el objetivo de cada persona sea el objetivo del grupo. Por lo tanto, es necesario brindar un equilibrio entre el aporte individual de cada involucrado y el objetivo grupal. Este equilibrio puede mejorar la posibilidad de colaboración, ordenar y organizar la participación (Gutwin, Greenberg y Roseman, 1996), ya que se vuelve una tarea compleja informar qué está pasando con el resto de los usuarios. Es por eso que estos procesos deben ser capaces de determinar en qué orden se realizan las actividades colaborativas y los protocolos que definen las acciones concretas, sin dejar de lado los roles y permisos de cada uno.

Para que la colaboración sea efectiva, los usuarios tienen que estar informados sobre las acciones que los otros participantes realizan en el contexto. Además, es necesario saber cómo esas acciones afectan el entorno de trabajo general y cuáles son los métodos, procesos o lenguajes que guían la construcción del conocimiento y de la funcionalidad. Estas características mejoran la calidad de los productos y se vuelven fundamentales cuando se deben llevar a cabo prácticas repetitivas y reusables.

De esta forma entendemos que la Web facilita en gran medida la construcción de conocimiento. Para que esto pueda lograrse de manera efectiva, primero es necesario entender que el conocimiento nace en primer paso como una construcción personal en la que el individuo necesita organizar, expresar y justificar sus propias ideas al resto de los involucrados.

Con esta premisa, el trabajo colaborativo no deriva entonces en la suma de información de muchos participantes. Tampoco garantiza que la simple interacción entre individuos genere conocimiento. Más bien se basa en que la elaboración y reformulación conjunta entre todos los involucrados permite, no sólo el propio aprendizaje obteniendo resultados individuales, sino facilitando el aprendizaje de los demás y aportando al objetivo grupal. Desde esta perspectiva puede verse que el rol de cada uno no se centra en confrontar e imponer sus ideas, sino aportar al conjunto.

La construcción del conocimiento puede explicarse con el siguiente conjunto de principios:

- El aprendizaje es un proceso constructivo interno donde toma importancia los conocimientos previos.
- El grado de aprendizaje tiene que ver con el nivel del desarrollo cognitivo.
- El aprendizaje se facilita con la interacción con otros.

La etapa para describir la construcción del conocimiento puede verse como un proceso cíclico donde se destacan dos grupos principales. El primero tiene que ver con el proceso personal en que un individuo basado en sus conocimientos previos puede generar su propia interpretación a determinada problemática y construir su propio enfoque personal. El segundo grupo nace a partir de la necesidad del individuo en querer transmitir su enfoque personal. Para ello, mediante la exposición a otras personas de estos enfoques, se discuten las alternativas y se argumentan las propias ideas dando como resultado un conocimiento compartido. La formalización de este conocimiento se implanta en objetos culturales que sirven como base para ser fundados nuevamente en el entendimiento personal de un individuo, es aquí donde el ciclo vuelve a comenzar.

Tecnología para dar soporte a la construcción del conocimiento

Cuando se habla de tecnología generalmente se lo asocia a una disciplina donde el foco está en la estructuración, la racionalidad y las reglas. Por otro lado, se tiene que la colaboración es un proceso que se realiza entre personas, relaciones, discusiones y subjetividades. Por lo tanto, resulta cuestionable la posibilidad de predecir soluciones para la colaboración. En este punto es donde podemos plantearnos herramientas capaces de unir estos conceptos es pos de aprovechar las ventajas de cada paradigma.

Muchas herramientas Web ayudan a la generación de conocimiento colaborativo. En ellas el foco se pone en la sociabilización y colaboración, como son las páginas del estilo de Facebook o Wikipedia entre otras, las cuales se denominan generalmente Web Social y son características de la llamada web 2.0. Estas webs son muy diferentes a las de la generación anterior a la que se conocía como web 1.0, donde los usuarios sólo eran consumidores de contenido. En esta nueva versión de la web los usuarios pueden retroalimentarse, perfeccionar y mejorar los contenidos que crean y comparten.

Si bien el número de páginas utilizadas para generar contenido son muchas, en general tienen un propósito particular. Entre los ejemplos más comunes puede ser compartir fotos o vivencias con familiares y amigos, acortar distancias entre los mismos, promover un negocio personal o divulgar distintas artes, también existen aquellas donde el propósito principal es académico, brindar información actual o de interés general.

Como caso concreto de sitios que implementan el trabajo colaborativo para enriquecer el conocimiento colectivo, nacen las Wikis (del hawaiano wiki, que quiere decir “rápido”). Estas son páginas en las cuales los usuarios pueden crear, actualizar, modificar y borrar la información creada, tanto por uno mismo como la de otros usuarios (dependiendo de las reglas de privacidad que contenga cada wiki).

El origen de las wikis es atribuido a Howard Cunningham, un programador norteamericano de patrones de programación, el cual creó WikiWikiWeb, la primera wiki aún vigente dedicada a sus proyectos personales y a patrones de diseño.

El formato que ofrecen las wikis permite la navegación a través de enlaces a diferentes páginas, lo que resulta ágil para usuarios con conocimientos básicos de uso de páginas web. En el presente trabajo se plantea un formato más estricto para limitar al usuario a cargar la información con la estructura particular de los modelos de representación para escenarios, símbolos LEL y casos de prueba (bajo la forma Task/Method). Para ello es necesarios instalar la extensión PageForms característica para la creación de páginas de una wiki con información semántica.

La web semántica

A partir de las ventajas derivadas del concepto de trabajo colaborativo se introduce otro concepto tecnológico: La Web Semántica. La misma es una extensión de la web común, donde los datos además de mostrarse, permite relacionarlos entre sí, automatizar e integrar aplicaciones mediante el uso de información enriquecida de significado que permite una mejor cooperación entre humanos y máquinas (Berners-Lee, Hendler y Lassila, 2001). Esto se basa en dos ideas principales: el uso de etiquetas semánticas

(lo que implica una separación estricta entre el contenido y la estructura de los documentos), y la creación de aplicaciones de software inteligente, lo que se conoce con el nombre de agentes, los cuales son capaces de procesar estas etiquetas a nivel semántico (Hendler, 2001).
Según Tim Berners-Lee: "La Web fue diseñada como un espacio de información, con el objetivo de que debería ser útil no solamente para la comunicación entre humanos, sino también para que las máquinas pudieran participar y ayudar. Uno de los principales obstáculos a ello ha sido el hecho de que la mayoría de la información presente en la web está diseñada para el consumo humano [...]. En vez de tratar el problema de inteligencia artificial de entrenar a las máquinas a comportarse como las personas, el enfoque de la Web Semántica es desarrollar lenguajes para expresar la información de una manera accesible y procesable por las máquinas." (Berners-Lee, 1998)

Los agentes son entidades de software que recolectan, filtran y procesan información de forma semiautónoma, los cuales cumplen el papel de intermediarios entre el usuario y las fuentes de información distribuidas en toda la red. De esta forma la web semántica alcanzará su potencial cuando esté poblada por un gran número de agentes que sean capaces de recoger información de diferentes fuentes distribuidas, procesarla e intercambiar resultados con otros agentes.

Según (Berners-Lee, Hendler y Lassila, 2001) "*The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation.*"

Las wikis

Las Wikis son herramientas las cuales su principal virtud radica en brindar al usuario la posibilidad de cargar y modificar información en la web sin necesidad de tener conocimiento en lenguajes de programación mediante interfaces amigables e intuitivas. Para realizar las ediciones y carga de páginas se realiza directamente desde el navegador, con muy pocas restricciones, por medio de un lenguaje de WikiText. Esta característica posibilita la participación activa y colaborativa de muchas personas. Generalmente no es necesaria la revisión de un moderador de contenido, a diferencia de los sistemas convencionales que los usuarios comunes acceden a información que gestiona un proveedor de contenido en particular.

La principal importancia que da valor a las wikis como herramientas potentes es el uso de enlaces entre páginas. La forma de identificar unívocamente una página es con un título. Si este título es encerrado entre corchetes dobles dentro del cuerpo de una página, esa palabra se convierte en un enlace a la página correspondiente. Por ejemplo, si se tiene la página "Scenario" y en su contenido se encierra entre corchetes dobles la palabra [[Actor]], si existe una página que se llame "Actor", será redirigida a dicha página. Generalmente las URI se arman con este título lo que sirve de utilidad para usar este recurso fuera de la página, además de generar coherencia y ordenamiento en el contenido del sitio.

En las wikis tradicionales existen tres representaciones para cada página:

- El código que puede ser editado por cualquier usuario, que está almacenado en el servidor local.

- Una plantilla que define disposición y elementos comunes en todas las páginas.
- Código HTML provisto por el servidor en tiempo real cada vez que la página es solicitada.

Algunas wikis permiten agregar etiquetas HTML propias para definir estilos personalizados, aunque dependiendo del motor de wiki que se use permitirá esto o algunos motores más complejos, soporte para imágenes, cuadros y fórmulas. En lo posible trata de evitarse el uso de lenguajes del estilo JavaScript o CSS, para que el desarrollo de páginas no sea difícil de aprender para usuarios que no tienen conocimiento de lenguajes de programación.

No necesariamente una wiki es una página web, en algunos casos pueden encontrarse versiones portables para uso de escritorio, incluso pueden llevarse en un USB que tenga LAMP o WAMP instalado.

Existen diferentes softwares que abarcan todas estas características. Para este trabajo en particular optamos por MediaWiki.

MediaWiki (Mediawiki) es un software para wikis, programado en PHP usando motor de base de datos MySql, es software libre bajo la licencia GNU General Public Licence (GPL), por lo tanto, se puede decir que utiliza arquitectura cliente/servidor. Es usado por Wikipedia y otros proyectos de la Fundación Wikimedia. Cada página usa el formato WikiText, de manera que los usuarios sin conocimiento de HTML o CSS pueden editar las páginas fácilmente.

Además de las características propias de las wikis genéricas, en MediaWiki también se puede destacar una estructura de enlaces entre distintas páginas. Esto permite navegabilidad, si el enlace no existe da la posibilidad de distinguirlo, por ejemplo, escribiendo en rojo. Clickeando este enlace permite la creación del mismo, mediante un editor. Otra ventaja destacable es el control de cambios permitiendo volver a versiones anteriores, sabiendo el usuario y el momento en que se realizó el mismo. Esto es posible gracias a que cada edición es reflejada en la base de datos (sin eliminarla). Esta característica es muy útil ya que la posibilidad de que cualquier usuario pueda alimentar de información la wiki conlleva a que las fuentes de dicha información no sean confiables, incluso malintencionadas.

Capítulo 3 – ARQUITECTURA DE LA HERRAMIENTA

En este capítulo se presenta la arquitectura elegida para el desarrollo de la herramienta. Además, se detallan las extensiones de MediaWiki instaladas y para qué se usa cada una, justificando su utilidad. Seguidamente nos introducimos en el lenguaje WikiText y su sintaxis.

La principal característica que aprovechamos de la wiki es el contenido semántico. Es decir, meta información que describe el significado, la estructura de las páginas contenidas en el sitio y cómo se relacionan entre sí a partir de un modelo de datos lógico que es abstracto para el usuario. El modelo que se plantea contiene Escenarios embebidos de símbolos LEL, ambos son cargados como páginas independientes mediante una extensión de MediaWiki, el motor utilizado para montar la wiki. Esta extensión se llama PageForms y es la que nos provee, mediante lenguaje de marcado propio, la posibilidad de darle significado semántico a los elementos que se desean cargar, utilizando las plantillas suministradas por la extensión. Estructuramos los Escenarios y los símbolos LEL de manera que exista relación semántica entre ellos. Además, se plantean reglas de derivación específicas para dar como resultado otro modelo: los casos de prueba.

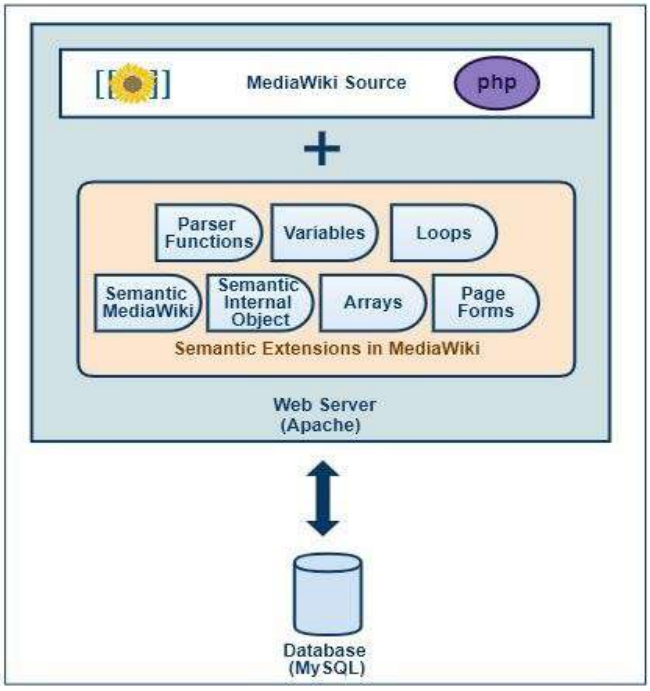


ILUSTRACIÓN 3-1. ARQUITECTURA DE LA APLICACIÓN.

En este trabajo se presentan las ventajas en cuanto a las necesidades de la solución planteada, de la elección de crear una herramienta del tipo wiki. Seguidamente se justifica esta decisión. Luego se detalla qué tipo de arquitectura provee el motor de MediaWiki y cómo necesitamos configurarlo para adaptarlo a nuestra necesidad. Se explica cómo es capaz de disponer de componentes y plantillas que nos permiten

dotar al sitio de significado semántico de acuerdo a nuestro modelo planteado, mediante la instalación de las llamadas “extensiones”.

Se elige MediaWiki dado que provee las cualidades apropiadas para representar información semántica y para la generación de conocimiento de manera colaborativa, podemos decir que reúne las características necesarias para desarrollar una herramienta muy potente. En primer lugar, nos permite representar escenarios y símbolos de léxicos extendidos del lenguaje. A su vez podemos representar las relaciones semánticas entre ambos modelos. En segundo lugar, crear una interfaz amigable e intuitiva que permite agregar, editar, listar y buscar escenarios y símbolos LEL. Como tercer punto definir reglas de derivación, es decir reglas que, a partir de los modelos recién mencionados, se pueda obtener de manera automática un tercer modelo: casos de prueba. A continuación, se detallará las técnicas usadas para desarrollar la aplicación que cumpla con estos requisitos.

A continuación, se muestra un diagrama de la arquitectura de la herramienta en la capa interna como se muestra en la Ilustración 3-1. Sobre un motor de wiki existente provisto se realizan las configuraciones. Para configurar la wiki a nuestras necesidades lo hacemos a partir de un archivo “playbook.yml” que se ejecuta con el programa Ansible (Ansible, 2018). MediaWiki está escrito en lenguaje de programación PHP. Está escrito para la plataforma LAMP (Linux, Apache, MySQL, PHP), aunque puede funcionar sobre otras plataformas, siempre y cuando soporte PHP. Usa principalmente servidores de base de datos MySQL. Para el desarrollo de nuestra herramienta es necesario además instalar Extensiones de Mediawiki, las cuales permiten que nuestra wiki sea más robusta y útil para diferentes fines.

Las extensiones utilizadas para nuestro desarrollo son: Semantic Media Wiki, Semantic Internal Object, Arrays, Page Forms, Parser Functions, Variables y Loops. Como se describe en la Tabla 3-1.

Semantic Media Wiki	Relaciona distintos componentes o datos del sitio por medio de propiedades semánticas.
Semantic Internal Object	Representa objetos complejos y permite asociar propiedades a más de un valor.
Arrays	Permite manipular conjuntos de datos y realizar operaciones sobre ellos.
Page Forms	Define plantillas para la creación de páginas por medio de formularios de carga.
Parser Functions	Agrega funciones que complementan las existentes en mediawiki. Requerida para utilizar Loops.
Variables	Permite definir variables dentro de una página para ser usadas y modificadas en cualquier momento, ya sea dentro de la página o dentro de los templates. Requerida para utilizar Loops.
Loops	Proporciona funciones para realizar iteraciones.

TABLA 3-1. EXTENSIONES MEDIAWIKI UTILIZADAS.

Semantic Media Wiki

Define un framework para el almacenamiento de datos que luego pueden ser consultados haciendo que la wiki se comporte en forma similar a una base de datos. Existen otras extensiones que se pueden instalar sobre SMW, las cuales, en conjunto, hacen que MediaWiki sea transformada de una wiki regular a una base de datos colaborativa. En Semantic MediaWiki cada dato se construye a partir de ternas: un sujeto, un predicado y un objeto. Un ejemplo de una terna puede ser: Scenario1 - Its context is - Context1, "Scenario1"es el sujeto; "Its context "es el predicado, y "Context1"es el objeto. En SMW, el predicado se conoce como propiedad, y el sujeto es siempre la página en la que el objeto se almacena. La manera más simple de representar esta terna sería ir a la página en la wiki denominada Scenario1, editarla, e ingresar lo siguiente:

```
…the context is [[Its context is:: Context1]]...
```

Listado 3-1. Sintaxis para setear propiedad.

Con la sintaxis de Listado 3-1 es la forma con la que MediaWiki almacena links. La diferencia es que están los dos puntos ::. *Its context is,* que representa el predicado, o propiedad, de la terna. Lo que se muestra en la página depende del tipo de la propiedad. Los tipos de datos soportados son los primitivos (strings, integer, boolean) y además añade varios con formatos especiales predefinidos como Fechas, URLs o Coordenadas Geográficas entre otros. Un tipo de dato particular, y muy utilizado, es el tipo Page o Página, que indica que el valor almacenado en la propiedad sería un enlace a otro artículo (existente o no) dentro de la wiki. SMW permite además acotar los valores posibles de una propiedad a una lista definida por el usuario, siempre y cuando respete el tipo de dato seleccionado.

Semantic Internal Object

Es una extensión que permite almacenar datos compuestos. Define la función #setinternal, la cual almacena lo que se denomina un "objeto interno". Se trata de tipos de datos que contienen relaciones que involucran más de un valor asociado. Esta extensión permite definir una tabla de dos dimensiones en una página, con la llamada a la función #setinternal para cada fila. Esto se realiza como muestra el Listado 3-2.

```
{{#set_internal: object -to-page property |property 1= value 1 |property 2 2=value 2 |...}}
```

Listado 3-2. Sintaxis de Semantic Internal Object.

Es útil en situaciones donde se debe asociar más de un valor a un elemento. Por ejemplo, se podría utilizar en un formulario donde se requiera ingresar, para una persona, sus hermanos y sus fechas de

nacimiento. Es preciso asociar cada fecha de nacimiento a cada hermano, y no como un dato suelto asociado al formulario. De esta manera, es necesario utilizar sub-objetos para representar a cada individuo, lo cual además permite realizar consultas sobre la información guardada de manera correcta.

Para nuestra aplicación es necesario definir un objeto de este tipo para permitir al usuario asociar varios episodios a un escenario. Cada episodio a su vez está compuesto por tres atributos: Sujeto-Verbo-Objeto y esto se representa como se muestra en el Listado 3-3.

```
{#set_internal:EpisodeObject
    |Who participates={{{Who participates|}}}
    |What does the participate do={{{What does the participate do|}}}
    |What the participate uses={{{What the participate uses|}}}
}
```

LISTADO 3-3. EJEMPLO SEMANTIC INTERNAL OBJECT.

ARRAYS

Esta extensión crea un conjunto de funciones capaces de operar con arreglos. Entre las funciones que se encontraron necesarias están:

-Arraydefine: construye un arreglo identificado con el valor seteado en el parámetro “key”, usando una lista de valores separados por un delimitador elegido por el desarrollador. El resultado es un arreglo de strings, si el delimitador no es especificado, por defecto toma la coma (,) (Listado 3-4).

```
{{#arraydefine:key|values|delimiter|options}}
```

LISTADO 3-4. EJEMPLO ARRAYDEFINE.

-Arrayprint: Imprime los valores de un arreglo en un formato personalizado (Listado 3-5).

```
{{#arrayprint:key|delimiter|pattern|subject|options}}
```

LISTADO 3-5. EJEMPLO ARRAYPRINT.

-Arraysize: Devuelve la longitud de un arreglo (Listado 3-6).

```
{{#arraysize:key}}
```

LISTADO 3-6. EJEMPLO ARRAYSIZE.

Estas son algunas de las operaciones que se pueden utilizar, además de operaciones para acceder a algún dato del arreglo (arrayindex, arraysize, arraysearch), modificar un arreglo (arrayunique, arrayreset, arraysort) y operaciones de interacción entre dos o más arreglos (arraymerge, arrayunion, arraydiff).

En el Listado 3-7 se muestra un ejemplo de aplicación de la operación arraydefine en nuestra herramienta.

```
{{#arraydefine: miArrayActorsOne
 | {{#ask: [[ActorsInScenarioObject::+]][[scenario::{{SUBJECTPAGENAME}}]]
           | mainlabel =- | headers = hide |?ActorInScenario}}
 | , | print=list, sort=asc, unique
}}
```

LISTADO 3-7. EJEMPLO ARRAYDEFINE.

PAGE FORMS

En versiones anteriores de esta extensión, se llamaba “Semantic Form”. Esta es la extensión que permite añadir, editar y consultar datos empleando formularios. Se basa en plantillas y define las llamadas “Páginas especiales” que son la base para la creación de nuestra aplicación. Por ejemplo, Special*:CreateProperty, Special:CreateTemplate, Special:CreateForm, Special:CreateCategory,* para crear los elementos que más adelante se detallarán con casos concretos*. Special:Templates, Special:Forms* para listar páginas, entre otras.

La definición de formulario es la primera de las dos secciones principales que consiste en cómo definir un formulario a través de una página dentro del espacio de nombres "Formulario:". Cubre toda la sintaxis de definición de formulario, incluyendo las etiquetas {{{{info}}}, {{{for template}}}, {{{end template}}}, {{{field}}} y {{{section}}. También cubre cómo agregar pestañas e información sobre herramientas.

Input types es la segunda sección principal. consiste en una lista de todos los tipos de entrada permitidos, así como los parámetros para cada uno, y el conjunto de tipos de datos de SMW (semantic media wiki) con los que se puede usar cada uno.

La extensión provee un mecanismo para asociar los formularios, a través de las funciones #forminput, #formlink, #formredlink y #queryformlink. Ésto permite a los usuarios la navegabilidad entre páginas además de cómo usar #formredlink para que las páginas enlazadas en rojo se creen automáticamente y cómo crear enlaces que modifiquen directamente una página, a través de la función #autoedit.

PARSER FUNCTIONS

Esta extensión añade funcionalidades relacionadas con lógica y operaciones con strings. A partir de la versión 1.15 ha incorporado estas últimas operaciones, ya que, en versiones anteriores, estaban incorporadas en otra extensión llamada StringFunctions.

La forma genérica de utilización es como se muestra en el Listado 3-8.

```
{{#functionname: argument 1 | argument 2 | argument 3 ... }}
```

LISTADO 3-8. SINTAXIS DE PARSER FUNCTIONS.

De las once funciones definidas en este template, hemos usado las que se detallan en el Listado 3-9.

```
{{#ifexpr: {{#var: i }} < {{#arraysize:allVerbs}} | true }}
{{#expr: {{#var: i }} + 1 }}
```

LISTADO 3-9. EJEMPLO PARSER FUNCTIONS.

VARIABLES

Esta extensión es utilizada para definir variables dentro de una página. Permite usarla dentro de la misma o en las plantillas incluidas en ella. Es posible cambiar su valor en cualquier momento, además se puede utilizar dada una determinada expresión matemática y en términos del valor anterior. Se pueden utilizar múltiples variables dentro de una página.

Con las etiquetas del Listado 3-10 es posible definir una variable y su valor. En el segundo caso, además de definir una variable y asignarle un valor, el mismo es impreso su valor.

```
{{#vardefine:nombredelavariable|valorespecificado}}
{{#vardefineecho:nombredelavariable|valorespecificado}}
```

LISTADO 3-10. SINTAXIS DE DEFINICIÓN DE VARIABLES.

Por último, se describe otra operación que pertenece a esta extensión. La misma se muestra en el Listado 3-11 y es utilizada para recuperar el valor almacenado en una variable. Una vez recuperado el valor de esta forma, se puede mostrar o realizar operaciones con el mismo.

```
{{#var:nombredelavariable}}
```

LISTADO 3-11. EJEMPLO DE DEFINICIÓN DE VARIABLES.

LOOPS

Esta extensión es utilizada para realizar operaciones de loops. Si bien cuenta con distintas operaciones tales como #while, #dowhile, #loop y #forargs, en este trabajo hemos utilizado solamente #while. Esta operación en conjunto con las últimas dos descriptas con anterioridad, permiten realizar la transformación de Scenarios a Test Cases en Task Method. Con estas operaciones es posible definir un solo template TestCaseData y renderizarlo tantas veces como episodios haya, representando así cada una de las tareas con la misma estructura. Además, nos permite mostrar el título de cada tarea con numeración dinámica.

La estrategia para dicha transformación con el uso de estas tres extensiones es la que se muestra en el Listado 3-12.

```
{{#vardefine: i | 0 }}
{{#while:
    | {{#ifexpr: {{#var: i }} < {{#arraysize:allVerbs}} | true }}
    | <nowiki />
    {{TestCaseData|param={{#var: i }}
    |item={{#vardefineecho: n | {{#expr: {{#var: i }} + 1 }} }}
    |iSubject={{#arrayindex: allSubjects | {{#var: i }} }}
    |iVerb={{#arrayindex: allVerbs | {{#var: i }} }}
    |iObject={{#arrayindex: allObjects | {{#var: i }} }}
}}
{{#vardefine: i | {{#expr: {{#var: i }} + 1 }} }}
}}
```

LISTADO 3-12. EJEMPLO DE USO DE LAS EXTENSIONES EN CONJUNTO.

Capítulo 4 - DISEÑO DE LA HERRAMIENTA

Esta sección describe cómo se implementa en la herramienta los distintos modelos, como así también las reglas de derivación. Algunos de los modelos implementados son centrales de la herramienta, mientras que otros son una extension realizada para mostrar las capacidades. Asi, los Escenarios son centrales, mientras que el Lexico Extendido del Lenguaje y los Casos de pruebas son extensiones. Las reglas definidas, también son una extensión, para poder obtener los casos de prueba. De esta forma, esta sección describe la implementación realizada y puede ser tomada como guía para futuras extensiones.

WIKITEXT

Los componentes que forman parte del sitio pueden crearse por medio de una interfaz gráfica. También puede editarse mediante una vista que muestra el contenido de la página en texto plano, con etiquetas especiales que proveen de formato al texto según la Ilustración 4-1.

ILUSTRACIÓN 4-1. EDICIÓN DE UN FORMULARIO.

Esta vista de edición de páginas está disponible para un grupo de usuarios con permisos específicos, los cuales pueden gestionarse desde la misma aplicación. A continuación, se muestra un conjunto de etiquetas que componen el lenguaje de mark-up WikiText.

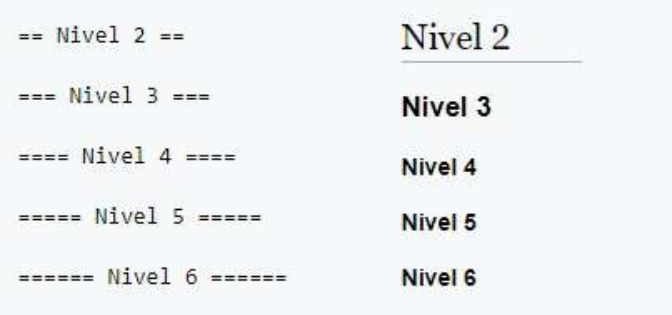


ILUSTRACIÓN 4-2. EJEMPLO DE TIPOS DE TÍTULOS EN WIKITEXT.

Una de las principales ventajas que se obtienen a partir de las wikis, es la navegación entre distintos enlaces. Éstos dotan de información semántica a las páginas que conforman el sitio. De estos enlaces se distinguen tres grandes grupos: links internos y links externos (Ilustración 4-3 e Ilustración 4-4). Un link en rojo indica que la página no existe aún. Puede ser creada haciendo click en el mismo.

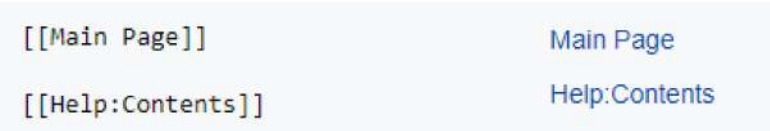


ILUSTRACIÓN 4-3. LINKS INTERNOS.

ILUSTRACIÓN 4-4. LINKS EXTERNOS.

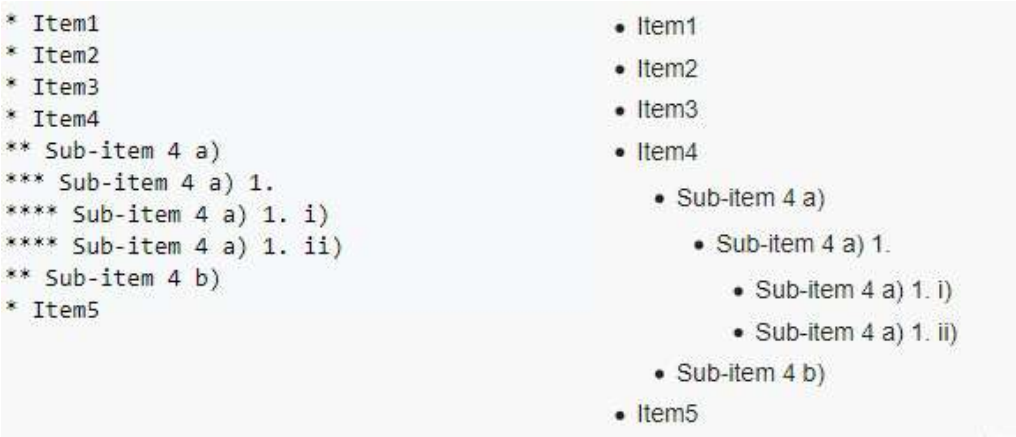


ILUSTRACIÓN 4-5. LISTAS DESORDENADAS.

Estos son ejemplo de la sintaxis básica de WikiText. Seguidamente se demostrará cómo se saca provecho de este metalenguaje que provee de significado semántico a la herramienta que se desarrolla. Antes de ello es necesario definir el modelo de clases que se utiliza.

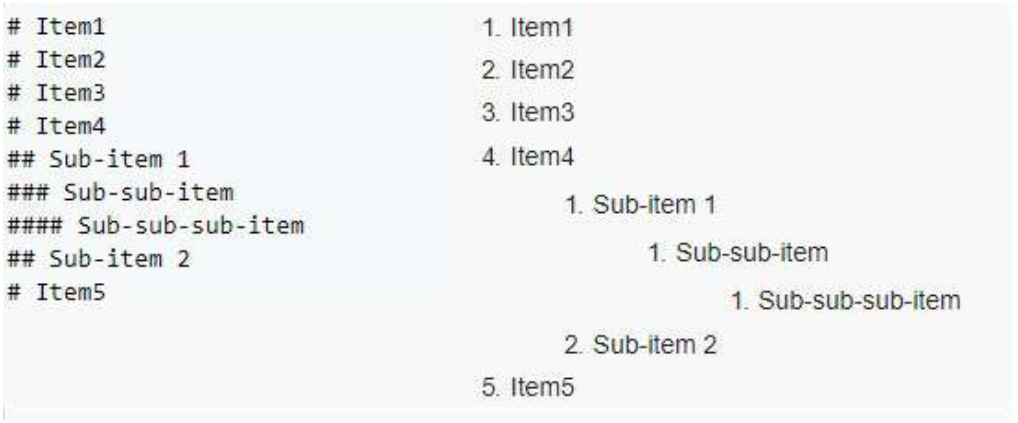


ILUSTRACIÓN 4-6. LISTAS ORDENADAS.

DIAGRAMA DE CLASES

En esta sección se describe el diagrama de clases empleado y los componentes de MediaWiki para representar sus relaciones y cómo aprovechar el concepto de propiedades semánticas que provee éste lenguaje. Si bien el desarrollo de la misma no está hecho con un lenguaje de programación orientado a objetos, es necesario contar con una representación que permita relacionar los componentes.

La clase LEL Symbol está compuesta por una noción y un impacto representado con las clases Notion e Impact respectivamente. Estas clases conocen el símbolo LEL al que pertenecen y no son necesariamente obligatorias para cada una de ellas, pero sí son únicos, es decir, no pueden pertenecer a otro símbolo. A su vez la clase LEL Symbol es una jerarquía con las clases hijas Subject, Object y Verb, necesarias para distinguir la relación que cada una de ellas tiene con los escenarios, ya que las propiedades son heredadas de la clase padre. Es decir, las clases hijas son necesarias ser distinguidas por sus comportamientos, pero no por su estructura.

La clase Scenario tiene dos atributos simples: Context y Goal. Estos atributos se definen como string. Esta clase también tiene los atributos Actor y Resources definidos como una relación de asociación con las subclases de LEL Symbol Subject y Object respectivamente. La asociación representa que un escenario puede tener uno o muchos actores y uno o muchos recursos. En el sentido inverso, un sujeto o un objeto pueden pertenecer a ningún o a muchos escenarios. En siguiente lugar, la clase Scenario está compuesto por muchos episodios. Inversamente el episodio pertenece a un escenario.

La clase Episode representa cada uno de los episodios que componen los escenarios. Los episodios conocen las subclases Subject, Object y Verb, dado que cada episodio se define en este trabajo como una terna sujeto-verbo-objeto. Esta relación es uno a uno. En el sentido inverso, cada sujeto, verbo y objeto pueden pertenecer a ninguno o a muchos episodios.

Por último, de define la clase Project que sirve para agrupar los escenarios. La relación se da a partir de que cada proyecto puede tener uno o muchos escenarios y cada escenario pertenece a un único proyecto.

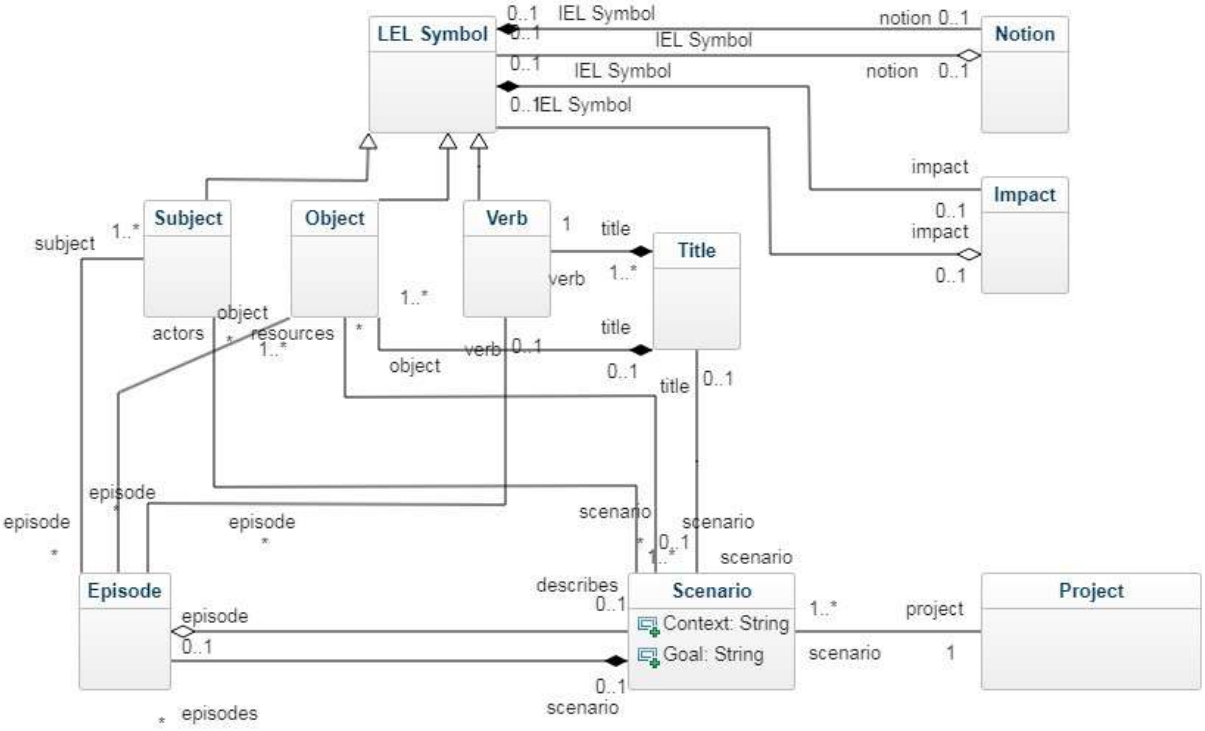


ILUSTRACIÓN 4-7. DIAGRAMA DE CLASES.

COMPONENTES DE LA HERRAMIENTA

Definidos los objetos y las relaciones entre ellos, se describen cómo se implementan en MediaWiki. Los componentes necesarios para la creación de la herramienta, son los provistos por MediaWiki y su extensión PageForms: Categories, Forms, Templates y Properties.

En la Tabla 4-1Tabla 4-3Tabla 4-3. Elementos que componen las páginas de escenarios. se muestran aquellas páginas necesarias para la carga y renderización de los proyectos que agrupan escenarios.

	Categories	Forms	Templates	Properties
Project	Category:Projects	Form:Project	Template:Project	Property:DescriptionProperty

TABLA 4-1. ELEMENTOS QUE COMPONEN LAS PÁGINAS PARA PROYECTO.

En la Tabla 4-2 se muestran las páginas creadas para mostrar los formularios y renderización de los componentes referidos a los símbolos LEL.

	Categories		Forms	Templates	Properties
Subject	Category:LELSymbol	Category:Subjects	Form:Subject	Template:Subject	Property:ImpactProperty Property:NotionProperty
Object		Category:Objects	Form:Object	Template:Object	Property:ImpactProperty Property:NotionProperty
Verb		Category:Verbs	Form:Verb	Template:Verb	Property:ImpactProperty Property:NotionProperty

TABLA 4-2. ELEMENTOS QUE COMPONEN LAS PÁGINAS PARA SÍMBOLOS LEL.

En la Tabla 4-3, se muestran los componentes necesarios para desarrollar las páginas correspondientes a los escenarios, es decir, para desarrollar el formulario de carga y la renderización del mismo junto a la derivación a Task Method Model.

Propiedades

A continuación, se numeran y describen todas las propiedades que son utilzadas en los tamplates para almacenar información y luego ser recuperada, tanto desde templates como desde forms.

-DescriptionProperty: propiedad de tipo Text que describe de forma sencilla un proyecto.

-Its context is: es una propiedad de tipo Text que define el contexto de cada escenario.

-The objective is: tipo Text expresa el objetivo de cada escenario.

-Its project is: tipo Text. Sirve para asociar cada escenario a un proyecto.

-EpisodeObject: es una relación entre episodios y escenarios. EpisodeObject es un objeto semántico que a su vez tiene tres atributos, en este caso son *who participates, what does the participate do* y *what the participate uses.* Cada uno de estos objetos representa un episodio del escenario compuesto por la terna sujeto-verbo-objeto. El objeto consta de un cuarto atributo llamado *Scenario*.

-Who participates: Atributo del objeto semántico EpisodeObject. Es un enlace a la página de un actor, que a su vez es un símbolo LEL del tipo sujeto.

-What does the participate do: Atributo del objeto semántico EpisodeObject. Es un enlace a la página de un verbo, que a su vez es un símbolo LEL del tipo verbo.

	Categories	Forms	Templates		Properties
Scenario	Category: Scenarios	Form: Scenario	Template:Scenarios		Property:Its context is Property:The objective is Property:Its project is
			Template:Episode	Template:EpisodeData	Property:EpisodeObject -Who participates -What does the participate do -What the participate uses -Scenario
			Template: ResourcesScenario		Property:Resources Property:ResourcesInScenarioObject -ResourceInScenario -Scenario
			Template: ActorsScenario		Property:Actors Property:ActorsInScenarioObject -ActorInScenario -Scenario
			Template:TestCase	Template:TestCaseData	

TABLA 4-3. ELEMENTOS QUE COMPONEN LAS PÁGINAS DE ESCENARIOS.

-What the participate uses: Atributo del objeto semántico EpisodeObject. Es un enlace a la página de un recurso, que a su vez es un símbolo LEL del tipo objeto.

-Scenario: Atributo del objeto semántico EpisodeObject creado para hacer un conocimiento bilateral entre cada uno de ellos y el Scenario al que corresponde.

-Resources: Es una propiedad del tipo Page. Son los recursos existentes en cada escenario. El enlace de su página muestra una breve descripción y la lista de escenarios con sus respectivos recursos.

-ResourcesInScenarioObject: es una propiedad del tipo InternalObject que representa una relación entre Escenarios y Recursos. Tiene los atributos *ResourcesInScenario* y *Scenario* utilizados para realizar operaciones sobre arreglos que permitan validar la consistencia entre los recursos usados en cada uno de los episodios dentro de un escenario y los recursos detallados para el escenario en general.

-Actors: Es una propiedad del tipo Page. Son los actores que participan en cada escenario. El enlace de su página muestra una breve descripción y la lista de escenarios con sus respectivos actores.

-ActorsInScenarioObject: es una propiedad del tipo InternalObject que representa una relación entre Escenarios y Actores. Tiene los atributos *ActorsInScenario* y *Scenario* utilizados para realizar operaciones sobre arreglos que permitan validar la consistencia entre los actores usados en cada uno de los episodios dentro de un escenario y los actores detallados para el escenario en general.

-NotionProperty: Propiedad tipo Text utilizada en los tres tipos de símbolos LEL: sujeto, verbo, objeto. Indica qué es el símbolo

-ImpactProperty: Propiedad tipo Text utilizada en los tres tipos de símbolos LEL: sujeto, verbo, objeto. Indica cómo repercute el símbolo en el sistema, es decir, los efectos de su uso dentro de la aplicación.

Templates

A continuación, se numeran y describen todos los templates utilizados para renderizar las páginas y a su vez, utilizados en los formularios de carga de los distintos componentes.

-Proyect: Es el cuerpo principal de la página de proyectos, usa la propiedad DescriptionProperty.

-Scenarios: Es el cuerpo principal de la página de escenarios, usa las propiedades *Its context is* y *The objective is*.

-Episode: Está definido unicamente con el campo EpisodesList, el cual sólo se ocupa de renderizar el template EpisodesData. Este template no muestra información, sino que genera las cajas del formulario para cargar los episodios.

-EpisodesData: Utiliza la instrucción set internal para generar cada objeto semántico EpisodeObject el cual representa cada episodio.

-ResourcesScenario: Este template utiliza la propiedad semántica The resources are, la cual especifica un conjunto de recursos asociados al escenario. A su vez, mediante la instrucción #formredlink se especifica que, en caso de no existir una página propia para el recurso mencionado, se debe redirigir al formulario de carga para un símbolo LEL de la subcategoría Object.

-ActorsScenario: Similar a ResourcesScenario, con la diferencia que el formulario de carga correspondiente es Subject.

-TestCase: En este template se realiza parte de la derivación de Scenarios a Test Cases en Task Method. Para ello define un arreglo de verbos, un arreglo de sujetos y otro arreglo de recursos, los mismos pertenecientes al escenario de la página actual. Con estos elementos define la estructura de la cabecera del Task Method y la iteración encargada de renderizar dinámicamente el template TestCaseData tantas veces como episodios haya.

-TestCaseData: Este template visualiza cada una de las tareas del Task Method correspondientes a cada uno de los episodios del escenario de la página actual.

-Subject: Implementa los datos referentes a un símbolo LEL, particularmente Subject, con los campos característicos de los símbolos, es decir Noción e Impacto, con las propiedades semánticas NotionProperty e ImpactProperty. Por otro lado, el template permite definir la categoría a la que pertenecerán las páginas creadas a partir del mismo, en este caso, la categoría Subject, que a su vez es una subcategoría de LELSymbol.

-Object: Similar a Subject.

-Verb: Similar a Subject.

Formularios

A continuación, se numeran y describen los formularios de carga de información para generar las distintas páginas. En ellos se utilizan los templates y las propiedades descriptas anteriormente.

-Proyect: Es el formulario para la carga de un proyecto. Tiene definido un solo campo que es la descripción del proyecto.

-Scenario: El formulario de carga del escenario define los datos que deben ser cargados para dar de alta una instancia del mismo. Para ello se definen dos campos simples: Objective y Context. Seguidamente se utilizan los templates definidos anteriormente con la instrucción for template.

-Subject: Se usa tanto para cargar un Sujeto, como para modificar uno existente. Posee los campos de carga Notion e Impact.

-Object: Similar a Subject.

-Verb: Similar a Subject.

Categorías

El último elemento que conforman los componentes necesarios para crear las páginas de esta herramienta, son las categorías. Estas categorías permiten agrupas páginas que fueron creadas con el mismo tipo de formulario.

-Projects: Categoría que lista todos los proyectos existentes ordenados alfabéticamente. Al crear un proyecto utilizando el formulario correspondiente, por defecto lo asocia a la categoría Projects.

-Scenarios: Categoría que lista todos los escenarios existentes ordenados alfabéticamente. Al crear un escenario utilizando el formulario ya descrito, por defecto se le asocia la categoría Scenarios.

-LELSymbol: Categoría padre de las categorías Subject, Object y Verb. Al listar las páginas pertenecientes a esta categoría, se muestra un listado de las subcategorías, de manera de poder elegir una de ellas y listar las páginas que pertenecen a la subcategoría seleccionada.

-Subjects: Lista todas las páginas pertenecientes a sujetos, los cuales son a su vez LELSymbol. Al crear la página por medio del formulario Subject, por defecto se crea la misma como parte de esta categoría.

-Objects: Similar Subject.

-Verbs: Similar Subject.

REGLAS DE DERIVACIÓN DE ESCENARIOS A CASOS DE PRUEBA

El núcleo de la herramienta se basa en las reglas de derivación. Las mismas proponen que a partir del modelo de Escenarios, propio de las etapas tempranas del ciclo de vida del software, se obtenga otro modelo: Task/Method Model (pruebas de aceptación), utilizando cinco reglas de derivación específicas.

Entre varias ventajas que brinda esta integración de modelos, la principal de ellas radica en que tener definidas las pruebas de aceptación en la etapa de modelado del sistema. Es decir, en la etapa de definición de escenarios (elicitación de requerimientos), facilita la identificación temprana y corrección de interpretaciones equivocadas entre las partes, es decir entre clientes y desarrolladores/analistas. Por otro lado, se obtiene una aproximación al código ejecutable para generar todos los casos de prueba, lo que aliviana el esfuerzo de las futuras etapas del proceso de desarrollo. Como se vio en la descripción del ciclo de vida de desarrollo del modelo V, el producto obtenido en cada paso (requisitos, diseño y codificación) debe ser probado y cuanto antes se identifiquen los malos entendidos, antes podrán solucionarse (Borst, Arana, Crave, Galeano, 2005).

La derivación se obtiene a partir de cinco reglas bien definidas. A continuación, se describe cada una de las mismas. Luego se ejemplifica la derivación tomando como punto de partida un escenario en particular y aplicando paso a paso las reglas correspondientes.

Regla 1: Identificación de tareas

Es natural considerar las acciones como tareas. Estas acciones aparecen en el nombre del Escenario y en la descripción de cada uno de los episodios. En el mapeo con LEL las acciones corresponden a la categoría verbo. Por lo tanto, la regla establece que cada símbolo LEL de la categoría verbo incluido en los episodios se traduce en una tarea del modelo Task/Method.

Regla 2: Identificación de los parámetros para las tareas

Los parámetros del modelo Task/Method corresponden a los objetos que desean manipularse en las tareas. En los Escenarios estos objetos están representados por los Recursos y Actores, correspondientes a los símbolos LEL de las categorías “objeto” y “sujeto” respectivamente. Por lo tanto, esta regla establece que los símbolos LEL objeto y sujeto correspondiente a cada episodio del escenario se transforman en parámetros de las tareas definidas para el correspondiente verbo del episodio.

Regla 3: Métodos de los episodios

En los escenarios, cada paso que logre cumplir el objetivo del mismo, se representa con los episodios, es decir que el éxito del escenario radica en el éxito parcial de cada uno y de los episodios. Este resultado, en el modelo Task/Method puede ser exitoso o fallido. Entonces esta regla establece que la ejecución de tareas, se describe mediante métodos, cada episodio del escenario se mapea a un método de Task/Method.

Regla 4: Secuencia de tareas

La secuencia de episodios en un escenario determina la secuencia de tareas en la sección de control de un método en el modelo Task/Method. Como cada episodio *ei* de un escenario *s* se traduce en una tarea *t* y la ejecución de escenarios se traduce en un método, se tiene que la secuencia de episodios *{ei}* determina la secuencia *{ti}* de tareas en la sección de control del modelo Task/Method.

Regla 5: Método de caso de prueba

En este modelo se asume que a cada caso de prueba le corresponde un estado binario de éxito de los episodios: verdadero o falso. Si se logra el éxito de un episodio, la ejecución del escenario continúa con el siguiente episodio. En una situación de fallo, el escenario se detendrá. Este caso de interrupción del flujo de ejecución se representará mediante un método para la siguiente tarea (episodio) en el que el campo de precondición corresponde a la falla del caso de prueba.

Cabe considerar que, en el desarrollo de la herramienta, no se contempla la ejecución de flujos alternativos para los casos de éxito de cada episodio, simplemente se muestra un mensaje de error que permita identificar en qué paso se detuvo la ejecución.

Ejemplo de aplicación de las reglas

Para llevar a cabo la derivación aplicando cada una de las reglas, se toma como referencia el escenario “Decidir tipo de labor cultural”, descrito en la Tabla 2-1. Para cada regla se numeran los criterios tenidos en cuentas y luego se muestra el resultado de la regla aplicada.

Regla 1: Identificación de tareas

- A partir del modelo de Escenarios, cada episodio es una tarea en el modelo de Task/Method. El título del Escenario es también una tarea.
- A partir del modelo LEL, cada símbolo verbo es traducido en una tarea.

Ejemplo:

1. Episodio: Agricultor elige un sistema de conducción
   Tarea: Elegir
2. Episodio: Agricultor deciden técnica de capacitación.
   Tarea: Decidir
3. Episodio: Líder establece las políticas de poda a aplicar

Tarea: Establecer

4. Episodio: Líder escribe un procedimiento de acuerdo al estándar que describe el sistema de conducción, la técnica de capacitación y las políticas de poda.
   Tarea: Escribir

Regla 2: Identificación de los parámetros para las tareas

- A partir del modelo de Escenarios, cada recurso asociado a una acción se traduce en un parámetro del modelo Task/Method.
- A partir del modelo LEL, cada símbolo de las categorías sujeto y objeto asociados a una acción se traducen en parámetros de una tarea asociada a un verbo.

Ejemplo:

1. Episodio: Agricultor eligen un sistema de conducción
   Tarea: Elegir(Agricultor, sistema_conduccion)
2. Episodio: Agricultor deciden técnica de capacitación.
   Tarea: Decidir(Agricultor, tecnica_capacitacion)
3. Episodio: Líder establece las políticas de poda a aplicar
   Tarea: Establecer(Lider, politica_de_poda)
4. Episodio: Líder escribe un procedimiento de acuerdo al estándar que describe el sistema de conducción, la técnica de capacitación y las políticas de poda.
   Tarea: Escribir(Lider, procedimiento_completo_de_decision)

Regla 3: Métodos de los episodios

- Los objetivos de cada uno de los episodios están asociados tareas del modelo Task/Method, que se traducen en métodos.

Ejemplo:

1. Objetivo del Escenario: Decidir tipo de labor cultural.
   Método M1.
2. Objetivo del Episodio: Agricultor eligen un sistema de conducción.
   Método M11.
3. Objetivo del Episodio: Agricultor deciden técnica de capacitación.
   Método M12.
4. Objetivo del Episodio: Líder establece las políticas de poda a aplicar.
   Método M13.
5. Objetivo del Episodio: Líder escribe un procedimiento de acuerdo al estándar que describe el sistema de conducción, la técnica de capacitación y las políticas de poda.
   Método M14.

Regla 4: Secuencia de tareas

- Cada episodio se traduce en una secuencia de tareas para el método que corresponde a esa tarea.

Ejemplo:

Episodios:

1. Agricultor eligen un sistema de conducción.
2. Agricultor deciden técnica de capacitación.
3. Líder establece las políticas de poda a aplicar.
4. Líder escribe un procedimiento de acuerdo al estándar que describe el sistema de conducción, la técnica de capacitación y las políticas de poda.

Método M1:

Control:

1. Tarea: Elegir(Agricultor, sistema_conduccion)
2. Tarea: Decidir(Agricultor, tecnica_capacitacion)
3. Tarea: Establecer(Lider, politica_de_poda)
4. Tarea: Escribir(Lider, procedimiento_completo_de_decision)

Regla 5: Método de caso de prueba

- Para cada episodio ei (tarea ti), se agrega un método mei + 1 a la tarea ti + 1 en el que el campo de condición previa corresponde a la falla del caso de prueba.

Ejemplo:

Caso de prueba:

Producto lavado correctamente/ lavado incorrectamente

Método: M12

Tarea: lavar(Agricultor, producto) # proxima tarea

Precondición: not Producto_lavado_correctamente

Control: message(“El producto no quedó correctamente lavado, stop”);

Stop;

Para llevar a cabo esta transformación es necesario que el modelo del dominio de aplicación sea representado en un modelo de objetos (descrito en lenguaje UML). Las tareas y métodos corresponden a métodos del modelo. Los objetos del dominio corresponden a la lista de parámetros. Las tareas no pueden alcanzar objetos que no hayan sido especificados en dicha lista.

DESARROLLO DE LA HERRAMIENTA

En esta sección se describe cómo se ha implementado la herramienta. Se detalla cómo se ha desarrollado cada uno de los elementos propuestos para especificar y derivar los modelos elegidos. Para cada elemento, se muestra cómo se crean los componentes propios de MediaWiki y sus extensiones, utilizando lenguaje de WikiText. En esta sección, además de mostrar cómo se implementa la renderización de la información, se detalla cómo se usan las propiedades semánticas para mantener las relaciones entre los elementos. De esta forma luego pueden ser accedidos y luego renderizados en el modelo derivado.

Project

La clase Project sirve para representar un proyecto los cuales agrupan los escenarios correspondientes al mismo. La información que nos interesa de cada proyecto es simplemente una descripción.

Para implementar la entidad Project es necesario crear los elementos descritos en la Tabla 4-1. Category:Projects permite agrupar todo el contenido de la wiki identificado como un Proyecto. El mismo sentido de categoría se aplica a todas las entidades utilizadas en este trabajo. Un formulario correspondiente a dicha categoría, Form:Project, este formulario presenta al usuario de manera amigable el campo requerido para la creación (descripción). El formulario a su vez está compuesto por el Template:Project con la propiedad Property:DescriptionProperty. Los templates son los encargados de estructurar la forma en que la página será renderizada. Las propiedades en este caso son propias de la extensión PageForm y corresponden a las propiedades semánticas.

Las propiedades, templates, forms y categories se crean por medio de las SpecialPages de MediaWiki. De esta forma se crean con la estructura por defecto. En nuestro caso accedemos a la edición del contenido mediante WikiText para personalizar la estructura que queremos que muestre una página de Project.

Los templates están divididos en dos secciones. La primera es la que se encierra entre los tags <noinclude>. El código encerrado en este tag no es renderizado, sino que muestra el texto tal cual se mostrará en la pantalla, útil para indicar al usuario cómo se debe invocar el template. Éste código se genera automáticamente, aunque puede ser editado. El código que realmente nos interesa es el que se encuentra encerrado entre los tags <includeonly> y a éste le daremos mayor relevancia en la descripción del armado de la herramienta.

El template Project (Listado 4-1) implementa el cuerpo de la página de un proyecto. Tiene el campo para la descripción en la línea 5. A partir de la línea 8 se realiza una consulta semántica para que, en caso de haber algún escenario asociado al proyecto de la página actual, los liste. En el caso contrario se muestra un texto advirtiendo que el proyecto aún no tiene escenarios asociados.

```
<includeonly>

{| class="wikitable"
! Description
| [[DescriptionProperty::{{{ProjectDescription|}}}]]
|}

{{#if:
  {{#ask:
    [[Its project is::{{SUBJECTPAGENAME}}]]
  }} |
  ==Episodes of this project==

  {{#ask:
    [[Its project is::+]]
    |mainlebel=-
    |format=ol
    | headers = hide
  }} |
  There are no scenarios for this project yet.
}}

[[Category:Projects]]
</includeonly>
```

LISTADO 4-1. TEMPLATE PROJECT.

Los templates además de renderizar el contenido de las páginas, sirven para implementar la carga de formularios, como se muestra en la Ilustración 4-2. La utilización del template se ve en el código encerrado entre los tags {{{for template|Project}}} y {{{end template}}}. Allí se hace referencia al campo Description, el cual setea el valor a la propiedad semántica ProjectDescription, ya explicada anteriormente. Este es el único campo utilizado en este formulario. Los tags de las últimas líneas son los botones de Guardar, Ver los cambios y Cancelar. Éstos son elementos estándar de MediaWiki que pueden o no ser utilizados en los formularios.

Scenario

La clase escenario es la más compleja de todo el diseño, ya que en ella convergen todas las demás clases. De cada escenario interesa registrar el proyecto al que pertenece, el objetivo, el contexto, los recursos que se utilizan, los actores que participan y cada uno de los episodios que lo componen. Además, se agrega la posibilidad de cargar un texto libre que permite al usuario escribir cualquier aclaración que no quede contemplada en los demás campos.

La carga de episodios es más compleja que el resto de los campos. Esto es debido a que el episodio en sí es un objeto complejo compuesto por la terna sujeto-verbo-objeto. Para representar esta complejidad se usan los objetos semánticos de MediaWiki, que se cargan en el formulario de alta de escenarios.

```
<includeonly>
<div id="wikiPreview"
        style="display: none; padding-bottom: 25px;
        margin-bottom: 25px;
        border-bottom: 1px solid #AAAAAA;"></div>

{{{for template|Project}}}
{| class="formtable"
! Description:
| {{{field|ProjectDescription}}}
|}
{{{end template}}}

{{{standard input|save}}}
{{{standard input|changes}}}
{{{standard input|cancel}}}
</includeonly>
```

LISTADO 4-2. FORM PROJECT.

La implementación de la entidad Scenario está dada por los templates Scenario, el cual es el cuerpo de la página; el template Episode y su subtemplate EpisodeData que implementan la carga de cada episodio en el formulario, el listado en la página y el objeto semántico; ResourcesScenario para el listado de recursos y ActorsScenario para el listado de actores. Además de éstos, existe el template TestCase y su subtemplate TestCaseData. La combinación de ambos resuelve la derivación de escenarios a tareas de Task Method, modelo elegido para realizar los casos de prueba.

El template Scenarios (Listado 4-3) es el encargado de proporcionar la estructura a la página de cada escenario. En él se definen los campos Its project is, propiedad que asocia el escenario a un proyecto; Context y Objective, campos de texto a modo descriptivo.

```
<includeonly>
= Principal elements =
{| class="wikitable"
! Project
| [[Its project is::{{{Its project is|}}}]]
|-
! Context
| [[Its context is::{{{Context|}}}]]
|-
! Objective
| [[The objective is::{{{Objective|}}}]]
|}
[[Category:Scenarios]]
</includeonly>
```

LISTADO 4-3. TEMPLATE SCENARIOS.

El template Episode (Listado 4-4) solamente define un campo que es el encargado de renderizar el template EpisodeData. Éste es quien se ocupa de la carga de información referida a cada uno de los episodios en el formulario correspondiente. Además, se encarga de incluir el template TestCase el cual renderiza la transformación de episodios a casos de prueba del TaskMethod.

```
= Episodes =
<ol>
  <includeonly>{{{EpisodesList|}}}</includeonly>
</ol>

{{TestCase}}

== Contextual information ==
```

LISTADO 4-4. TEMPLATE EPISODE.

El template EpisodeData (Listado 4-5), por un lado, define Who participates, What does the participates do y What the participates uses como propiedades correspondientes a los formularios Subject, Verb y Object respectivamente. Cada uno de ellos se muestran como links en rojo en caso de no existir aún dicha página. Estos links redireccionan a los formularios de creación según corresponda. El hecho de mostrarse como un link en rojo lo permite la sentencia #formredlink. Si la página ha sido creada, el link se muestra en celeste y redirecciona a la página correspondiente.

En la segunda sección del template se define el objeto semántico EpisodeObject con las cuatro propiedades. Las tres primeras referidas a las descritas en el párrafo anterior. La cuarta propiedad, Scenario, es creada para tener una correspondencia bilateral entre en el escenario y cada episodio. Esto quiere decir que cuando creamos un objeto semántico EpisodeObject estaremos asignando el nombre del escenario al que corresponde el episodio en proceso de creación. SUBJECTPAGENAME es una de las llamadas "Palabras Mágicas" en la sintaxis de MediaWiki. Su utilidad es devolver el resultado de una función asociado a cada palabra mágica. En este caso particular obtiene el valor de la ruta de la página actual. Así mismo se explica con más detalle el uso de esta propiedad cuando se describe la implementación del formulario Scenario.

```
<includeonly>
<li>
{|
| {{#formredlink:target={{{Who participates|}}}|form=Subject}}

| {{#formredlink:target={{{What does the participate do|}}}|form=Verb}}

| {{#formredlink:target={{{What the participate uses|}}}|form=Object}}

|}

{{#set_internal:EpisodeObject
|Who participates={{{Who participates|}}}
|What does the participate do={{{What does the participate do|}}}
|What the participate uses={{{What the participate uses|}}}
|Scenario={{SUBJECTPAGENAME}}
}}
</li>

[[Category:Scenarios]]
</includeonly>
```

LISTADO 4-5. TEMPLATE EPISODEDATA.

El template ResourcesScenario (Listado 4-6) representa los recursos (a su vez símbolos LEL del tipo objeto) que son utilizados para un escenario. Se define con un arreglo mediante la instrucción #arraymap. Esto permite que el campo Resources admita múltiples elementos. Cada elemento será un Object. Si el Object ingresado en el campo no ha sido creado aún se renderiza en la pantalla con un link en rojo, cuya aplicabilidad ya se explicó anteriormente. A su vez por cada el elemento en el arreglo se crea una instancia del objeto semántico ResourcesInScenarioObject. Este objeto semántico está compuesto por el Object y el Scenario al que corresponde dicho objeto (asociado a un recurso del escenario). La razón por la cual se crea este objeto semántico, se da a partir de la necesidad de crear validaciones de coherencia de datos en la carga de la información de un escenario.

```
<includeonly>
{| class="wikitable"
!Resources
|{{#arraymap:{{{Resources|}}}
            |,
            |xx
            |[[Resources::xx| ]]{{#formredlink:target=xx|form=Object}}

   {{#set_internal: ResourcesInScenarioObject
       | ResourceInScenario=xx
       | Scenario={{SUBJECTPAGENAME}} }}
  }}
|}
</includeonly>
```

LISTADO 4-6. TEMPLATE RESOURCESSCENARIO.

El template ActorsScenario (Listado 4-7) tiene la misma funcionalidad que el template ResourcesScenario, pero refiriéndose a los actores que pertenecen al escenario. En este caso el objeto semántico que se crea es ActorsInScenarioObject.

```
<includeonly>
{| class="wikitable"
! Actor(s)
|{{#arraymap:{{{Actors|}}}
        |,
        |xx
        |[[Actors::xx| ]]{{#formredlink:target=xx|form=Subject}}

    {{#set_internal: ActorsInScenarioObject
        | ActorInScenario=xx
        | Scenario={{SUBJECTPAGENAME}} }}
  }}
|}
</includeonly>
```

LISTADO 4-7. TEMPLATE ACTORSSCENARIO.

Los templates, además de definir la forma en que se renderiza cada sección de la página, sirven para implementar la carga de datos desde un formulario. El formulario en cuestión se llama Scenario.

De la misma forma que en el resto de las páginas, la etiqueta <includeonly> le da comienzo al código que nos interesa describir. Los templates se incluyen en el formulario con la etiqueta <for template>.

En el caso del campo Project (Listado 4-8) se utiliza la instrucción #info, cuyo texto se muestra en el campo de la página renderizada como un tooltip informativo que sirve de guía para el usuario. Al definir el tipo del campo como combobox permite que la herramienta ofrezca al usuario una lista de opciones posibles para la carga del mismo.

```
<includeonly>
{{{for template|Scenarios}}}
{| class="formtable"
! Project: {{#info:Complete here the project}}
| {{{field|Its project is | input type=combobox}}}
|-
! Context:
| {{{field|Context}}}
|-
! Objective:
| {{{field|Objective}}}
|}
{{{end template}}}
```

LISTADO 4-8. FORM SCENARIO. DATOS GENERALES.

En la renderización del campo Actors (Listado 4-9), además de mostrar un tooltip con la instrucción #info, permite tomar todos los valores de las páginas creadas bajo la categoría Subjects.

```
{{{for template|ActorsScenario}}}
{| class="formtable"
! Actors: {{#info:Complete here the actors who participate in the episodes}}
| {{{field|Actors|values from category=Subjects}}}
|}
{{{end template}}}
```

LISTADO 4-9. FORM SCENARIO. ARREGLO DE ACTORES.

En el formulario de carga de escenarios no solamente se renderizan los templates. Para el desarrollo de la herramienta propuesta se ve la necesidad de mantener coherencia en la carga de información. Para ello se agregan validaciones para que el usuario sea advertido de que en el campo Actors podría haber inconsistencias (Listado 4-10). La forma de realizar esta validación se hace usando la instrucción #arraydefine que permite crear arreglos. El primero que creamos es miArrayActorsOne. En este arreglo se obtiene mediante la consulta semántica #ask todos los actores cargados en el objeto semántico ActorsInScenarioObject cuyo escenario sea el de la página actual. En el arreglo miArrayActorsTwo se obtienen todos los actores cargados en el objeto semántico EpisodeObject cuyo escenario sea el de la página actual. Seguidamente se realiza la operación de diferencia de conjuntos entre ambos arreglos para obtener aquellos actores que han sido cargados como actores en el escenario, pero no se han utilizado para definir episodios.

```
<div style="display:none;">
{{#arraydefine: miArrayActorsOne
 | {{#ask: [[ActorsInScenarioObject::+]][[scenario::{{SUBJECTPAGENAME}}]]
           | mainlabel =- | headers = hide |?ActorInScenario}}
 | , | print=list, sort=asc, unique
}}

{{#arraydefine: miArrayActorsTwo
 | {{#ask: [[EpisodeObject::+]][[scenario::{{SUBJECTPAGENAME}}]]
           | mainlabel =- | headers = hide |?Who participates}}
 | , | print=list, sort=asc, unique
}}

{{#arraydiff: difActorsOne | miArrayActorsOne | miArrayActorsTwo}}
</div>

{{#ifexpr: {{#arraysize:difActorsOne}} = 0
        | <div style="diplay:block;"></div>
        | <span> The actors {{#arrayprint: difActorsOne }} have not been
        defined in any episode yet. You must use them in one of the
        episodes, in the field 'Actor (who?):'.</span>}}
```

LISTADO 4-10. FORM SCENARIO. VALIDACIONES DE CONSISTENCIA DE INFORMACIÓN DE ACTORES.

La misma lógica se mantiene para definir la carga y validación del campo Resources.

```
{{{for template|ResourcesScenario}}}
{| class="formtable"
! Resources: {{#info:Complete here the resources which are uses in the episodes}}
| {{{field|Resources|values from category=Objects}}}
|}
{{{end template}}}

<div style="display:none;">
{{#arraydefine: miArrayResourcesOne
 | {{#ask: [[ResourcesInScenarioObject::+]][[scenario::{{SUBJECTPAGENAME}}]]
           | mainlabel =- | headers = hide |?ResourceInScenario}}
 | , | print=list, sort=asc, unique
}}

{{#arraydefine: miArrayResourcesTwo
 | {{#ask: [[EpisodeObject::+]][[scenario::{{SUBJECTPAGENAME}}]]
           | mainlabel =- | headers = hide |?What the participate uses}}
 | , | print=list, sort=asc, unique
}}

{{#arraydiff: difResourcesOne | miArrayResourcesOne | miArrayResourcesTwo}}
</div>

{{#ifexpr: {{#arraysize:difResourcesOne}} = 0
        | <div style="diplay:block;"></div>
        | <span> The resources {{#arrayprint: difResourcesOne }} have not been
        defined in any episode yet. You must use them in one of the episodes, in the
        field 'Resources (what he uses?):'. </span>}}
```

LISTADO 4-11. FORM SCENARIO. VALIDACIONES DE CONSISTENCIA DE INFORMACIÓN DE RECURSOS.

Para la carga de los episodios se utilizan los templates Episode y EpisodeData (Listado 4-12). El atributo múltiple se utiliza para indicar que múltiples instancias del template EpisodeData. El atributo embed in especifica que las llamadas a este template dentro de las páginas generadas, se incrustan dentro del valor de un campo de otra plantilla. Debe utilizarse en conjunto con el atributo holds.

```
{{{for template|Episode|label=Episode}}}
  {{{field|EpisodesList|holds template}}}
{{{end template}}}

{{{for template
        |EpisodeData
        |multiple
        |embed in field=Episode[EpisodesList]}}}
{|
|'''Actor (who?):'''
        {{{field|Who participates |input type=combobox}}}
|-
|'''Verb (what he does?):'''
        {{{field|What does the participate do |input type=combobox}}}
|-
|'''Resource (what he uses?):'''
        {{{field|What the participate uses |input type=combobox}}}
|}
{{{end template}}}
```

LISTADO 4-12. FORM SCENARIO. EPISODIOS.

```
<div style="display:none;">
{{#arraydiff: difActorsTwo | miArrayActorsTwo | miArrayActorsOne}}
</div>

{{#ifexpr: {{#arraysize:difActorsTwo}} = 0
        | <div style="diplay:block;"></div>
        | <span> In the episodes there are the actors:
        {{#arrayprint: difActorsTwo }}
        which have not been defined in main actors of the scenario.
        You must complete it there to maintain the consistency of
        information. </span>}}

<div style="display:none;">
{{#arraydiff: difResourcesTwo | miArrayResourcesTwo | miArrayResourcesOne}}
</div>

{{#ifexpr: {{#arraysize:difResourcesTwo}} = 0
        | <div style="diplay:block;"></div>
        | <span> In the episodes there are the resources:
        {{#arrayprint: difResourcesTwo}}
        which have not been defined in main resources of the scenario.
        You must complete it there to maintain the consistency of
        information.</span>}}

'''Contextual information'''

{{{standard input|free text|editor|rows=10}}}
{{{standard input|save}}}
{{{standard input|changes}}}
{{{standard input|cancel}}}
</includeonly>
```

LISTADO 4-13. FORM SCENARIO. VALIDACIONES PARA LOS EPISODIOS.

Luego se realiza la validación para advertir al usuario sobre aquellos actores y recursos que han sido utilizados en los episodios pero que no han sido definidos como actores y recursos del escenario en general (Listado 4-13). Finalmente se define un campo de texto libre para que el usuario realice alguna descripción que no sea contemplada en el resto de los campos especificados. Se agregan botones para guardar, ver cambios, y cancelar la operación de carga.

LEL Symbols

Para crear las páginas de los símbolos LEL, además de crear las propiedades semánticas, los formularios y las plantillas propias que han sido descritas a lo largo de este trabajo, es necesario editar las categorías. En las páginas que representan los componentes de esta herramienta, hasta el momento, se ha mantenido una categoría por plantilla para los templates Project y Scenario. Esto es así, porque estos dos templates son los encargados de la renderización de una página, el resto de los templates definidos son complementarios para la manipulación de la información. La edición planteada para la definición de categorías de los símbolos LEL pretende establecer una jerarquía. Es decir, tener una categoría padre llamada LEL Symbol y tres categorías hijas llamadas Subjects, Verbs y Objects. Esto permite organizar la navegación de la herramienta, teniendo agrupadas todas las páginas correspondientes a una de las subcategorías y por otro lado listar las categorías que corresponden a símbolos LEL.

```
<includeonly>
{| class="wikitable"
! Notion
| [[NotionProperty::{{{Notion|}}}]]
|-
! Impact
| [[ImpactProperty::{{{Impact|}}}]]
|}

{{#if:
  {{#ask:
    [[EpisodeObject::+]]
    [[Who participates::{{SUBJECTPAGENAME}}]]
  }} |
  ==Included in the Scenarios (Episodes)==

  {{#ask:
    [[EpisodeObject::+]]
    [[Who participates::{{SUBJECTPAGENAME}}]]
    |mainlebel=-
    |format=ol
    | headers = hide
    |? Who participates
    |? What does the participate do
    |? What the participate uses
  }} |
  No scenarios have been established with this subject yet.
}}

[[Category:Subjects]]
</includeonly>
```

LISTADO 4-14. TEMPLATE SUBJECT.

Para el caso de los símbolos LEL del tipo Subject, el template correspondiente es que el se muestra en el Listado 4-14. Dentro de la etiqueta definida en las primeras siete líneas, se establecen los campos y las propiedades que serán inputs de carga en el formulario. Dentro de la sentencia if se consulta si existen objetos semánticos EpisodeObject cuya propiedad Who participates sea el sujeto correspondiente a la página actual. Es decir que se muestran todos los episodios en los que el sujeto es un actor de la página actual. En caso que no existan objetos con coincida esta propiedad, se muestra un cartel advirtiendo de tal situación. En última línea se determina que las páginas creadas con este template, es decir los Subject, pertenecen a la categoría Subjects.

El template es utilizado en el formulario de carga de un símbolo LEL del tipo Subject (Listado 4-15). Solamente define los campos Notion e Impact y los botones para guardar, mostrar cambios y cancelar.

```
<includeonly>
<div id="wikiPreview"
        style="display: none; padding-bottom: 25px;
        margin-bottom: 25px; border-bottom: 1px
        solid #AAAAAA;">
</div>

{{{for template|Subject}}}
{| class="formtable"
! Notion:
| {{{field|Notion}}}
|-
! Impact:
| {{{field|Impact}}}
|}
{{{end template}}}

{{{standard input|save}}}
{{{standard input|changes}}}
{{{standard input|cancel}}}
</includeonly>
```

LISTADO 4-15. FORM SUBJECT.

Para los templates Object y Verb se definen los templates de la misma forma que Subject, pero utilizando las propiedades “What the participate uses” y “What does the participate use” respectivamente. Cada uno de estos templates con sus forms correspondientes.

Test Cases

Los casos de prueba se generan a partir de la derivación de episodios y basándose en las reglas especificadas en este trabajo. La renderización de los casos de prueba se hace usando los datos cargados para los episodios. Debido a ello, a diferencia del resto de los componentes que conforman esta herramienta, se definen dos templates, pero ningún formulario.

El primer template es TestCase (Listado 4-16), que representa un template padre donde se define el encabezado del método principal que servirá para validar la completitud exitosa del escenario. Este encabezado traduce el nombre del escenario en el nombre de un método. El criterio elegido para esta

traducción es convertir los espacios en blanco a guion bajo. A su vez define como parámetros la lista de actores y de recursos utilizados en todos los episodios.

```
=Method: M1=
<b>Task: </b>
{{#replace:{{SUBJECTPAGENAME}}| |_}}
({{#arraydefine: actorsInThisScenario
        | {{#ask:[[EpisodeObject::+]][[Scenario::{{SUBJECTPAGENAME}}]]
                | mainlabel =-
                | headers = hide
                |? Who participates
                | limit = 10000
        }}
 | , | print=list, sort=asc, unique
}},
{{#arraydefine: resourcesInThisScenario
        | {{#ask:[[EpisodeObject::+]][[Scenario::{{SUBJECTPAGENAME}}]]
                | mainlabel =-
                | headers = hide
                |? What the participate uses
                | limit = 10000
        }}
 | , | print=list, sort=asc, unique
}})<br>
```

LISTADO 4-16. TEMPLATE TESTCASE. DATOS GENERALES.

```
<b>Control: </b><br>
{{#ask:[[EpisodeObject::+]][[Scenario::{{SUBJECTPAGENAME}}]]
        | mainlabel =-
        | headers = hide
        |?What does the participate do
        |?Who participates
        |{{#replace:?What the participate uses| |_}}
        |format=ul
}}
{{#arraydefine:allSubjects
        |{{#ask:[[EpisodeObject::+]][[Scenario::{{SUBJECTPAGENAME}}]]
        |mainlabel =-
        |headers = hide
        |?Who participates}}
}}
{{#arraydefine:allVerbs
        |{{#ask:[[EpisodeObject::+]][[Scenario::{{SUBJECTPAGENAME}}]]
        |mainlabel =-
        |headers = hide
        |?What does the participate do}}
}}
{{#arraydefine:allObjects
        |{{#ask:[[EpisodeObject::+]][[Scenario::{{SUBJECTPAGENAME}}]]
        |mainlabel =-
        |headers = hide
        |?What the participate uses}}
}}
```

LISTADO 4-17. TEMPLATE TESTCASE. LISTA DE TAREAS.

Luego se lista el flujo de control que debe tener la ejecución de tareas atómicas con sus correspondientes parámetros. Estas tareas provienen de la lista de episodios propios del escenario de la página actual.

Antes de hacer la invocación al subtemplate TestCaseData es necesario definir tres arreglos (Listado 4-17). El primero para almacenar todos los sujetos, el segundo todos los verbos y el tercero todos los objetos. La decisión de utilizar arreglos para este desarrollo es para aprovechar la indexación de esta estructura. De esta manera la posición *i* de cada uno de los arreglos corresponde a la terna propia de un episodio en particular, es decir cada uno de los elementos utilizados para enviar como parámetro de forma individual a las *n* invocaciones del subtemplate TestCaseData.

Seguidamente se procede a la invocación de las *n* veces al subtemplate TestCaseData con sus parámetros correspondientes (¡Error! No se encuentra el origen de la referencia.).

```
{{#vardefine: i | 0 }}
{{#while:
        | {{#ifexpr: {{#var: i }} < {{#arraysize:allVerbs}} | true }}
        | <nowiki />
        {{TestCaseData|param={{#var: i }}
        |item={{#vardefineecho: n | {{#expr: {{#var: i }} + 1 }} }}
        |iSubject={{#arrayindex: allSubjects | {{#var: i }} }}
        |iVerb={{#arrayindex: allVerbs | {{#var: i }} }}
        |iObject={{#arrayindex: allObjects | {{#var: i }} }}
}}
{{#vardefine: i | {{#expr: {{#var: i }} + 1 }} }}
}}
```

LISTADO 4-18. TEMPLATE TESTCASE. PREPARACIÓN DE PARÁMETROS PARA TESTCASEDATA.

Por último, se define el subtemplate TestCaseData el cual establece la estructura del pseudocódigo que determina la validación de un episodio en particular (Listado 4-19).

```
<includeonly>
== Method: M1{{{item}}} ==
<b>Task:</b>
{{#formredlink:target={{{iVerb|}}}|form=Verb}} ({{{iSubject}}},
{{#formredlink:target={{{iObject|}}}|form=Object}}
)
<br>
<b>Precondition:</b> <br> not _correctly_
{{#formredlink:target={{{iVerb|}}}|form=Verb}}
<br>
<b>Control:</b> <br> message("not correctly_{{{iVerb}}}, stop");<br>
stop;
</includeonly>
```

LISTADO 4-19. TEMPLATE TESTCASEDATA.

Ajustes generales de la herramienta

Además de las definiciones para la generación de las páginas concretas de la herramienta, es necesario definir cuestiones generales, tales como el menú, la imagen que identifica al sitio, la pantalla de bienvenida y los perfiles de usuarios.

Para el caso del menú lateral es posible editarlo accediendo a localhost/index.php/MediaWiki:Sidebar. Allí se puede referenciar tanto a páginas creadas por nosotros como páginas provistas por el motor de MediaWiki, tales como acceso a la página de ayuda de MediaWiki, la página para establecer el lenguaje o la versión impresa.

Para editar el logo del sitio es necesario agregarlo en la carpeta www/html/resources/assets, luego agregar el acceso al mismo en el archivo Localsettings.php.

La pantalla de inicio es posible editarla accediendo a localhost/index.php/Main_Page.

Por último, para crear cuentas de usuario, es necesario tener un usuario administrador, y acceder a la página especial localhost/index.php/Special:CreateAccount. Para asignar permisos de edición y eliminación entre otras operaciones, es necesario editar el archivo Localsettings.php y setear las variables $wgGroupPermissions, definiendo los grupos, usuarios y acciones permitidas a cada uno.

Capítulo 5 - MANUAL DE USO

En este capítulo se describe el funcionamiento y la especificación del desarrollo de la herramienta. Para ingresar al sitio es necesario contar con un usuario y contraseña. La página inicial cuenta con un mensaje de bienvenida y una breve explicación de las funcionalidades principales de la herramienta. En el sector izquierdo se encuentra el menú, provisto por las plantillas de MediaWiki, para nuestro caso está editado para mostrar los accesos a las búsquedas y a la carga de material referido a los modelos desarrollados para este trabajo. Para ello fue necesario acceder la página especial provista por Media Wiki llamada MediaWiki:Sidebar. La forma en que organiza el menú es con cuatro secciones, la primera con accesos directos a funcionalidades generales y propias de MediaWiki, tales como Home, Recent changes y Help. La segunda llamada "Categories" que permite listar todos los elementos cargados en el sitio agrupados por sus respectivas categorías. En el caso de las categorías "LEL Symbol", al tratarse de una categoría abstracta, es decir no es una categoría en sí, sino es padre de las subcategorías "Objects", "Subjects" y "Verbs", en vez de listar los elementos, lista estas tres subcategorías permitiendo que el usuario elija una de ellas. La tercera sección lista los enlaces a los formularios de carga de cada una de los elementos que son posibles crear. De la misma forma que ocurre con las categorías, en el caso de querer cargar un "LEL Symbol", la herramienta redirige a una página donde le da la opción al usuario de elegir qué tipo de símbolo LEL desea cargar, pudiendo ser "Objects", "Subjects" y "Verbs".

A continuación, se describe cada una de las categorías elegidas para almacenar la información basándose en la arquitectura que provee MediaWiki.

PROJECT

La clase Project es la que permite agrupar a todos los Escenarios que corresponden a un mismo proyecto. Esta clase tiene dos atributos, el nombre y la descripción. El nombre es dado como identificador de la página el cual se otorga al crear el proyecto, el segundo atributo es la descripción, que permite hacer un breve resumen sobre en qué consiste el proyecto.

En la Ilustración 5-1 se muestra cómo se crea un proyecto por medio de la opción del menú "Create" >> "Project". Al cargar el nombre del proyecto, el sistema tiene la inteligencia de determinar si el nombre elegido ya fue usado para crear una página. Si la página no existe la crea redirigiendo al formulario de carga correspondiente a la categoría "Project" (como muestra la Ilustración 5-2). Si la página ya existe existen a su vez dos posibilidades, que dicha página exista para la categoría "Project", de ser así redirige al formulario de edición, similar al formulario de carga, en el caso contrario advierte al usuario que se está editando una página con el formulario de carga equivocado.

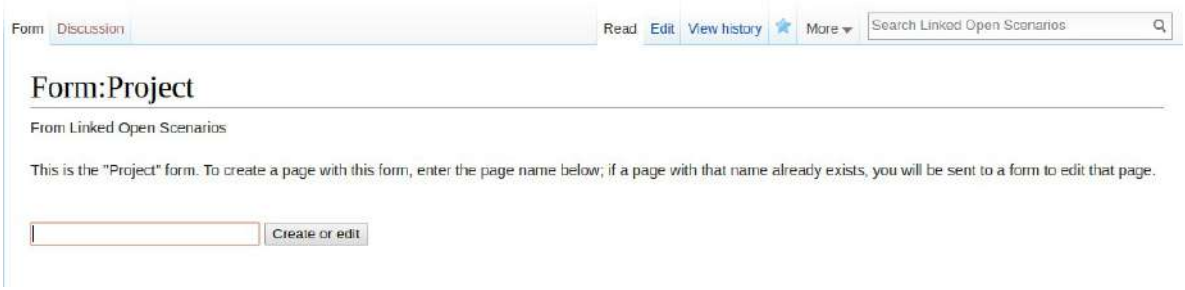


ILUSTRACIÓN 5-1. CREACIÓN DE UN PROYECTO.

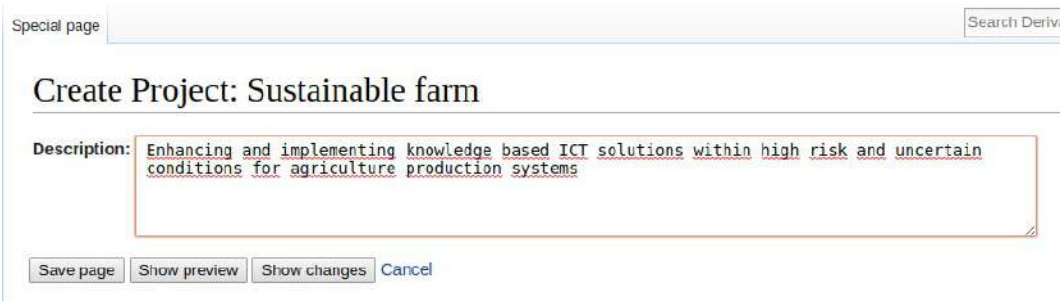


ILUSTRACIÓN 5-2. EDICIÓN DE UN PROYECTO.

Para este ejemplo se muestra la renderización de la página de un proyecto que tiene dos escenarios cargados (Ilustración 5-3).

ILUSTRACIÓN 5-3. PÁGINA DE UN PROYECTO CREADA.

SCENARIO

El proceso de elicitación de requerimientos es considerada una actividad iterativa sea cual fuera la estrategia o modelo de representación que se elija. La construcción de escenarios no es ajena a esta característica, es por ello que es normal no lograr identificar todos los actores, recursos, y conjunto de tareas que se realizarán para cada escenario. En cada ciclo de la iteración, el analista/programador encontrará detalles y elementos que alimenten cada especificación para completar el escenario.

Esta particularidad se ve reflejada en la herramienta propuesta de forma tal que el usuario podrá crear escenarios donde los actores identificados en primera instancia, podría no necesariamente participen de los episodios. En el sentido inverso, el usuario podría identificar un episodio donde participe un actor aún no incorporado a la descripción general del escenario. Esta misma idea sucede con los recursos.

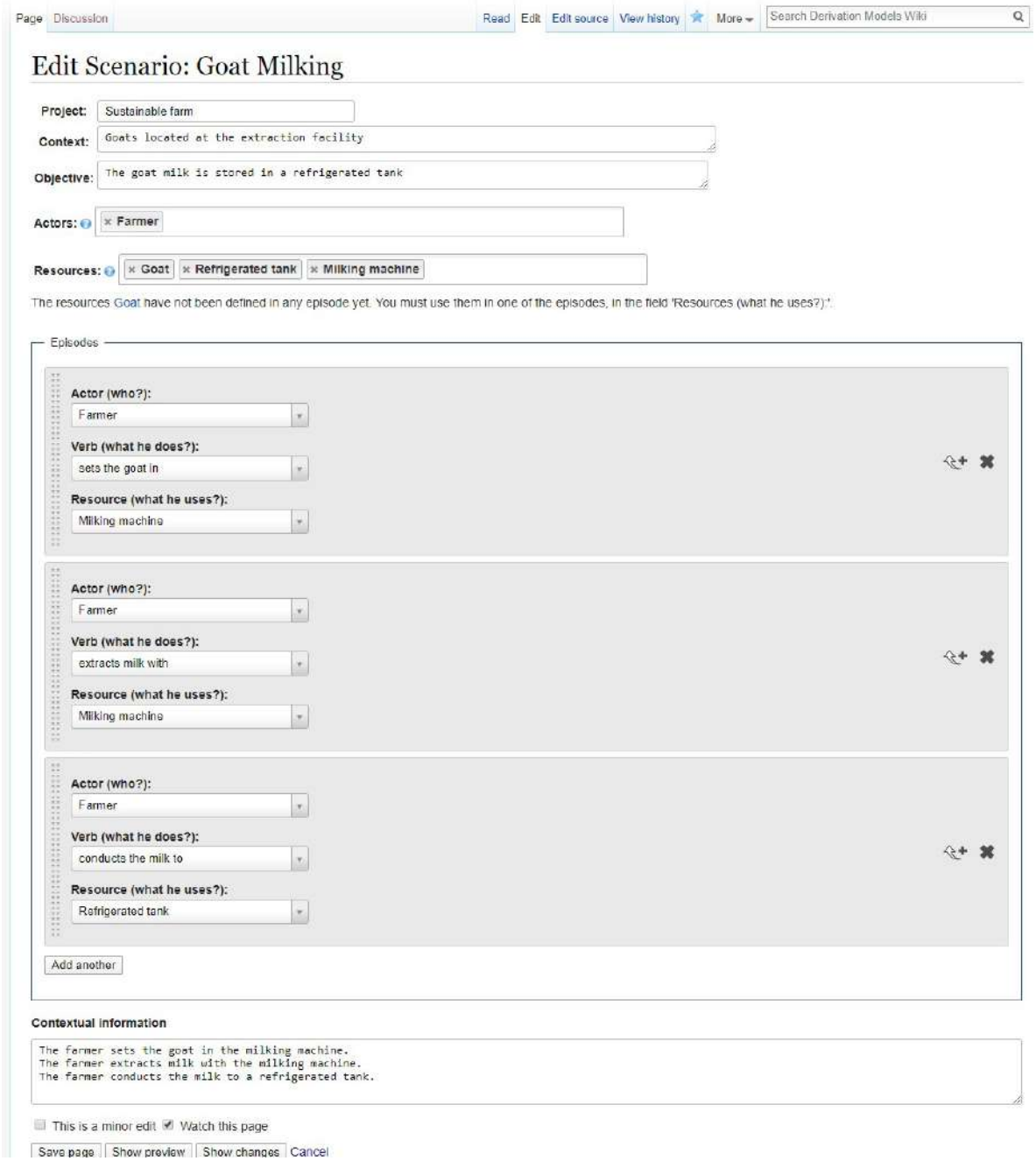


ILUSTRACIÓN 5-4. EDICIÓN DE UN ESCENARIO.

Para crear o editar un escenario se accede a la opción de menú “Create” >> “Scenario”. En caso de ser un nuevo escenario, se crea. Si el escenario ya existe, se edita el mismo (Ilustración 5-4).

La renderización de la página correspondiente a un escenario es la que se puede ver en la Ilustración 5-5. En primer lugar, se muestra cómo quedan los elementos cargados a partir del formulario enbebido con los templates correspondientes.

Luego en la Ilustración 5-6 se muestra el resultado de la derivación que surge a partir de los episodios y da como resultado, primero el flujo de ejecución de las tareas y luego el detalle de cada una de ellas, tomando como parámetros los actores y recursos definidos para los escenarios.

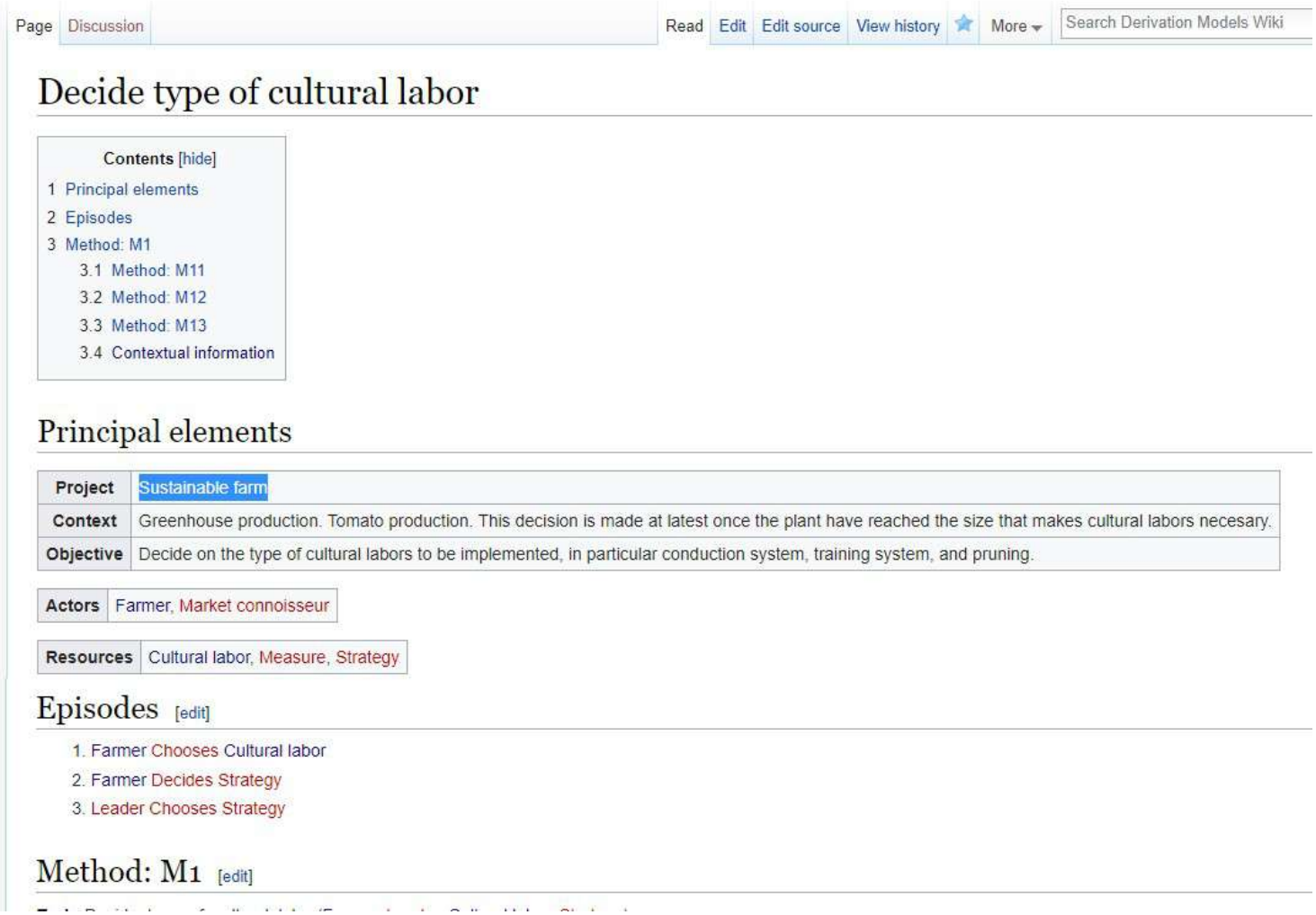
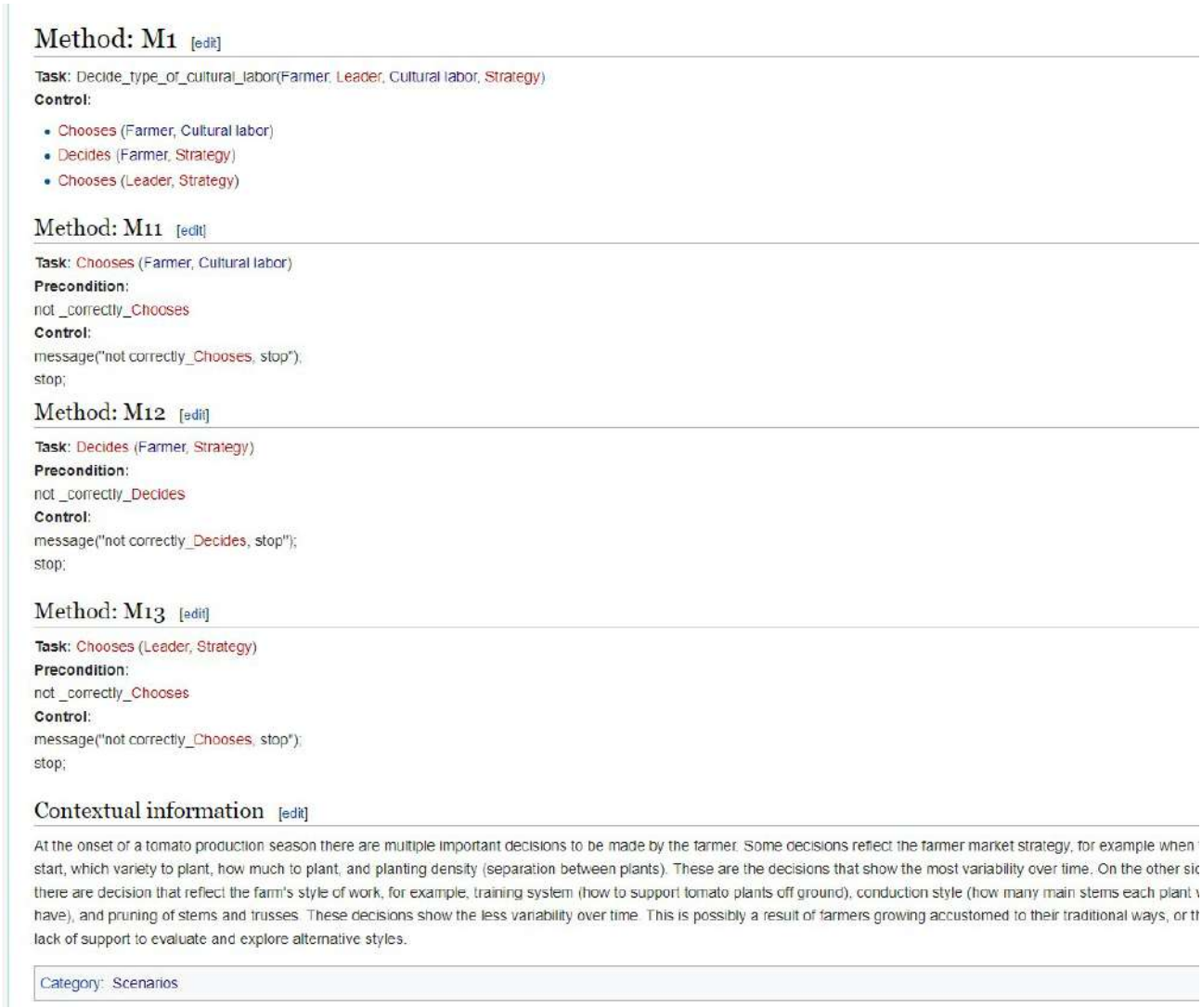


ILUSTRACIÓN 5-5. PÁGINA DE UN ESCENARIO CREADA.



Method: M1 [edit]

Task: Decide_type_of_cultural_labor(Farmer, Leader, Cultural labor, Strategy)
Control:

- Chooses (Farmer, Cultural labor)
- Decides (Farmer, Strategy)
- Chooses (Leader, Strategy)

Method: M11 [edit]

Task: Chooses (Farmer, Cultural labor)
Precondition:
not _correctly_Chooses
Control:
message("not correctly_Chooses, stop");
stop;

Method: M12 [edit]

Task: Decides (Farmer, Strategy)
Precondition:
not _correctly_Decides
Control:
message("not correctly_Decides, stop");
stop;

Method: M13 [edit]

Task: Chooses (Leader, Strategy)
Precondition:
not _correctly_Chooses
Control:
message("not correctly_Chooses, stop");
stop;

Contextual information [edit]

At the onset of a tomato production season there are multiple important decisions to be made by the farmer. Some decisions reflect the farmer market strategy, for example when
start, which variety to plant, how much to plant, and planting density (separation between plants). These are the decisions that show the most variability over time. On the other sic
there are decision that reflect the farm's style of work, for example, training system (how to support tomato plants off ground), conduction style (how many main stems each plant
have), and pruning of stems and trusses. These decisions show the less variability over time. This is possibly a result of farmers growing accustomed to their traditional ways, or th
lack of support to evaluate and explore alternative styles.

Category: Scenarios

ILUSTRACIÓN 5-6. PÁGINA DE UN ESCENARIO CREADA. RESULTADO DE LA DERIVACIÓN A TASK METHOD.

LEL SYMBOLS

Para la carga de símbolos LEL se accede a través de la opción de menú correspondiente a la subcategoría del símbolo que se desea cargar. La misma es “Create” >> “Subject”, “Create” >> “Verb” o “Create” >> “Object”. Si por algún motivo el usuario accede a la opción “Create” >> “LEL Symbol” (Ilustración 5-7), el sistema dirige a una página especificando que dicha categoría es padre de las tres subcategorías mencionadas anteriormente. Además, muestra los links correspondientes a cada una de ellas.

ILUSTRACIÓN 5-7. CREACIÓN DE UN SÍMBOLO LEL.

Tanto el símbolo de la subcategoría Subject como Verb y Object poseen el mismo formulario de carga. Se diferencian uno de otro en la categoría a la que pertenece dicha página (Ilustración 5-8).

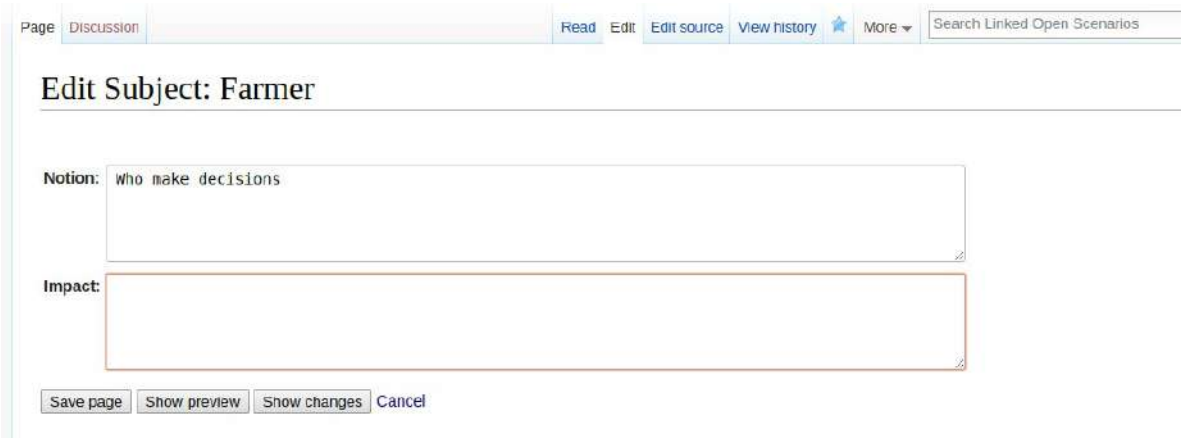


ILUSTRACIÓN 5-8. EDICIÓN DE UN SÍMBOLO LEL SUBJECT.

En la renderización de la página se muestran los datos cargados a través del formulario, es decir la noción y el impacto (Ilustración 5-9). Además, se muestran todos los episodios a los que pertenece dicho símbolo, gracias al entramado semántico provisto por las propiedades de MediaWiki.

ILUSTRACIÓN 5-9. PÁGINA DE UN SÍMBOLO LEL SUBJECT.

TEST DE USABILIDAD

En esta sección se describen los pasos para llevar a cabo el proceso de evaluación de usabilidad de la herramienta. Estos pasos comprenden desde los preparativos de las pruebas, elegir los usuarios de prueba hasta la evaluación de los resultados obtenidos.

El proceso fue validado con la metodología SUS (System Usability Scale, 2019). Se elige esta metodología porque resulta efectiva para diferenciar los sistemas usables de los no usables. Además, se pueden obtener resultados confiables con una pequeña muestra y es fácil de procesar los resultados. Originalmente fue creado por John Brooke en 1986 (Brooke, 1996), para evaluar todo tipo de productos y servicios, incluyendo software, hardware, sitios web o dispositivos.

SUS consiste en diez ítems con un rango de valores entre 1 y 5, que el encuestado debe responder considerando que 1 representa fuertemente en desacuerdo y 5 fuertemente de acuerdo respectivamente. La puntuación que obtendrá cada ítem variará entre 0 y 4. Para los ítems 1, 3, 5, 7 y 9 se debe restar 1 al valor dado por el encuestado. Para los ítems 2, 4, 6, 8 y 10 se debe restar 5 al valor dado por el encuestado. De esta forma cada valor se convierte en uno nuevo. Luego se debe sumar y multiplicar por 2,5 para convertir los valores originales de 0-40 a 0-100, lo que no quiere decir que represente un porcentaje. Basado en la investigación de Brooke, una puntuación superior a 68 se considera superior a la media.

En 2008 Bangor (Bangor, 2008) introdujo el cálculo de un promedio de los puntajes de SUS. Las preguntas originales se complementaron con una adicional sobre la facilidad de uso. Ésta se califica usando una escala de 6 puntos: 1) El peor imaginable; 2) pobre; 3) OK; 4) bueno; 5) excelente; 6) Mejor imaginable. En 2012 McLellan (McLellan, 2012) presentó los puntajes del SUS en otros valores: "No aceptable", 0-64; "Aceptable", 65-84 y "Excelente", 85-100. Finalmente, (Brooke, 2013) se llegó a nuevas conclusiones, luego de su definición original, en la que adoptó las ideas de Bangor.

Para este trabajo se han adaptado los ítems para usuarios con diferentes niveles de conocimientos en ingeniería de requerimientos, todos ellos usuarios habituales de distintas wikis.

Para llevar a cabo las pruebas se desarrolla un formulario con Google Forms con los diez ítems correspondientes a la metodología SUS.

- Me gustaría usar este sistema con frecuencia.
- Este sistema es demasiado complejo para lo que debe resolver.
- Pensé que era fácil de usar.
- Necesito que alguien con conocimiento me explique cómo usarlo.
- Las funcionalidades del sistema están bien integradas.
- Encontré muchas inconsistencias en el sistema.
- Creo que cualquier persona podría aprender a usar el sistema fácilmente.
- El sistema me resultó incómodo de usar.
- Me sentí seguro al usar el sistema.
- Necesitaba aprender muchas cosas antes de empezar a usar el sistema con fluidez.

Se ha elegido una muestra de usuarios en su mayoría entendidos del área de informática, y una pequeña proporción de usuarios entendidos en otras disciplinas, pero que han reconocido ser ávidos usuarios de wikis. Esta muestra intenta ampliar el campo respecto a los distintos puntos de vista que pueden aportar los diversos niveles de conocimientos técnicos en el área de la informática.

Una vez seleccionado este grupo, se les envía un mail con el link del formulario y se les da un plazo de 30 días para contestar. En el mail además se les explica que el objetivo del formulario es contribuir con el análisis y conclusiones del desarrollo de la herramienta en el marco de una tesina de licenciatura en sistemas.

Pregunta	Respuestas					
1	3	4	5	5	4	4
2	2	3	4	1	2	2
3	3	3	4	5	5	3
4	4	5	2	1	3	2
5	4	4	5	5	5	4
6	1	1	4	1	1	1
7	4	2	5	5	5	2
8	3	4	1	4	1	2
9	3	3	5	5	5	3
10	3	5	1	1	2	1
Scores	60	45	80	92.5	87.5	70
Puntaje Final	72.5					

TABLA 5-1. RESULTADOS DE LA EVALUACIÓN DE SUS.

Los usuarios seleccionados fueron diez, entre ellos uno sin relación con las ciencias de la informática, tres con conocimiento estrecho de los modelos de representación de requerimientos planteados en este trabajo. Los demás conocedores de las ciencias de las informáticas, pero no tan interiorizados con los temas planteados en este trabajo. De este último grupo sólo dos contestaron el cuestionario.

Luego de contestados los cuestionarios por todos estos usuarios seleccionados, se procede a la recolección de las respuestas. A partir de estas respuestas y del cálculo que propone el sistema SUS, se obtiene el valor 72.5. Según Brooke se puede decir entonces que es el sistema es usable. Si tomamos como referencia los valores planteados por (McLellan, 2012), se puede decir que, si bien el sistema es usable, no llega a ser excelente. Por lo tanto, podemos concluir que la herramienta se encuentra por encima de la media de los niveles de usabilidad, pero que podría evaluarse la forma de mejorar la usabilidad. En una interpretación personal, esto puede darse principalmente a que el aspecto de la misma es conocido por la mayoría de los usuarios de wikis, pero que la información no se encuentra de manera intuitiva, principalmente si los usuarios no reciben una aproximación previa de cómo se representa la información de los distintos modelos involucrados.

Capítulo 6 - CONCLUSIONES

Cuando se habla de desarrollo de software es muy común pensar inmediatamente en un lenguaje de programación, en un entorno de desarrollo y en capacidades puramente tecnológicas. Sin embargo, estas cuestiones abordan sólo una pequeña parte de lo que realmente significa el desarrollo de software. Cuando nace una idea, la cual se debe materializar a través de un software, se deben llevar a cabo, diferentes etapas. Estas etapas involucran a todas las partes interesadas, sean usuarios sin conocimientos tecnológicos o miembros del equipo de desarrollo con conocimiento técnico. Cabe destacar que con el desarrollo global de software los equipos y los stakeholders se encuentran distribuidos físicamente por lo cual, es necesario un modelo colaborativo para la captura de conocimiento. Ademas, se requiere que estos involucrados logren ponerse de acuerdo en cuanto al vocabulario con el que se expresan.

Principalmente cuando un usuario identifica una necesidad debe ser capaz de poder expresar claramente lo que desea. De la misma forma un ingeniero o desarrollador que no entiende en detalle el vocabulario propio del dominio de aplicación, debe poder interpretar las necesidades al mismo nivel que el usuario. En caso de no existir este entendimiento entre las partes, es muy probable que se cometan errores. Estos errores surgen en las etapas iniciales del ciclo de vida del software. Por lo tanto, en caso de no detectarse a tiempo, serán arrastrados a las etapas posteriores del ciclo de vida. Una especificación incorrecta de requerimientos causa errores ocultos, provocando un efecto dominó en las fases posteriores. Además de producir altos costos de recursos, llámense tiempo, dinero o personal dedicado a rehacer trabajo.

Las plataformas web brindan la infraestructura para el desarrollo de herramientas para utilizar cumpleindo con estas características. En particular, MediaWiki es una herramienta ampliamente difundida. Ahora bien, el desarrollo de software implica capturar el conocimiento del dominio y transformarlo sucesivamente a través de diferentes modelos hasta llevarlo al código.

En esta tesina, se propone un acercamiento a resolver todas estas cuestiones. Para ello se ha desarrollado una herramienta que permite capturar el conocimiento del dominio en etapas tempranas del ciclo de vida de desarrollo de software. Y la misma, permite transformar estos modelos tempranos, en elementos que aparecen al final del ciclo de vida, como ser los casos de prueba. De esta forma, y conforme al modelo en V de desarrollo de software, se vinculan productos del inicio y del fin del ciclo de vida. Así, con los casos de prueba, se puede volver a los productos previos para mejorarlos y enriquecerlos si fuera necesario. En particular, hemos utilizado Scenarios y un glosario específico, el LEL para describir los productos de etapas iniciales. La definición de Scenarios que hemos utilizado es la de Leite, la cual se complementa con la definición del glosario LEL, del mismo autor. Estos dos modelos propios de la etapa de elicitación de requerimientos son el punto de partida para realizar la derivación a los casos de prueba, los cuales, son especificados utilizando el modelo Task Method.

Una parte importante de la tesis, es la derivación de modelos. Si bien, en esta tesis se derivan casos de prueba a partir de LEL y Scenarios, la infraestructura brindada permite con poco esfuerzo, poder definir otros modelos y realizar distintas derivaciones. Se ha utilizado la plataforma MediaWiki con extensiones de manejo semántico para la definición de los modelos, y por intermedio de queries semánticos se posibilita la derivación de los mismo. La elección de esta herramienta se debe a que es muy sencillo instalar y

configurar el ambiente de desarrollo. MediaWiki está desarrollado con código abierto en php y en su instalación cuenta con archivos de configuración que permiten personalizar el sitio. Por otro lado, nos permite crear una herramienta basada en un sitio conocido por la mayoría de los usuarios de internet, ya que todos alguna vez utilizamos Wikipedia. Esto permite que cualquier usuario pueda aprender fácil y rápidamente su utilización.

La derivación de modelos tiene dos pilares. Por un lado, una serie de reglas de derivación, las cuales se podría aplicar en forma manual. Sin embargo, este trabajo arduo y propenso a errores necesita ser automatizado. Es por ello que los queries semánticos provistos por MediaWiki y la extensión semántica, permiten ejecutar las reglas. Este es el aspecto más sobresaliente de la contribución. Se provee una infraestructura que permite definir modelos (en esta tesis ejemplificados con Scenarios, LEL y casos de prueba), y a través de reglas implementadas por queries semánticos, permiten transformar ciertos modelos en otros.

Cabe destacar que se han encontrado algunas desventajas en esta propuesta. Una de ellas es que, si bien la base de MediaWiki está desarrollada en php, el usuario interactúa a través de un lenguaje denominado Wikitext, que es un lenguaje de markup poco flexible. Para un desarrollador acostumbrado a lenguajes procedurales u orientados a objetos, puede encontrar muy lejana la lógica que mantiene WikiText, incluso para desarrolladores de html, el lenguaje de markup más utilizado. Ahora bien, en esta tesis, se provee la definición y derivación de ciertos modelos. Para extenderlos, es necesario definir formularios y queries semánticos interactuando con WikiText. Como trabajos futuros, se plantea definir un método de extensión, en el cual, el usuario final (ya sea experto del dominio o desarrollador) pueda hacer estas extensiones a través de una interfaz más amigable.

Capítulo 7 - REFERENCIAS BIBLIOGRÁFICAS

(Agile Manifiesto, 2019) Sitio oficial. http://agilemanifesto.org/principles.html. Recuperado: Julio 2019.

(Antonelli, Rossi, Leite y Oliveros, 2012) Antonelli L., Rossi G., Leite J.C.S.P., Oliveros A. (2012). *Deriving requirements specifications from the application domain language captured by Language Extended Lexicon.* Memorias de Workshop en Ingeniería de Requerimientos WER 2013, Buenos Aires, Argentina.

(Ansible, 2018) Sitio oficial de Ansible. https://www.ansible.com/. Recuperado: agosto 2018.

(Antonelli, Rossi, Leite y Oliveros, 2013) Antonelli L., Rossi G., Leite J.C.S.P., Oliveros A. (2013). *Buenas prácticas en la especificación del dominio de una aplicación*, Memorias del XVI Workshop de Ingeniería en Requisitos WER 2013, Montevideo, Uruguay. pp. 80–92.

(Antonelli, Rossi y Oliveros, 2016) Antonelli L., Rossi G., Oliveros A (2016), *A Collaborative Approach to Describe the Domain Language through the Language Extended Lexicon*, Journal of Object Technology, Bs.As. Argentina.

(Bangor, 2008) Bangor A., Kortum P.T. y Miller J.T. (2012). An empirical evaluation of the system usability scale. *Intl. Journal of Human–Computer Interaction* 24.6, pp. 574-594.

(Berners-Lee, 1998) Berners-Lee T. (1998). *Semantic Web Road Map, W3C Design Issues.* Recuperado de: http://www.w3.org/DesignIssues/Semantic.html.

(Berners-Lee y Fischetti, 2001) Berners-Lee T. y Fischetti M. (2001), *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web.*

(Berners-Lee, Hendler y Lassila, 2001) Berners-Lee T., Hendler J. y Lassila O. (2001). The Semantic Web, Scientific American. The Semantic Web: A New Form of Web Content That is Meaningful to Computers Will Unleash a Revolution of New Possibilities.

(Booch, 1994) Booch G., (Septiembre-Octubre 1994). Scenarios. Report on Object Oriented Analysis and Design, Vol 1, 3.

(Boehm y Papaccio, 1988) Boehm BW. Papaccio PN. (Octubre 1988). *Understanding and controlling software costs. IEEE transactions on software engineering, Volumen 14.* pp 1462-1477.

(Borst, Arana, Crave, Galeano, 2005) Borst I., Arana C., Crave S., Galeano N., (Octubre 2005). *Technical Report* (Deliverable) D62.2 ICT-I Business Models.

(Brooke, 1996) Brooke J., (1996). SUS-A quick and dirty usability scale. Usability evaluation in industry. pp 189(194) 4-7

(Brooke, 2013) Brooke J., (2013). SUS: a retrospective. *Journal of usability studies* 8.2, pp.29-40

(Camilleri, Soubie y Zalaket, 2003) Camilleri G., Soubie J.L. y Zalaket J. (2003), *TMMT: Tool Supporting Knowledge Modelling,* in *Knowledge-Based Intelligent Information and Engineering Systems*, (2773), pp. 45–52.

(Dijkstra, 1972) Dijkstra EW. (Octubre 1972). *The Humble Programmer, Communications of the ACM, Volumen 15, Issue 10,* New York, USA pp 859-866.

(Gutwin, Greeberg y Roseman, 1996) Gutwin, C., Greenberg, S., & Roseman, M. (1996). Workspace awareness in real-time distributed groupware: framework, widgets, and evaluation. In People and computers XI (Proceedings of the HCI'96).

(Hadad, Kaplan, Oliveros y Leite, 1997) Hadad, G., Kaplan, G., Oliveros, A. y Leite, J.C.S.P. (1997). *Construcción de Escenarios a partir del Léxico Extendido del Lenguaje.* JAIIO'97, SADIO Buenos Aires. pp 65-77.

(Hendler, 2001) Hendler J. (2001). *Agents and the Semantic Web.* DOI: 10.1109/5254.920597.

(IEEE, 1990) IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering.

(Leite, 1989) Leite, J.C.S.P., Application Languages: A Product of Requirements Analysis, Departamento de Informática, PUC-/RJ, January 1989.

(Leite y Franco, 1990) Leite, J.C.S.P. y Franco, A.P.M. (Octubre 1990) *O uso de Hipertexto na Elicitaçao de Linguagens da Aplicaçao.* Anais de IV Simpósio Brasilero de Engenharia de Software, SBC, Brasil. pp 134-149.

(Leite y Franco, 1993) Leite J.C.S.P., Franco A.P.M. (1993). *A Strategy for Conceptual Model Acquisition*, RE'93, First Intl Symposium on Requirements Engineering, IEEE Computer Society Press, Los Alamitos, EEUU. pp 243-246.

(Leite y Oliveira, 1995) Leite J.C.S.P y Oliveira A.P. (1995). *Procedings of 1995 IEEE International Symposium on Requirements Engineering*, RE'95, York, UK.

(Leite, Rossi, Balaguer, Maiorana, Kaplan, Hadad y Oliveros, 1997) Leite J.C.S.P, Rossi G., Balaguer F., Maiorana V., Kaplan G., Hadad G. y Oliveros A. (1997) *Enhancing a requirements baseline with scenarios. Requirements Engineering* 2(4). pp 184-198.

(Leite, Hadad, Doorn y Kaplan, 2000) Leite J.C.S.P., Hadad G.D.S.,Doorn J.H., Kaplan G.N.(2000). *A Scenario Construction Process*, Requirements Engineering Journal, 5.

(LNCS, 2009) Laboratorio Nacional de Calidad de Software. INTECO. (2009). Ingeniería del Software: Metodologías y Ciclos de Vida. España.

(Lowry A. y Lowry M., 2004) Lowry A.C.P.B. y Lowry M.R. (2004), *Building a taxonomy and nomenclature of collaborative writing to improve interdisciplinary research and practice*. J. Bus. Commun, 41(1), pp. 66–69.

(McLellan, 2012) McLellan S., Muddimer A. y Peres, S.C. (2012). The effect of experience on System Usability Scale ratings. *Journal of usability studies* 7.2, pp. 56-67.

(Mendes Calo, Estevez y Fillottrani, 2009) Mendes Calo K., Estevez E. C., y Fillottrani P. R. (2009). *Un framework para evaluación de metodologías ágiles.* San Salvador de Jujuy, Jujuy, Argentina. pp 1-10.

(Mizuno, 1983) Mizuno Y. (Marzo 1983). *Software Quality Improvement, IEEE Computer, Volumen 16*, No. 3. pp 66-72.

(Montserrat, Páez , Arias , Rivadeneira, Vilanova y Miranda, 2012) Montserrat C., Páez A., Arias C., Rivadeneira,S., Vilanova G., Miranda M. (2012). *El modelado de procesos como técnica de elicitación de requerimientos.* II EIPA. Santa Cruz. Argentina.

(MediaWiki) Sitio oficial de MediaWiki. Recuperado: Octubre 2018.

(Pressman, 2002) Pressman R. S. (2002), *Ingeniería del Software: Un Enfoque Práctico*. 5ª Edición. McGraw-Hill. cap 1.

(Sommerville, 2005) Sommerville I. (2005), Ingeniería del Software. 7ª Edición. Madrid: Pearson Addison Wesley.

(Trichet y Tchounikine, 1999) Trichet F. y Tchounikine P. (1999), *DSTM: A framework to operationalise and refine a problem solving method modeled in terms of tasks and methods*, International Journal of Expert Systems With Applications, Elsevier Science, 16(2). pp 105–120.

(System Usability Scale) Sitio https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html. Recuperado: Octubre 2019.

ANEXO I - CONFIGURACIÓN Y PUESTA EN FUNCIONAMIENTO

La herramienta se desarrolla sobre el sistema operativo Ubuntu. Sobre un motor de wiki existente provisto se realizan las configuraciones. Para configurar la wiki a nuestras necesidades lo hacemos a partir de un archivo "playbook.yml" que se ejecuta con el programa Ansible (Ansible, 2018).

Ansible es un automatizador de tareas, también suele denominarse orquestador de tareas. Permite generar los despliegues (deploy) teniendo en cuenta los distintos entornos, como pueden ser "Desarrollo", "Test" y "Producción". También permite automatizar la instalación de todos los programas que deseen utilizar, sus configuraciones particulares y cómo se relacionan entre sí. Permite especificar usuarios, claves, permisos de usuario y definir reglas de firewall. Todo esto lo hace usando YAML, un lenguaje de programación muy simple que permite describir las tareas de automatización de una manera parecida al inglés. Estas tareas son pequeños programas que pueden estar alojados en cualquier parte de la máquina, no requieren servidores ni bases de datos, se ejecutan por defecto a través de SSH con la terminal que se desee.

En el archivo "playbook.yml" se enumeran las siguientes tareas:
-Elección de usuario y password para la base de datos.
-Elección de usuario y password para el servidor.
-Elección de usuario y password para el administrador de la wiki.
-Instalación de PhpMyAdmin y todos los paquetes básicos.
-Configuración de MySql.
-Instalación de Nodejs.
-Extracción del comprimido con la base de la wiki.
-Creación de la carpeta local donde se alojará la aplicación.

Como se trabaja con el sistema operativo Ubuntu, para comenzar a realizar las instalaciones, primero ejecutamos:

$ sudo apt-get update

$ sudo apt-get install ansible -y

$ sudo apt-get install phyton -y

La primer línea nos asegura que los paquetes de la máquina estén actualizados para no tener problemas de incompatibilidad con el resto de los programas a instalar. Las siguientes dos líneas instalan Ansible (automatizador de tareas) y Phyton que es el lenguaje sobre el cual corre Ansible.

Una vez instalado lo necesario, se debe descomprimir el archivo que contiene la wiki base provista. Antes de instalarla es necesario setear los archivos que contienen las tareas que se ejecutarán con Ansible,

con los parámetros propios de nuestra aplicación, como pueden ser usuarios y claves de la máquina u otros parámetros que se quieran personalizar.

Seguidamente se procede a ejecutar:

$ sudo ansible-playbook playbook.yml --connection=local

Este comando instala toda la aplicación con las configuraciones establecidas según se detalló.

Para el desarrollo de nuestra herramienta es necesario, además instalar Extensiones de MediaWiki, las cuales permiten que nuestra wiki sea más robusta y útil para diferentes fines. Las extensiones pueden tener distintos niveles de complejidad y están agrupadas en distintas categorías entre las que se destacan las del tipo Parser Tags, Parser Function, Special Pages, Skins, Authentication, entre otras.

Para importarlas a nuestra wiki debemos tener un usuario administrador del sitio de la wiki que estamos creando, acceder al repositorio de https://www.mediawiki.org y descargarla. No todas las extensiones se instalan de la misma manera, pero cada una cuenta con las instrucciones de cómo hacerlo desde su página. Generalmente, se descargan como carpetas en un archivo comprimido el cual debe ubicarse en la carper miMaquina/extensions. Por último en el archivo LocalSettings.php ubicado en miMaquina/var/www/html agregar una línea del estilo:

wfLoadExtension('ExtensionName')

Para algunas extensiones es necesario incluir la siguiente línea en lugar de la recién mencionada.

require_once "$IP/extensions/Arrays/Arrays.php";

La alternativa a esta instalación es agregarla en en los archivos.yml de Ansible para automatizarlo al instalar la wiki desde cero.

El próximo paso es crear las estructuras para mantener, mostrar y habilitar la adición y edición de los datos. De las extensiones mencionadas, Page Forms es la que permite crear los formularios con contenido semántico que necesitamos para crear páginas.

El primer paso para usar esta extensión es definir la estructura de datos, el tipo de páginas necesarias, los datos contenidos en cada una, establecer qué datos se podrán ser editados. Una vez que tengamos eso (idealmente, escrito), crearemos las páginas. Para ello crearemos propiedades semánticas, templates, formularios y categorías provistos por Media Wiki y las extensiones anteriormente mencionadas.

En base a la estructura definida empezaremos a crear las páginas para el desarrollo de la herramienta, la cual se conformará por un conjunto de Propiedades semánticas, Formularios, Plantillas (Templates) y Categorías.

Lo primero que crearemos serán las propiedades semánticas.

Propiedades: Son la base de cualquier siti1o semántico y crea las conexiones entre los datos. Una propiedad se utiliza para especificar un solo bloque de información sobre el tema de esta página. Para crearla accederemos a la url

"*http://localhost/index.php/Special:CreateProperty*". Pueden ser de distintos tipos como Text, Number, Boolean, Page entre otras, siendo la última el valor por defecto de todas las propiedades, aunque siempre debe especificarse explícitamente y quiere decir que esa propiedad tendrá un espacio de nombres a través de la URL "*http://localhost/index.php/Property:unaPropiedad*".

Templates: (o Plantilla) configura la visualización de los datos en una página, contiene la inteligencia para convertir los datos en información semántica y, en caso de necesitarlo, define la página como perteneciente a una determinada categoría y, por lo tanto, a un determinado tipo de página. En general, habrá una plantilla por tipo de página, aunque a veces un solo tipo de página contendrá más de una plantilla. Para crearlas se accederá a la URL "*http://localhost/index.php/Special:CreateTemplate*". Para mostrar todos los templates creados se accederá a la URL "*http://localhost/index.php/Special:Templates*".

Formularios: son creados para permitir a los usuarios agregar y editar fácilmente páginas de varios tipos. Debe haber un formulario por tipo de página; el formulario llenará la plantilla o plantillas que contiene este tipo de página. De la misma forma que se ha visto anteriormente para crearlos se accede mediante la URL "*http://localhost/index.php/Special:CreateForm*" y para mostrarlos se accederá a la URL "*http://localhost/index.php/Special:Forms*".

Categorías: Las páginas creadas a partir del formulario se renderizan según el template con el que se relacione, el cual debe pertenecer a una categoría concreta. Crearemos una página para cada categoría, y se debe especificar un formulario por defecto para que cada página se pueda editar automáticamente con el mismo formulario que lo creó. Las crearemos accediendo a la URL "*http://localhost/index.php/Special:CreateCategory*" y se mostrarán todas la categorías a través de la URL "*http://localhost/index.php/Special:Categories*".

ANEXO II - PUBLICACIONES RELACIONADAS

An extension to scenarios to deal with business cases for the decision-making processes in the agribusiness domain

L. Antonelli[1], G. Camilleri[2], C. Challiol[1,3], A. Fernandez[1,4], M. Hozikian[1], R. Giandini[1,4], J. Grigera[1,4], A.B. Lliteras[1,4], J. Martin[1], D. Torres[1,4,5], P. Zarate[6]

[1] Lifia - Fac. de Informatica, Universidad Nacional de La Plata,
50 esq 120, La Plata, Bs As, Argentina, {leandro.antonelli, cecilia.challiol, alejandro.fernandez, marian.hozikian, roxana.giandini, julian.grigera, alejandra.lliteras, jonathan.martin, diego.torres}@lifia.info.unlp.edu.ar
[2] SMAC group, IRIT, 118 route de Narbonne, 31062 Toulouse Cedex 9, France, camiller@irit.fr
[3] also CONICET. Argentina.
[4] also CICPBA Pcia. Bs As. Argentina.
[5] also Depto. CyT, Universidad Nacional de Quilmes. Bernal. Argentina.
[6] ADRIA group, IRIT, Université de Toulouse, 2 rue du Doyen Gabriel Marty, 31042 Toulouse Cedex 9, France, zarate@irit.fr

Abstract. With the aim of pushing innovation through information and communication technology in the agri-business field, working closely with farmers is essential. It is especially important to systematically capture their knowledge in order to analyze, propose and design innovation artifacts (in terms of software applications). In this article, we use Scenarios to capture the knowledge of the experts that is elicited in early meetings previous to the definition of requirements. At those early stages, there are many uncertainties, and we are particularly interested in decision support. Thus, we propose an extension of the Scenarios for dealing with uncertainties. Scenarios are described in natural language, and it is very important to have an unbiased vocabulary. We complement Scenarios with a specific glossary, the Language Extended Lexicon that is also extended to decision support. According to V-model life cycle, every stage has a testing related stage. Thus, we also propose a set of rules to derive tests from the Scenarios. Summing up, we propose (i) an extension to Scenarios and the Language Extended Lexicon templates, (ii) a set of rules to derive tests, and (iii) an application to support the proposed technique. We have applied the proposed approach in a couple of case studies and we are confident that the results are promising. Nevertheless, we need to perform a further exhaustive validation.

Keywords. Scenarios, Uncertainties, Decision Support, Agri-Business, LEL

1 Introduction

Agricultural processes are complex by nature because they rely on unpredictable conditions as weather or market demand, as well as human decision on biased opinions and incomplete information. Many people participate in the processes, usually with different objectives, background, experience and level of studies. Thus, it is hard to obtain a complete and accurate understanding of the whole process. This is the motivation of the RUC-APS project [9], a

European funding project dealing with Enhancing and implementing Knowledge based ICT solutions within high Risk and Uncertain Conditions for Agriculture Production Systems.

Requirements engineering is one of the most important stages in software development. Errors made at this stage can cost up to 200 times to repair if they are discovered when the software is delivered to the client. There are two approaches to elicit requirements: classic and agile. Use Cases are a widely used tool to describe software requirements in a classical approach. There are different templates with different level of detail to use according to the definition of the requirements. Nevertheless, when the definition of requirements is vague, agile methods use User Stories to discover the requirements in an iterative and incremental way while the software application is developed.

In both cases, with classic and agile development, the client should have a clear idea of the role of the technology and a vision about the software artifact that he needs. Although the requirements of the software are mainly described during the requirements engineering phase, there are some early meetings previous to the requirements stage to discuss needs, wishes and expectation. In these meetings, the objectives and boundaries of the software application are defined. Following these definitions, the requirements engineering stage can be performed to analyze and describe requirements (either with Use Cases or User Stories for example).

It is important to have tools to capture the information of the early meetings. Moreover, when stakeholders are not aware of how technology can help, it is necessary to support them. In this case, the IT team needs to learn about the domain and make proposals about innovation. We have been participating in the RUC-APS project with the aim of providing innovation in information and communication technology (ICT) to agriculture. In this period we have learned that agriculture is a field with no much integration with ICT. And it is a field with many uncertainties.

Some uncertainties are related to decision that farmers have to take and once taken, it cannot be changed. For example, the conducting system of the plants is related to make plants go upward or go down (as a bush). It should be defined before planting because both conducting systems need different distance between the plants. After the decisions made, it cannot be changed. Another decision is the training system to use. If it is decided that the plants go upward, some string is needed to help the plant to go upward. Besides a string, others elements can be used. Each element has different advantages and disadvantages. Then, the pruning system establishes how to cut the plant in order to allow it to go upward or down. All these decisions are related among them. Nevertheless, it is quite impossible to evaluate the impact of one decision on another. So, the decision has to be made one by one in a progressive process.

Others uncertainties are related to everyday situations that should be evaluated to react in consequence. For example, the temperature monitoring of a greenhouse must be set in order to keep it constant. We have seen that both types of uncertainties are captured in early meetings. Specifically, we are interested in innovation in Decision Support Systems, that is, to take a decision on the first type of uncertainties. Nevertheless, we also consider the second type of uncertainties.

In this article, we propose to capture the knowledge obtained from early meetings through Scenarios. We present an extension to the Scenarios to deal with uncertainty. We also propose to complement the description of the Scenarios with a particular glossary, the Language Extended Lexicon (LEL). An unbiased language is very important to understand the scenarios. We also propose an extension to the LEL to deal with uncertainty in order to capture business knowledge through Scenarios and complement them with LEL. According to the V-model development life cycle, the product obtained in each step (requirements, design, and codification) should be tested. Thus, we also propose a set of rules to automatically generate tests from Scenarios. We propose a set of rules to derive tests from Scenarios (only from Scenarios, not from LEL) in order to fulfill with the V-model. These tests derived from Scenarios should be used as input to design tests for the requirements.

The paper is organized in the following way. Section 2 introduces LEL and Task/Method models we use in this work. Section 3 details our contributions, i.e. a process to capture Scenarios and LEL, as well as the extensions to the Scenarios and the LEL to deal with uncertainty. Section 3 also presents the rules to derive tests from Scenarios using the task/method model as the specification. Finally, Section 4 discusses some conclusions.

2 Background

This section describes the base elements used in our approach. It describes the original template of Scenarios and LEL that we use to capture the knowledge about business cases. And it also describes the Task/Method model, the technique used to describe tests derived from the Scenarios through the set of rules proposed. The topics of this section are the base for our approach. In the next section, the original template of Scenarios and LEL are extended to deal with business cases for the decision-making processes in the agribusiness domain. In addition, in a further section, the Task/Method model is used to describe the proposed rules of our approach.

2.1 Scenarios

Scenarios describe interactions between users and a future system [17]. It is also used to understand the context of the application because they promote the communication when there is a great variety of experts [6].

Leite [12] defines a scenario with the following attributes: (i) a title; (ii) a goal or aim to be reached through the execution of the episodes; (iii) a context that sets the starting point to reach the goal; (iv) the resources, relevant physical objects or information that must be available, (v) the actors, agents that perform the actions, and (vi) the set of episodes.

The following Scenario describes the activity of determining cultural labors for tomato production. It is important to mention that the scenario describes some task with uncertainty because many decisions have to be taken. Although people (the farmers and their leader) take the decision, some software application can be used to support the process. Moreover, the last episode of the scenario is related to describe the definitions arrived according to some standard. This task can also be supported by an application software since the application can receive the information about the decisions, organize and present according to the standard. Moreover, the task of writing the procedure can also be tested to assure if the report produced by a future application satisfies or not the standard.

Listing 1. Scenario about cultural labors

Title: Determine cultural labors
Goal: Decide the conducting system, the training system and the pruning policies that should be used
Context: Tomato production
Resources: conducting system techniques, training system techniques, pruning policies, standard to describe procedures, procedures for the cultural labors
Actors: farmers, leader
Episodes:
The farmers and their leader decide a conducting system to use
The farmers and their leader decide a training system to use
The leader establishes the pruning policies to apply
The leader writes a procedure according to the standard describing the conducting system, the training system, and the pruning policies

2.2 Language Extended Lexicon

The Language Extended Lexicon (LEL) is a glossary used to capture and describe the domain´s language [11]. Terms (also called symbols) are classified into four types: Subject, Object, Verb, and State. Subjects represent an active element that performs actions. Objects are passive elements on which subjects perform actions. A verb is used to represent the actions. Finally, States represent situations in which subjects and objects can be located. A symbol is described by two attributes: (i) notion and (ii) behavioral responses. Notion describes the symbol denotation and explains its literal meaning. While Behavioral responses describe its connotation, that is, the

effects and consequences of the relationship between the defined symbol and others symbols defined in the LEL [18].

The following examples describe one term of each symbol category. It is important to remark the example in Listing 4. The verb describes the activity of controlling the temperature of the greenhouse. This situation presents some uncertainty since the temperature can change and the farmer should act in consequence in order keep it within a specific range.

Listing 2. LEL subject: farmer

Subject: Farmer
Notion: Person that grows tomatoes in a shared lot with other farmers
Behavioral responses:
The farmer participates in the determination of the cultural labors
The farmer plant the tomatoes
The farmer grows the tomatoes
The farmer control the temperature of the greenhouse

Listing 3. LEL object: greenhouse

Object: Greenhouse
Notion: Place to grow tomatoes in a controlled environment.
Behavioral responses:
The farmer plant tomatoes in a greenhouse
The farmer control the temperature of the greenhouse

Listing 4. LEL verb: Control the temperature of the greenhouse

Verb: Control the temperature of the greenhouse
Notion: Action of monitoring the temperature in order to maintain it between certain range
Behavioral responses:
The farmers monitors the temperature
The farmers aerate the greenhouse to descend the temperature
The farmers close the windows of the greenhouse to increase the temperature

Listing 5. LEL state: Flowering

State: Flowering
Notion: Phenological state of the tomato, characterized by the appearance of leaves and flowers.
Behavioral responses:
The tomato change to fruition state after the appearance of the first fruit.

2.3 The Task/Method model

The Task/Method model is a knowledge modeling paradigm that considers the reasoning as a task [19] [21]. Its main advantage is to have a declarative form to express knowledge which can be easily processed by tools such as execution engines [4].

A Task/Method model is composed of two sub-models: (i) the domain model and (ii) the reasoning model. The domain model describes the objects of the world that are used by the reasoning model. The reasoning model describes how a task can be performed. It uses two modeling primitives: (i) task and (ii) method.

A task is a transition between two world state families (an action) and is defined by the following attributes: (i) name that describes the task, (ii) par, a typified list of parameters handled by the task, (iii) objective, the goal state of the task, (iv) method, describes one way of performing a task.

A method is characterized by the following attributes: (i) heading, the identification of the task achieved by the method, (ii) preconditions which must be satisfied to be able to apply the method,

(iii) effects, consequences of a successful application of the method, (iv) control, achievement order of the subtasks, and (v) subtasks. This paper focuses on some of the attributes described before. A full description of this modeling paradigm is performed by Camilleri et al. in [4] [5].

We use the Task/Method model in order to describe the tests that should be applied to the Scenarios. Then, the Task/Method model is processed by an execution engine to finally test the Scenarios. Listing 6 shows the example of a Task/Method model to test the scenario of determining cultural labor described in Listing 1. The Scenarios are descriptions in natural language, while the Task/Method model is a computer language, that is why some changes must be done to the names. For example, while "conducting system" is valid in a Scenario, in Task/Method should be translated as "conductingSystem". Moreover, Scenarios are too wide and abstract, not all the situation can be tested. In this example, we only define to test the method of writing the procedures in order to verify if the description of the procedures satisfies some standard. The procedures are definition from the decision of conducting system, training system and pruning policies performed previously. It is important to mention that Task/Method model is a hierarchical model, describing decompositions of tasks using other tasks. That is why the test is finally performed in method M14 (the fourth line in M1) of Listing 7.

Listing 6. Method determine cultural labor

```
Method: M1
Task: determineCulturalLabor
Control:
Decide (farmers, leader, conductingSystem);
Decide (farmers, leader, trainingSystem);
Establish (leader, pruningPolicies);
Write (leader, procedures, standard, conductingSystem, trainingSystem,
pruningPolicies);
```

Listing 7. Method write procedures according to the standard

```
Method: M14
Task: Write (leader, procedures, standard, conductingSystem, trainingSystem,
pruningPolicies)
Control:
  message("procedures do not conform to writing standards, stop");
stop;
```

3 Proposals

This section describes the contribution of this paper: (i) an extension to Scenarios and the Language Extended Lexicon templates, and (ii) a set of rules to derive tests from Scenarios.

We propose to capture the actual knowledge obtained from early meetings through Scenarios. Nevertheless, it is very hard to collect domain information in software development. Thus, the Scenarios extended by the proposed approach help to acquire hypothetical and unclear situations, so as to convert the Scenarios in real based and concrete Scenarios. In order to deal with the uncertainty, we extend the Scenarios with 4 more attributes: key decision, variables (frozen and contextual), identified risks and factors of uncertainty. These attributes capture relevant information to make a decision. For example "determine the conduction system" is a key decision. In Listing 1 it was defined as part of the episodes, but in fact, this element should be captured as a key decision. Then, the LEL is used to complement the Scenarios, and we also proposed an extension of the LEL to deal with uncertainty. For example, the temperature of the greenhouse is a variable that change according to the weather conditions (its context) and some actions must be done to maintain it in a specific range. This information should be captured by the LEL.

Scenarios and LEL are elements used to capture the business knowledge in early meetings. According to the V-model, this should be done on the top level of the V-model. Thus, in order to fulfill with the V-model, we also propose a set of rules to derive test from Scenarios. It is important

to mention that the derivation is performed only from Scenarios, the LEL is not considered in this paper.

Thus, we provide techniques to use in the top level of the V-model (the business level): Scenarios (complemented with LEL) and business test (derived from Scenarios through a set of rules). These both elements can be used to produce the elements of the following level (requirements level). That is, Scenarios should be used to describe requirements, and business tests should be used to describe acceptance test.

The rest of the section is organized in the following way. First, a collaborative process to capture the knowledge to describe Scenarios is presented. Then, the extensions to the Scenarios and the LEL are described. Finally, the rules to derive tests described in Task/Method model from Scenarios are detailed.

3.1 A collaborative and iterative process to capture knowledge

This section describes an iterative and incremental process to capture the knowledge from the stakeholders in an early stage of software definition. In that early stage, there are many uncertainties, that is why the most critical elements to capture are the potential decision and eventual variabilities that should be considered.

The process begins with the definition of the Scenarios by a multidisciplinary team. During the description of the Scenarios, it is common that specific terms of the domain appear, that are used by the experts of the domain but are unknown to the technical team members. These terms should be described in the LEL. Scenarios are used to capture multiple views and promote communication among stakeholders [6]. Moreover, a multidisciplinary team helps to obtain a broader perspective to define scenarios.

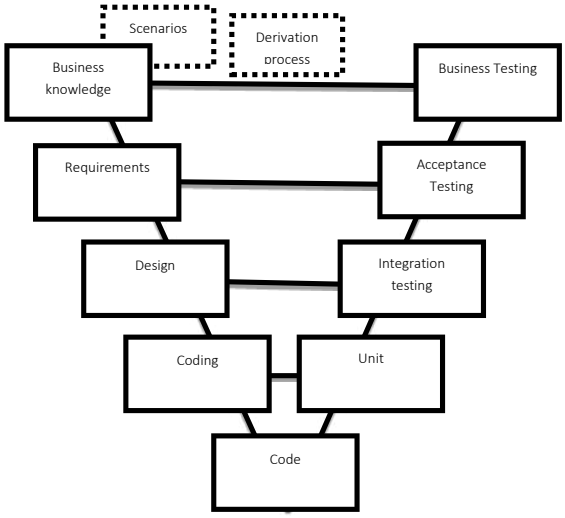


Fig. 1. Contribution located on the top of the V-model

It is important to state that the proposed process consists of describing mainly Scenarios, and describe terms in the LEL only if necessary. Traditionally, LEL is defined completely first and then Scenarios are described [12].

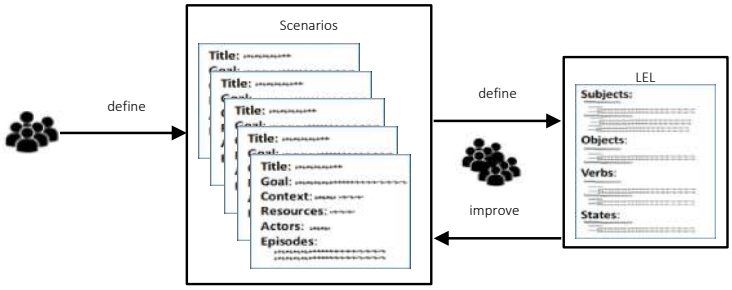


Fig. 2. Scenarios' definition

3.2 Extensions to Scenarios and LEL

The background section describes the original proposal to describe Scenarios and LEL. In order to support the capture of the knowledge in a very early stage of software definition, with many uncertainties and lack of precision, this section describe the attributes added to Scenarios and LEL.

3.2.1 Scenarios extended to deal with uncertainty

In early meetings to discuss the incorporation of technology, it is important to add to the Scenarios information about factors of uncertainty, that is, the doubts that must be clarified later. We propose to add the following attributes: (i) key decisions, (ii) variables, (iii) risks, and (vi) factors of uncertainty.

Lupetti et al. [14] incorporate the concept of variables at an abstract level and categorize them according to their participation in Scenario design decisions. On the other hand, when a business process problem in decision support system is proposed, the work of De Maio et al. [7] stresses the importance of context-aware heterogeneous fuzzy consensus model learning from the past executions. So, there is a need to feed and maintain a knowledge base storing the associations between contextual variables, key decisions, and weights for each decision maker. Based on this reasoning, this article proposes to incorporate variables and key decisions in the description of the Scenarios.

Some uncertainties are related to decision that farmers have to take and once taken, it cannot be changed. For example, the conducting system mentioned before. Others uncertainties are related to everyday situations that should be evaluated to react in consequence. For example, the monitoring of the temperature of a greenhouse mentioned before. The first type of decisions are fixed when they are defined, and it makes possible to define more specific Scenarios considering that definition. For example the layout of the plantation can be discussed considering certain conducting system. While the second type, depends on situation that varies according the context.

Thus, we consider two types of variables: (i) frozen variables and (ii) contextual variables. Frozen variables are decisions that should be taken and they will not change from that moment. For example, the conduction system. Contextual variables refer to a decision that should be taken very often. For example, the monitoring of the temperature.

Both types of variables are important to decision support. People decide about the frozen variables, thus, they need tools to support the decision. While contextual variables are important to design the software application considering that it must react according to the variable.

In order to analyze the key decision, more information about the variables is needed. Both types of variable, mainly frozen variables, depends on risks and uncertainties. A risk is anything that could potentially impact your plan [16]. Thus, if a frozen variable for a decision is related to defining the growing space (indoor or outdoor), although the historical temperature can support the decision to grow outdoor, according to the geographical situation, there could exist the risk of unexpected low temperature.

The uncertainty concept is closely related to risks. A risk is an event that could potentially occur, thus, there is a measurable probability of occurrence. For example, there is a chance of 0.15 of low temperature during the harvest. But the factor of uncertainties is related to elements that there is no historical information, or cannot be estimated or predicted [10].

Listing 8 describes each proposed attribute with some information related to determine cultural labor scenario.

Listing 8. Extension to the Scenarios

Key decisions:
conduction system (How many main stems each plant will have?)
training system (How to support tomato plants off ground?)
pruning policies (When to start the pruning of stems and trusses to comply with the conduction system decisions? How often to prune?)

Variables
Frozen: soil type, seed type, geographic area, growing space (outdoor, greenhouse)
Contextual: diurnal temperature, nocturnal temperature, natural lighting

Identified risks: unexpected climate phenomena: frost, stronger winds, flood risk, low temperatures

Factors of uncertainty: plant disease, market demand, water pollution

Taking decisions (give values to frozen variables and deal with contextual variables) is related to analyzing and balance risk and uncertainties. From a classic point of view, it is necessary to analyze the risk involved in the key decisions, perform a quantitative and qualitative analysis of the uncertainties in order to make the decision [22] [23]. Of course, this decision could be made in different ways, but it is important to have documented the information to analyze in order to make the decision. That is the objective of adding these attributes to the Scenarios.

3.2.2 LEL extended to deal with contextual information

We propose to use Scenarios to capture uncertainty in order to take a decision. That is, experts and IT team analyze the scenarios and discuss alternatives to take a decision. Nevertheless, the uncertainty that the scenario represent could be a cause and effect relation, with different situations and different actions. In this case, is no need to take a decision, because all the alternatives should be considered by the application software. This, is another interpretation of decision making: a context-aware behavior, where relevant variables (context-features) take different values (representing specific situation) to trigger some decisions [7]. The context-aware behavior could present uncertainty when it is not possible to estimate or measure one or more variables' values. So, it is not possible to establish which decision should be triggered [3].

Litvak et al. [13] present an extension of LEL to provide more expressivity for Verbs (such as an action of an effect of), but, this is not enough to represent context-aware behaviors. Fortier et al. [8] model context-features and trigger decisions (handlers) as a first-class citizen due to the complexity involved in this kind of applications. Using the concepts defined in [8], this article proposes an extension to LEL that would help to define contextual information in a common vocabulary.

The LEL originally categorizes symbols in Subject, Object, Verbs, and States. This article proposes two new categories: Context-Feature and Contextual Decisions. The context-feature category allows representing each contextual feature associated to Subjects or Objects. Context-Features' values could be simple values (e.g. sensed data) or more complex elements [8]. Contextual Decision allows to represents for each situation (defined by a context-features' values) the list of triggered actions. The Given-then specification [24] can be used to describe behavioral responses for Contextual Decisions.

Let's consider the following examples. A Context-Feature "Temperature of greenhouse" which defines for some interval values trigger Contextual Decisions (Listing 9). And its Contextual Decision definition "Evaluate Temperature Decision" (Listing 10) that determines the action to be performed.

Listing 9. LEL Context-Feature: temperature of the greenhouse

Context-Feature: Temperature of the greenhouse Notion: Range of temperature values measured in the greenhouse Behavioral responses: Value < 10 ºC, Evaluate Temperature Decision Value between 10º C and 25º C, No Action Value > 25º C, Evaluate Temperature Decision

Listing 10. LEL Context-decision: evaluate temperature decision

Context-decision: Evaluate temperature decision Notion: When temperature is high or low, worker leader should be notified Behavioral responses: Given (Temperature of greenhouse < 10 ºC or Temperature of greenhouse > 25º C) Then Notify to worker leader

3.3. Test derivation from Scenarios

We propose five rules to derive test from the Scenarios [2]:

Rule 1. Tasks Identification: each verb in the Scenario's episodes is translated into a task in Task/Method model. Each Scenario title is also a task in Task/Method model. Listing 11 shows the example.

Listing 11.

Title: Determine cultural labors → Task: DetermineCulturalLabors Episodes The farmers and their leader decide a conducting system to use → Task: Decide The farmers and their leader decide a training system to use → Task: Decide The leader establishes the pruning policies to apply → Task: Establish The leader writes a procedure according to the standard describing the conducting system, the training system, and the pruning policies → Task: Write

Rule 2. Task's Parameters Identification: each actor and resource used in the episodes of a Scenario is translated into a parameter in Task/Method model. Listing 12 shows the example.

Listing 12.

The farmers and their leader decide a conducting system to use → Task: Decide (farmers, leader, conductingSystem) The farmers and their leader decide a training system to use → Task: Decide (farmers, leader, trainingSystem) The leader establishes the pruning policies to apply → Task: Establish (leader, pruningPolicies) The leader writes a procedure according to the standard describing the conducting system, the training system, and the pruning policies → Task: Write (leader, procedures, standard, conductingSystem, trainingSystem, pruningPolicies)

Rule 3. Episode's method: the episodes part of a scenario is translated by a method in Task/Method model. Listing 13 shows the example. Since the main Scenario is translated into the method named M1, each episode of the Scenario is translated into a method M1#.

Listing 13.

```
The farmers and their leader decide a conducting system to use → Method: M11
The farmers and their leader decide a training system to use → Method: M12
The leader establishes the pruning policies to apply → Method: M13
The leader writes a procedure according to the standard describing the conducting
system, the training system, and the pruning policies → Method: M14
```

Rule 4. The Sequence of tasks: the sequence of different lines in the episodes part of a Scenario determines the sequence of tasks in the control part of a Task/Method model method. The use of expressions like "then", "after", etc... in the episodes of a Scenario determines also a sequence of tasks in the method's control part. Listing 14 shows the example.

Rule 5. Test Case Method: We assume that each test case (Test cases part of scenario) corresponds to an achievement status (succeed or fail) of the task. In a failure situation, the scenario will stop. This stop case will be represented by a method for the next task in which the precondition field corresponds to the test case failure. The example was shown in Listing 7.

Listing 14.

```
Episodes:
…
The leader establishes pruning policies
The leader writes a guide…
Or
… The leader establishes pruning policies, then the farmers' leader writes a guide…

Method: M1
Task: determineCulturalLabor
Control:
…
Establish (leader, pruningPolicies);
Write(leader,procedures,standard, conductingSystem, trainingSystem, pruningPolicies);
```

4 Conclusions

We have presented a proposal to use Scenarios at an early stage of software development, when there are many uncertainties and the software is not defined yet. Also, gathering knowledge from farmers can be difficult, since they don't always organize their ideas in a way that's useful for requirements, so the proposed extensions to Scenarios and LEL, as well as the rules to derive tests, help acquiring information from them.

We enriched the Scenarios with some attributes that capture critical information to help stakeholders to take a decision and perform a further requirements definition of a software application. We complement the Scenarios with a particular glossary, the Language Extended Lexicon that we also extended to deal with uncertainty. According to the V-Model, in which every software development phase has a related testing stage, we also propose a technique to derive test from the Scenarios. Thus, we are providing a technique to deal with uncertainty and decisions on the top level of the V-model life cycle.

We have also built software applications to manage all the information. A Media Wiki platform [15] is used as a repository of the Scenarios and LEL. A semantic media Wiki extension was also added to allow the semantic support and the creation of forms in order to make the CRUD operations (create, retrieve, update and deleted) in a more user-friendly way. Then, in order to

provide support to the derivation of tests, the tool relies on a Natural Language Processor Framework [20] and on a task automation tool and administrator of configurations [1].

This work was motivated with the aim of providing decision support to the agri-business field in the context of the RUC-APS project. Although we have done some preliminary validation of the proposed strategy, we are planning to develop some pilot project in order to conduct case studies and to obtain feedback for improvement and validation of the proposal.

Acknowledgement. Authors of this publication acknowledge the contribution of the Project 691249, RUC-APS: Enhancing and implementing Knowledge based ICT solutions within high Risk and Uncertain Conditions for Agriculture Production Systems (www.ruc-aps.eu), funded by the European Union under their funding scheme H2020-MSCA-RISE-2015.

References

1. Ansible IT automation (2018), https://www.ansible.com/
2. Antonelli L, Camilleri G, Grigera J, Hozikian M, Sauvage C, Zaraté P (2018) A Modelling Approach to Generating User Acceptance Tests, International Conference on Decision Support Systems Technologies (ICDSST 2018), Heraklion, Greece.
3. Borodin V, Bourtembourg J, Hnaien F, Labadie N (2016) Handling uncertainty in agricultural supply chain management: A state of the art. European Journal of Operational Research, 254(2), pp 348-359.
4. Camilleri G, Soubie JL, Zalaket J (2003) TMMT: Tool Supporting Knowledge Modelling. In Knowledge-Based Intelligent Information and Engineering Systems, vol. 2773, 2003, pp. 45–52.
5. Camilleri G, Soubie JL, Zaraté P (2005) Critical Situations for Decision Making: A Support Based on a Modelling Tool. In Group Decision and Negotiation, Springer Verlag, Vol. 14 N. 2, pp 159-171.
6. Carroll, JM (2000) Five reasons for scenario-based design. Interacting with computers 13.1. doi: 10.1016/S0953-5438(00)00023-0. pp 43-60.
7. De Maio C, Fenza G, Loia V, Orciuoli F, Herrera-Viedma E (2016) A framework for context-aware heterogeneous group decision making in business processes. Knowledge-Based Systems 102 (2016) pp 39–50.
8. Fortier A, Rossi G, Gordillo SE, Challiol C (2010) Dealing with variability in context-aware mobile software. Journal of Systems and Software, 83(6), pp 915-936.
9. Hernandez, J., Mortimer, M., Patelli, E., Liu, S., Drummond, C., Kehr, E., Calabrese, N., Iannacone, R., Kacprzyk, J., Alemany, M. and Gardner, D., (2017) RUC-APS: Enhancing and implementing Knowledge based ICT solutions within high Risk and Uncertain Conditions for Agriculture Production Systems. In 11th International Conference on Industrial Engineering and Industrial Management, Valencia, Spain.
10. Khodakarami V, Fenton NE, Neil M (2007) Project scheduling: improved approach to incorporate uncertainty using Bayesian networks. Project Management Journal, 38(2), PP 39–49.
11. Leite JCSdP, Franco APM (1993) A strategy for conceptual model acquisition. In Requirements Engineering conference. IEEE. doi:10.1109/ISRE.1993.324851, pp 243–246.
12. Leite JCSdP, Rossi G, Balaguer F, Maiorana V, Kaplan G, Hadad G, Oliveros A (1997) Enhancing a requirements baseline with scenarios. In requirements Engineering. Vol 2.4. doi: 10.1109/ISRE.1997.566841. pp 184-198.
13. Litvak CS, Hadad GDS, Doorn JH (2018) Nominalizations in Requirements Engineering Natural Language Models. In Encyclopedia of Information Science and Technology, Fourth Edition. IGI Global. pp. 5127-5135.
14. Lupetti ML, Gao J, Yao Y, Mi H (2017) A scenario-driven design method for Chinese children edutainment. In proceedings of the Fifth International Symposium of Chinese CHI. ISBN: 978-1-4503-5308-3 doi>10.1145/3080631.3080636. pp 22-29.
15. Media Wiki (2018), https://www.mediawiki.org
16. Project Management Institute (2017) Project Management Body of Knowledge.
17. Potts C (1995) Using schematic scenarios to understand user needs. In proceedings of the 1st conference on Designing interactive systems: processes, practices, methods, & techniques. ACM. doi: 10.1145/225434.225462
18. Sarmiento E, Leite JCSdP, Almentero E (2014) C&L: Generating model based test cases from natural language requirements descriptions. In Requirements Engineering and Testing (RET), 2014 IEEE 1st International Workshop on. IEEE, 2014. doi: 10.1109/RET.2014.6908677

19. Schreiber G, Akkermans H, Anjewierden A, De Hoog R, Shadbolt NR, Wielinga B (2000) Knowledge Engineering and Management: The CommonKADS Methodology, vol. 99.
20. Stanford Natural Language Processing (2018), https://nlp.stanford.edu
21. Trichet F, Tchounikine P (1999) DSTM: A framework to operationalise and refine a problem solving method modeled in terms of tasks and methods, Expert Syst. Appl., vol. 16, no. 2, pp. 105–120.
22. Trutnevyte E, Guivarch C, Lempert R, Strachan N (2016) Reinvigorating the scenario technique to expand uncertainty consideration Climatic Change Volume 135, Issue 3, pp 373-379.
23. Weaver CP, Lempert RJ, Brown C, Hall JA, Revell D, Sarewitz D (2013) Improving the contribution of climate model information to decision making: the value and demands of robust decision frameworks, WIREs Climate Change vol 4 pp 39-60.
24. Wynne M, Hellesoy A, Tooke S (2017) The cucumber book: behavior-driven development for testers and developers. Pragmatic Bookshelf.

Semantic Support for Scenarios to Improve Communication in Agribusiness

Leandro Antonelli[1], Diego Torres[1,2,3],
Mariángeles Hozikian[1], Jorge E. Hernandez[4,5]

[1] Lifia – Facultad de Informatica, Universidad Nacional de La Plata, Argentina

[2] CICPBA – Comision de Investigaciones Cientificas de la Provincia de BsAs, Argentina

[3] Departamento de Ciencia y Tecnologia, Universidad de Nacional de Quilmes, Argentina

[4] School of Management, University of Liverpool, United Kingdom

[5] Universidad de La Frontera, Temuco, Chile

{leandro.antonelli, diego.torres, marian.hozikian}@lifia.info.unlp.edu.ar

J.E.Hernandez@liverpool.ac.uk

Abstract. Organizations produce and exchange a huge amount of critical information, which main purpose is to obtain acceptable results. Hence, the trend is by considering integrated systems that can be easily adapted to several domains, especially when they need to exchange information. In this context, the agribusiness sector is a good example where massive data is generated, which implies the need for information sharing and collaboration, where the great challenged is support and understand the colliding context. However, every software system relies on its context, with its own rules, dynamism, and languages. Hence, it implies a significant effort to have a complete understanding of the composed domain. For this purpose, scenarios are well-known tools to describe dynamic domains and are commonly described under text-based context. When different stakeholders build Scenarios, it is essential to review them in order to unify their description. Thus, Scenarios under this unified perspective will better support the analysis and identification of relationship between two or more domains. This analysis is the key to design mechanisms to exchange information. Therefore, in the light of this, this paper proposes a semantic definition of Scenarios and a set of queries to identify issues in the Scenarios and improve their quality. In addition to this, a wiki platform to implement the semantic support and the queries is also provided.

Keywords: Agribusiness, Requirements, Scenarios, Ontologies

1 Introduction

Nowadays, there is a huge level of integration between different software systems. Everyone produces a big amount of data and different organizations share this information to improve their results [5]. Collaboration is needed in every sector. Food and agribusiness are not an exception. Their supply chains are pioneers in the use of massive data, sometimes due to rigorous legislation that force to trace lots of variables along the supply chain [14]. Scenarios are well-known tools to describe situations of the domain [2]. They can be used to capture the context of different applications to identify their relationship. Thus, it is possible to establish a mechanism to make the applications to exchange information.

Nevertheless, it is not an easy task to design a mechanism to interoperate two different applications already developed [5]. Every software system relies on its context, with its rules, dynamic, and language. Scenarios should use the language of the stakeholders since the stakeholders are the ones that describe them. Thus, Scenarios need to be described with narrative text [7]. However, it can be hard to identify joints points in Scenarios that are described by two different groups of stakeholders that belong to two different contexts [11].

There are some quality attributes that good specification must satisfy: completeness, consistency, unambiguity, and correctness [6]. Completeness means that no piece of a specification can be missed because some absence can lead to suppositions. Consistency means that the different points of view should provide a unified description. Unambiguous is related to the use of terms and expressions that should be carefully chosen in order to avoid misunderstanding. Finally, correctness is related to assure that the description satisfy the reality. That is, there is no gap between the intended meaning and the specification.

Scenarios are used to understand the context of the application since they promote communication when there is a great variety of experts [2] [10]. Scenarios should be written carefully in order to satisfy the quality attributes. Nevertheless, it is challenging to achieve this goal [12]. Scenarios have been historically described by only one person, the requirements engineer who elicited the knowledge, organized it and produced a homogenous specification [7]. This classical view is being replaced by a collaborative model, where every stakeholder contributes directly to the specification [4]. Let us consider the expression “cultural labor”. In the agricultural domain, it refers to some task (labor) to take care of the plants (cultures). Nevertheless, the expression can also refer to some artistic (cultural) activity (labor).

A semantic support helps to improve the quality of narrative descriptions [3]. An ontology description is a semantic mechanism that relates every relevant syntactic element (for example, nouns and verbs) to a semantic element [13]. For example, a homonym could be related to two different ontology elements. Thus consistency and unambiguity can be improved [15][1]. Moreover, ontologies can be described in semantic tools that make possible automatic processing to infer conclusions. For

example, let consider the following sentences: “A tomato is a vegetable” and “Any vegetable needs irrigation.” A semantic query can conclude that “A tomato needs irrigation.”

Different approaches use ontologies as a body of knowledge to create scenarios in many domains. To our knowledge, there are no approaches that create ontologies from narrative scenarios to improve their quality. In this paper, we propose a semantic description of the Scenarios, a set of semantic queries, and a tool support for them. This contribution provides an automatic processing of the Scenarios to help to improve their quality regarding consistency, ambiguity, completeness and correctness. The proposal identifies issues in the description of the Scenarios while stakeholders are describing them. Thus, the stakeholders alerted by the tool can discuss the issues among them in order to improve their shared knowledge and consolidate it in the Scenarios.

This knowledge makes possible the analysis of the colliding areas captured in Scenarios to design an interoperation mechanism. This paper only focuses on identifying issues to improve the quality of the Scenarios. Nevertheless, this is a crucial step to design an interoperation mechanism. Commonly, every organization has its own culture (language, techniques, and process). Thus, when two organization need to interoperate, they need to share the same culture. It is important to mention, that it is also needed in differents working group in the same organization. The rest of the paper is organized in the following way. Section 2 describes the template of the Scenario. Section 3 presents the semantic definition. Section 4 proposes semantic queries to identify issues. Section 5 describes the tool. Section 6 discusses some conclusions.

2 Scenario template

Leite [7] defines a Scenario with the following attributes: (i) a title that identifies the Scenario; (ii) a goal to be reached through the execution of the episodes; (iii) a context that sets the starting point to reach the goal; (iv) the resources, relevant physical objects or information that must be available, (v) the actors, agents that perform the actions, and (vi) the set of episodes, smaller task (that could also be described as a Scenario) to accomplish the goal

Listing 1 and 2 provide examples. The domain used is a farm that grows vegetables, but it also breeds animals in order to be ecologically self-sufficient as well as profitable. The goat milking Scenario (Listing 1), describes some basic steps to obtain milk from the goats. The actors and resources attributes should be used in the episodes, although it is possible that episodes mention actors and resources not mentioned in these both attributes due to the iterative construction of the Scenarios. That is, in a first step, some stakeholder identifies a Scenario describing its title, then other stakeholders describe the main actors and resources, and finally some other with more knowledge describes the set of episodes. The Cheesemaking Scenario (Listing

2) is related to the Goat milking Scenario because the milk obtained with the first Scenario is used to produce cheese. This relation is showed in the context of the Scenario Cheesemaking and the goal of the Scenario Goat Milking.

3 Semantic definition of the Scenarios

This section describes the ontology designed for providing a semantic description of the Scenarios. Using the proposed ontology, stakeholders can keep using an iterative and incremental approach to describe the Scenarios, but the ontology will provide support to identify inconsistencies.

The description uses the main principles of the OWL language [8]. We defined six main semantic concepts that are described as classes. The first one is the Scenario. Then, some attributes of the Scenario are also classes: Actor, Resource, and Episodes. Finally, there are two different attributes (Goal and Context) that are described with the same class: Condition. Each of the class concepts has the following intent:

Scenario: It is the core conceptualization in the ontology. It has a title, a data property defined as a string. Additionally, Scenario includes a Goal and a Context, both of them are Conditions. The Context is the pre-condition to perform the Scenario while the Goal is the postcondition. The Scenario also contains actor, resources, and episodes, steps that could be atomic actions represented by Episodes, or more complex ones, described as Scenarios.

Condition: It represents a situation, and it is used to describe goals (the desired situation to achieve) and context (needed situation to allow the execution of the Scenario).

Actor: It represents the subject that is in charge of the Episodes actions and the owner of the scenarios.

Resource: It represents the resources that are used in the episodes by the actors.

Episode: It represents each task that the actor performs with some resource. Thus, the episode is related to an actor, a resource and a verb. Moreover, the episode is related to a previous episode that must be completed.

Action: It represents the main action of an episode. It is important to mention, that the semantic representation of the action not only consider a verb, but it could also be a more complex expression that provides an accurate description of the domain.

Scenario: Goat Milking

Goal: The goat milk is stored in a refrigerated tank.
Context: Goats located at the extraction facility
Resources: goats, refrigerated tanks, milking machine
Actors: farmer
Episodes:
The farmer sets the goat in the milking machine
The farmer extracts milk with the milking machine.
The farmer conducts the milk to a refrigerated tank.

Listing 1. Goat milking Scenario

Scenario: Cheesemaking
Goal: To have cheese to sell and obtain money to run the farm
Context: The goat milk is stored in a refrigerated tank.
Resources: Milk
Actors: Cheesemaker
Episodes:
The cheesemaker curdles the milk with lactic ferments
The cheesemaker adds rennet to the milk
The cheesemaker drains the milk in mussels
The cheesemaker salts the milk
The cheesemaker leaves the milk to refine for 24 hours

Listing 2. Cheesemaking Scenario

Figure 1 shows the different classes and the dependencies between them. The figure uses the Scenarios described in Listing 1 and 2. The Scenario Goat Milking (Listing 1) is completely described, while the figure only describes the elements of the Scenario Cheesemaking (Listing 2) that are related to the first one. That is the case of the condition “The goat milk is stored in a refrigerator tank”, shared as a goal and a context. Then, it is important to mention that the actions are complex expressions, for example: “conducts the milk to,” instead of referring only to a verb.

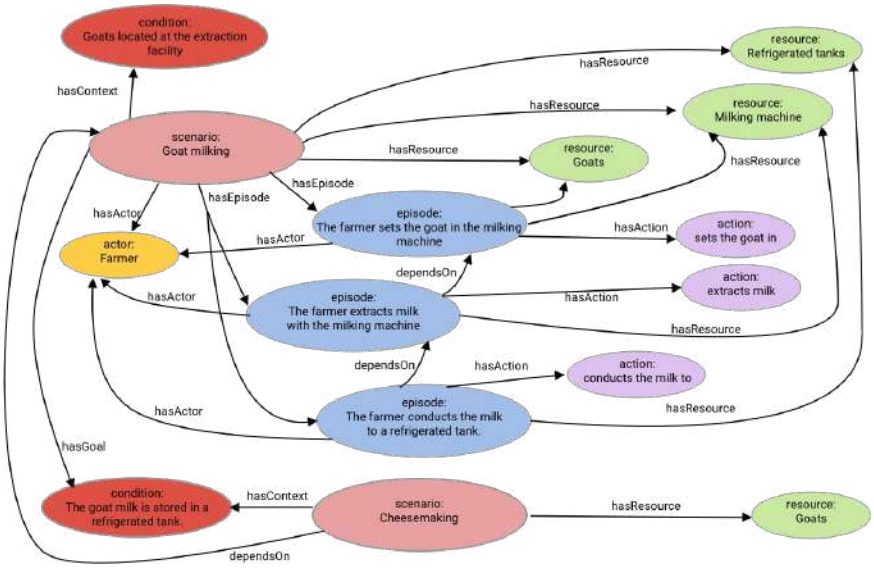


Figure 1. Classes and dependencies

4 Semantic queries to identify issues

This section describes the semantic artifacts that allow the stakeholders to check requirements quality attributes as completeness and consistency. These queries should be checked constantly along with the collaborative definition of Scenarios. Thus, when some issue is identified, an alert is shown explaining the issue, so that it can be fixed. The rest of this section describes five semantic queries conceptually and we also present a SPARQL query definition.

Query 1. Consistency between actors and episodes

All the actors included in the attribute actor of the scenario should be mentioned in at least one of the episodes. That is, if an actor a belongs to the scenario s, there should be an episode (or scenario which is an episode of s) that refers to the actor a. Because of the iterative and incremental description of the Scenarios, it is not necessary to check that all the actor mentioned in the episodes should be listed in the attribute actor. The SPARQL query detailed in Listing 3 shows the list of actors that are inconsistent for the <scenario>. If the query returns an empty list, it represents the lack of actors and episodes inconsistency.

For example, Listing 4 shows a new version of the Cheesemaking Scenario (partially described) that has actors and episodes inconsistency because the actor farmer is not mentioned in any episode. The query applied to the example will return a list with farmer.

```
1. SELECT ?actors WHERE {
2. {<scenario> hasActor ?actor}
3. MINUS{
4. <scenario> hasEpisode ?episode.
5. ?episode hasActor ?actor.}}
```

Listing 3. SPARQL query to detect inconsistency between actors and episodes.

Scenario: Cheesemaking
Actors: farmer
Episodes:
The cheesemaker curdles the milk with lactic ferments
The cheesemaker adds rennet to the milk

Listing 4. A scenario with inconsistency between actors and episodes

Query 2. Consistency between resources and episodes

All the resources included in the attribute resource of the scenario should be mentioned in at least one of the episodes. That is, if a resource r belongs to the scenario s, there should be an episode (or scenario which is an episode of s) that refers to the resource s. This query is similar to the previous one. The SPARQL query detailed in Listing 5 shows the list of resources that are inconsistent for the <scenario>. For example, Listing 6 shows a new version of the Goat Milking Scenario that has resources and episodes inconsistency because the resource horses is not mentioned in any episode. The query applied to the example will return a list with horses.

```
1. SELECT ?resources WHERE {
2. <scenario> hasResource ?resource}
3. MINUS{
4. <scenario> hasEpisode ?episode.
5. ?episode hasResource ?resource.}}
```

Listing 5. SPARQL query to detect inconsistency between resources and episodes.

Scenario: Goat Milking
Resources: goats, refrigerated tanks, milking machine, horses
Episodes:
The farmer sets the goat in the milking machine
The farmer extracts milk with the milking machine.
The farmer conducts the milk to a refrigerated tank.

Listing 6. A scenario with inconsistency with a resource

Query 3. Completeness with the satisfaction of contexts by goals

A scenario s can be performed if all its conditions described in the context attribute are contained in the union of the conditions described in the goal of other scenarios. Goals describe the intended situation (final states, postconditions) while contexts describe the starting point situations (initial states, preconditions). Thus, the context of a Scenario should be satisfied with the goals of other Scenarios, in order to be performed. The SPARQL query detailed in Listing 7 shows the list of conditions (contexts) for the <scenario> that are not satisfied by any other Scenario. For example, Listing 2 describes the Cheesemaking Scenario, where its context is satisfied by the goal of the Goat Milking Scenario described in Listing 1. Nevertheless, the context of the Goat Milking Scenario, is not satisfied with the goal of Cheesemaking Scenario.

```
1. SELECT ?contextCondition WHERE {
2. <scenario> hascontext ?contextCondition.}
3. MINUS{
4. ?otherScenario a Scenario.
5. ?otherScenario hasGoal ?contextCondition.}}
```

Listing 7. SPARQL query to detect context and goals completeness.

Query 4. Consistency in the sequence of the Scenarios

A scenario s can be performed if all its conditions described in the context attribute are contained in the union of the conditions described in the goal of the depending on scenarios. This query is a complement of the previous query that only checks if some goal can satisfy a context, while this query checks that a previous Scenario is the one that should satisfy the goal. The SPARQL query detailed in Listing 8 shows the list of conditions (contexts) for the <scenario> that are not satisfied by any depending on Scenario. For example, Figure 1 shows a dependency between Cheesemaking Scenario on Goat milking Scenario. This dependency is based on some stakeholders who stated that Goat milking should be done first and after that can be done Cheesemaking. Considering this dependency, this query tests if the Cheesemaking Scenario context is satisfied by the goal of the Goat Milking Scenario described.

```
1. SELECT ?contextCondition WHERE {
2. <scenario> hascontext ?contextCondition.}
3. MINUS{
4. ?otherScenario a Scenario.
6. <scenario> dependsOn ?otherScenario.
7. ?otherScenario hasGoal ?contextCondition.}}
```

Listing 8. SPARQL query to detect consistency in the sequence of the Scenarios.

Query 5. Completeness in the redundancy of goals

Some scenarios s1 and s2 have the same goal, thus, they should be refined in order to have different and specifics goals. When a group of stakeholders is collaboratively describing Scenarios, it is difficult that all of them have a complete understanding of the whole domain. Thus, when Scenarios with duplicated goals are identified it means that two overlapping scenarios are described. The SPARQL query detailed in Listing 9 shows the list of Scenarios that has a duplicated goal with the <scenario>. For example, Listing 10 and 11 shows a new version of the Goat milking and Cheesemaking Scenarios. This new version has the same goal because both scenarios are overlapped. The last episode of the Goat Milking Scenario overlaps Cheesemaking Scenario, and the first episode of the Cheesemaking Scenario overlaps with the GoatMilking Scenario.

```
1.SELECT ?scenarios ?goal WHERE {
2. ?scenario a Scenario.
3. <current> hasGoal ?goal.
4. ?scenario hasGoal ?goal.
5. FILTER(?scenario <> <current>).}
```

Listing 9. SPARQL query to detect redundancy of goals.

Scenario: Goat Milking
Goal: Obtain cheese from the goats
Episodes:
The farmer sets the goat in the milking machine
The farmer extracts milk with the milking machine.
The farmer conducts the milk to a refrigerated tank.
The cheesemaker producer makes cheese.

Listing 10. Goat milking Scenario overlapped with Cheesemaking Scenario

Scenario: Cheesemaking
Goal: Obtain cheese from the goats
Episodes:
The farmer do goat milking.
The cheesmaker curdles the milk with lactic ferments
The cheesmaker adds rennet to the milk
The cheesmaker drains the milk in mussels
The cheesmaker salts the milk
The cheesmaker leaves the milk to refine for 24 hours

Listing 11. Cheesemaking Scenario overlapped with Goat milking Scenario

5 Tool support

We developed a Media Wiki [9] based application to support the semantic representation of the Scenarios and the queries to identify issues. Media Wiki is an open source implementation written in PHP that uses the MySql database engine. Wikipedia and other projects of Wikimedia use Media Wiki. We have added two extensions: (i) an ad-hoc collaborative catalog and editor, and (ii) a semantic Media Wiki. Since it relies on the wikitext format, users with no knowledge of HTML or CSS can easily edit the pages and the result looks like web pages that users are familiar to. Media Wiki stores in a database all the different versions of each page, in a collaborative environment could be necessary to access a previous version. Another advantage of Media Wiki is the management of the links between pages. Although the destination of a link does not exist, the link can also be written and Media Wiki shows it anyway, when the user clicks the link, Media Wiki allow to create the page. It is a useful feature to connect Scenarios while they are being described. Figure 2 shows a screenshot of the Goat Milking Scenario with some report about an inconsistency detected with actors.

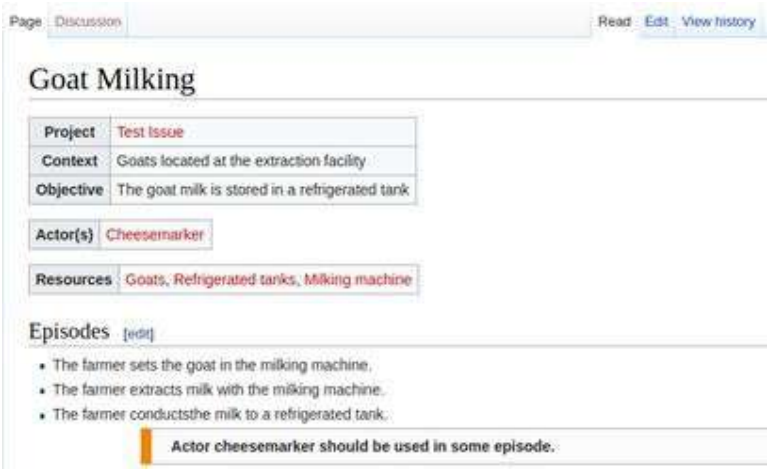


Figure 2. Goat milking Scenario without any reported issue

6 Conclusions

We have presented a semantic description of Scenarios and a set of semantic queries, both things implemented in a semantic Media Wiki in order to support the

collaborative description of Scenario. This proposal contributes to identify issues that arise because of the collaborative nature of the construction. Moreover, the agricultural domain is very specific because practices vary between different regions as well as their language. Thus, in order to communicate and interoperate different software systems, it is necessary to unify the knowledge of the different domains. We claim that our proposal provides an approach to capture the knowledge from different stakeholders and obtain a shared knowledge through an iterative and incremental process of checking and improving. This work is supported by the RUC APS project, in which three different groups of teams participate: IT experts, agricultural engineers and business specialist. We are using Scenarios and preliminary results are satisfactory. We plan to improve the scenarios verification developing more complex queries to check internal consistency between scenarios and we are also working in comparing scenarios with other sources of knowledge.

Acknowledgement

This research is supported by Agroknowledge and Ruc-Aps, a H2020 RISE-2015 project, aiming at Enhancing and implementing Knowledge based ICT solutions within high Risk and Uncertain Conditions for Agriculture Production Systems.

References

[1] Bhatia, M.P.S., Kumar, A., Beniwal, R,: Ontology based framework for detecting ambiguities in software requirements specification. In 3rd INDIACom, New Delhi. pp. 3572-3575. (2016)

[2] Carroll, J.M.: Five reasons for scenario-based design. In Interacting with computers 13.1. doi: 10.1016/S0953-5438(00)00023-0. pp 43-60. (2000)

[3] Dzung, D.V., Ohnishi, A.: Improvement of Quality of Software Requirements with Requirements Ontology. In 9th ICQS, Jeju, doi: 10.1109/QSIC.2009.44. pp. 284-289. (2009)

[4] Ge, C., Yu, S., Yang, G., Wang, W,: A collaborative requirements elicitation approach based on scenario. In 10th International Conference on Computer-Aided Industrial Design & Conceptual Design, Wenzhou. doi: 10.1109/CAIDCD.2009.5375171. pp. 2213-2216. (2009)

[5] Ilyas, M., Khan, S.U.: An empirical investigation of the software integration success factors in GSD environment. In 15th SERA, London, pp. 255-262. (2017)

[6] Kummler, P.S., Vernisse, L., Fromm, H.: How Good are My Requirements?: A New Perspective on the Quality Measurement of Textual Requirements. In 11th QUATIC, Coimbra. doi: 10.1109/QUATIC.2018.00031. pp. 156-159. (2018).

[7] Leite, J.C.S.dP., Rossi, G., Balaguer, F., Maiorana, V., Kaplan, G., Hadad, G., Oliveros, A.: Enhancing a requirements baseline with scenarios. In requirements Engineering. Vol 2.4. doi: 10.1109/ISRE.1997.566841. pp 184-198. (1997)

[8] McGuinness, D. L., Van Harmelen, F.: OWL web ontology language overview. W3C Recommendation, 10(10), 2004.

[9] Media Wiki, https://www.mediawiki.org

[10] Potts, C.: Using schematic scenarios to understand user needs. In 1st conference on Designing interactive systems: processes, practices, methods, & techniques. (1995)

[11] Ramasubbu, N., Kemerer, C.F.: Managing Technical Debt in Enterprise Software Packages. In IEEE Transactions on Software Engineering, 40-8. pp. 758-772. (2014).

[12] Sarmiento, E., Leite, J.C.S.d.P., Almentero, E.: Using correctness, consistency, and completeness patterns for automated scenarios verification. In 5th RePa. pp. 47-54. (2015)

[13] Shunxin, L., and S. Leijun, S.: Requirements Engineering Based on Domain Ontology. In 2010 International Conference of Information Science and Management Engineering, Xi'an. doi: 10.1109/ISME.2010.110. pp. 120-122. (2010)

[14] Tan, L., Haley, R., Wortman, R., Zhang, Q.: An extensible and integrated software architecture for data analysis and visualization in precision agriculture. In 13th IRI, Las Vegas, NV. doi: 10.1109/IRI.2012.6303020. pp. 271-278. (2012)

[15] Zait, F., Zarour, N.: Addressing Lexical and Semantic Ambiguity in Natural Language Requirements. In Fifth ISIICT, Amman. doi: 10.1109/ISIICT.2018.8613726. pp. 1-7. (2018)

Wiki Support for Software Use Cases

Abstract

The design of tests is a very important step in the software development process since it allows us to match the users' expectations with the finished product. Considered as a cumbersome activity, efforts have been made to automatize and alleviate the burden of test generation, but it is still a largely neglected step. We propose taking advantage of existing requirement artifacts, like Scenarios that describe the dynamic of the domain in a very early stage of software development, to obtain tests from them.

In particular, the approach proposed complement the Scenarios that are textually described with a glossary, the Language Extended Lexicon. Thus, a set of rules to derive tests from Scenarios is also proposed. The tests are then described using the Task/Method model.

The main findings of this work consist of an extension of a previously presented set of rules. And a tool based on a media wiki platform that makes possible to record Scenarios and the Language Extended Lexicon and implement the rules to obtain the tests.

The main originality of this work is the glossary which complements Scenarios, the semantic support to obtain tests and the tool to automatize the approach.

Introduction

Developing software still remains a very complex process involving several actors and consisting of different steps. The testing step remains as one of the biggest problems, and it is frequently avoided. As a consequence, the resulting system fail to meet users' expectations, rendering it useless. To tackle this problem, V-model (Forsberg et al., 2005) suggests to relate test with requirements in order to assure that the resulting software satisfy their needs. In particular, this proposal is interested in an early stage of requirements definition, because from them the ultimate test will arise. The objective is deriving tests in a semi-automatic way from requirements artifacts. A set of rules to perform the derivation and a tool to implement them are provided. Nevertheless, it is important to mention that it is a semi-automatic way, because tests derived should be adjusted since Scenario could be considered to be too abstract and the implemented functionality could be

more precise. Nevertheless, the approach provided is an automatic support to make easier the testing design step.

The approach combines three modelling techniques:

- Scenarios to describe behavior;
- the Language Extended Lexicon (LEL) a particular glossary, coming from the requirement engineering field as well as the Scenarios;
- the Task/Method model, coming from the Artificial Intelligence field, particularly from the knowledge-based systems.

The contribution of this paper is a set of rules that uses the Scenarios and the LEL as input to obtain tests as output. A tool to support the capture and the description of the Scenarios and the LEL is provided. It is also provided the rules for translating the scenarios in a Task/Method description. Thus, the tool obtains automatically the resulting tests.

The Scenarios are described with narrative text. The LEL is a good complement to describe and organize the vocabulary used. Thus, the benefit of using the LEL is double. It helps the requirements engineer to use a precise vocabulary reducing the ambiguity, but also structures the description in order to make simple the application of the rules.

This tool automatizes the application of the rules, but it also provides a semantic support. This semantic capability is not fully used, but it is planned to extend this work using it. This paper proposes an extension of a previous work in which the LEL was not used. Thus, according to several case studies and the feedback received with the previous publication the approach was enriched.

This work is applied to the RUC-APS project. RUC-APS is a H2020 RISE-2015 project, aiming at Enhancing and implementing Knowledge based ICT solutions within high Risk and Uncertain Conditions for Agriculture Production Systems. In this context the examples used are based on agricultural production.

The rest of the paper is organized as follows. First, related work is introduced. Then the background describes Scenarios, Language Extended Lexicon and the Task/method paradigm. In the third section, the rules to obtain tests from Scenarios are presented. In the fourth section, the tool as a proof of concept of the proposed approach is describe. Finally, some conclusions and future work are presented.

Related works

Test cases may be generated from requirements, designs and source code. In particular, the use of abstract artifacts like UML diagrams, helps defining User Acceptance Tests, and much research has been done in this direction. Two approaches can be distinguished in this area: those that consider the relationships between elements (inter-scenario dependency), and those that consider the variations within each element (intra-scenario) (Husain et al., 2015).

Inter-scenario dependency approaches provide a high-level organization of the artifact to cover different dependencies between them. Boucher et al. (Boucher and Mussbacher, 2015) transform workflow models (Use Case Maps) into Acceptance Test Cases that can be automated with the JUnit framework. Huaikou

et al. (Huaikou and Ling, 2000) analyze Z specifications, in particular, the prior and posterior state of every operation to generate test cases. Philip et al. (Philip et al., 2017) analyze a model with safety requirements to generate fault trees representing functional hazards. Then, test cases for validation of mitigation of hazards are generated automatically from the model. Nomura et al. (Nomura et al.m 2014) model the business context in a matrix representing the dependency between business process, from which they design test scenarios from the perspective of Personas to cover the different situations. Sarmiento et al. (Sarmiento and Leite, 2016) propose a similar approach using scenarios. The work of Nogueira et al. (Nogueira et al., 2012) propose generating test cases from specifically formatted use cases templates that capture key requirements' features as control flow, input and output. Using a similar approach, but using annotated UML Activities Diagrams, Vieira et al. also are able to automatically generate tests with the aim of maximizing coverage (Vieira et al., 2006).

Intra-scenario approaches focus on the detail of some product (a piece of requirement or a design artifact) and analyze the steps or elements of each artifact to design tests. Pandit et al. (Pandit et al., 2016) automatically design UATs based on acceptance criteria written in the form of Given-When-Then template. These criteria are divided in steps, and dependencies amongst steps are arranged in a dependency graph. Lei et al. (Lei and Wang., 2016) propose a framework to analyze testing constraints in requirements as the basis of test scenarios. Lipka et al. (Lipka et al., 2015) derive test scenarios from use cases stated in natural language, enriched with annotations to connect the specification with the source code of the application.

Background

This section gives a brief introduction to Scenarios and the Language extended Lexicon, which are the input to the proposed approach to derive Tests. This section also describes the Task Method / Model, the technique used to describe the tests.

Scenarios and Language Extended Lexicon

Scenarios describe interactions between users and a future system (Potts, 1995). It is also used to understand the context of the application because they promote the communication when there is a great variety of experts (Carroll, 2000). Leite (Leite et al., 1997) defines a Scenario with the following attributes: (i) a title that identifies the Scenario; (ii) a goal or aim to be reached through the execution of the episodes; (iii) a context that sets the starting point to reach the goal; (iv) the resources, relevant physical objects or information that must be available, (v) the actors, agents that perform the actions, and (vi) the set of episodes.

The following Scenario describes the activity of sorting out the products according to their quality in order to send them to different customers. It is important to mention that Scenario describes a task (summarized in the goal) that is divided in smaller tasks (described in the episodes). Every one of these smaller tasks (i.e. every episode) can be performed successful or not. These are the kind of Scenarios that our approach work with. Furthermore, it is also important to remark that Scenario describes some dynamic aspects of the domain but not specific requirements of a software application. Thus, this particular example describes tasks that performs a farmer and can be automatized with information and communication technology.

The Language Extended Lexicon (LEL) is a glossary used to capture and describe the domain´s language (Leite and Franco, 1993). The LEL is composed of terms (also called symbols) that are classified into four types:

Subject, Object, Verb, and State. Subjects represent active element that performs actions. Objects are passive elements on which subjects perform actions. A verb is used to represent the actions. Finally, States represent situations in which subjects and objects can be located.

Scenario: Sorting out the product
Goal: Determining the destination of a product according to its quality level
Context:
The farmer have orders from different clients.
The clients have different requirements about quality levels of the products.
The product have been recently harvested and they are dirty.
Resources: Product, orders, quality levels.
Actors: Farmer
Episodes:
The farmer washes the product
The farmer brushes the product
The farmer ranks the quality level
The farmer chooses the destination
The farmer packs the product

Listing 1. Scenario about Sorting out product

A symbol is described by two attributes: (i) the notion and (ii) the behavioral responses. Notion describes the symbol denotation and explains its literal meaning. While Behavioral responses describe its connotation, that is, the effects and consequences of the relationship between the defined symbol and others symbols defined in the LEL (Sarmiento et al., 2014). These two attributes are very important to describe the symbols, because they make a complete descripcion, while the notion describe the symbol from an internal point of view, the behavioural responses make a description from the exterior of the symbol. Nevertheless, for the description of this approach, we will only use the categorization of the symbols. That is why we will not provide examples of fully described symbols. Moreover, for the proposed approach, we only deal with subject, object and verbs. But we do not use states. Listing 2 enumerates the symbols used in the Scenario and their category.

Subject: Farmer
Object: Product
Object: Orders
Object: Destination
Object: Quality level
Verb: Wash

Verb: Brush
Verb: Rank
Verb: Choose
Verb: Pack

Listing 2. Terms used in the Scenario about sorting out the products

There is a relationship between the attributes of the Scenario and the LEL. In general symbols of subject category of the LEL are used as actors in Scenarios, and objects in the LEL are resources in Scenarios. Then, verbs in the LEL are used to describe the name of the Scenario, in particular, it is commonly to use expressions that contains verbs followed by an object. Then, the actions described in each episode also have some similar structure. The only difference is that each episode begins with a subject that perform the action: <subject of the LEL> + <verb of the LEL> + <object of the LEL>. Thus, the mapping between LEL glossary and Scenario should be the following:

Scenario: <Verb of the LEL> + <Object of the LEL>
Goal: …
Context: …
Resources: <Object of the LEL>
Actors: <Subject of the LEL>
Episodes:
<Subject of the LEL> + <Verb of the LEL> + <Object of the LEL>
<Subject of the LEL> + <Verb of the LEL> + <Object of the LEL>

Listing 3. Mapping between Scenario and LEL

The definition of Scenarios and the glossary LEL should be done iteratively and both elements (LEL and Scenarios) should be described at the same time. That is, the experts can describe some terms in the LEL and then use these terms to describe Scenarios. And he can also describe some Scenarios and while describing it he realize that some terms used in the Scenario are very important and should be described in the glossary.

Tests description using the Task Method / Model paradigm

The task/method paradigm is a knowledge modelling paradigm seeing the reasoning as a task. This paradigm mainly comes from the artificial intelligence research domain (more precisely from knowledge acquisition and modelling, expert systems, knowledge based systems fields) (Trichet and Tchounikine, 1999). The essential advantage of this paradigm is to have a declarative form to express knowledge which can be easily processed by tools such as execution engines (for performing tasks), planners (for defining new ways to achieve tasks), etc (Camilleri et al., 2003).

A task/method model is composed of two submodels: the domain model and the reasoning model. The domain model describes the objects of the world used (directly or indirectly) by the reasoning model. This model

can be viewed as an application ontology, and is often described with the UML language and implemented with an object oriented language. The reasoning model describes how a task can be performed. It uses two modelling primitives:

1. Task: a task is a transition between two world state families (an action) and is defined as follow:
   - *Name*: Task name
   - *Par*: Typified list of parameters handled by the task
   - *Objective*: Goal state of the task
   - *Methods*: Method list achieving the task
2. Method: a method describes one way of performing a task. A method is characterized by the following fields:
   - *Heading*: Task achieved by the method
   - *Prec*: Preconditions which must be satisfied to be able to apply the method
   - *Effects*: Effects generated by the successful application of the method
   - *Control*: achievement order of the subtasks
   - *Subtask* : subtask set

Here, we will focus only on fields/concepts used in this work, a complete presentation of this modelling paradigm can be found in (Camilleri et al., 2003). The task's field *Name* specifies the name of the task. The field *Par* contains the list of parameters, that is all objects handled by the task.

For example, the Task / Method Model for the same Scenario should be the one described in Listing 4.

Method: M1
Task : Sorting_Out (product)
Control:
wash (farmer, product);
brush (farmer, product);
rank (farmer, quality_levels);
choose (farmer, destination);
pack (farmer, products);

Method: M11
Task: bush (farmer, product)
precondition:
not Product_correctly_washed
Control:
message("not correctly washed, stop");
stop;

Method: M12
Task: brush (farmer, product)
Precondition:

not Product_correctly_brushed

Control:

message(“not correctly brushed, stop”);

stop;

and so on…

Listing 4. Task / Method model for the Scenario of sorting out products

As it was stated in the introduction, our approach deals with Scenarios that consider situation true / false, where an action can be performed correctly or not. Thus, after every episode, the result of the task should be analyzed. If the result is successful the execution of the Scenario continue, but if the result of the execution of the Scenario fail, the Scenario ends. Thus, the situation considered for the Task / Model method of listing 4 are described in listing 5.

- Product correctly washed /not correctly washed
  - If Product correctly washed then brush / if not correctly washed then stop
- Product correctly / not correctly brushed
  - If Product correctly brushed then rank the quality level / if not correctly brushed then stop
- Product with rank of the quality level known / unknown
  - If Product has a known rank of the quality level then choose the destination / If has not a known rank of the quality level then stop
- Product with a destination known / unknown
  - If Product has a destination known then pack the product / If Product has not a destination known then stop
- Product packed / not packed
  - If Product was packed then finished / If Product was not packed then stop

Listing 5. Tests flow for the Scenario about sort out products

Rules to derive tests

This section enumerates the rules proposed to derive tests from Scenarios. Some preliminary version of the rules were presented in a previous work (self reference removed). Although the rules could look similar to the previous ones, the Scenarios they relies on a more structured template to describe them.

Rule 1. Tasks Identification

Each verb in the Scenario’s episodes is translated into a task in Task/Method model. Each Scenario title is also a task in Task/Method model. Examples:

Scenario: Sorting out the product → LEL mapping: <*sort out* verb> → Task: Sort out
Episode: The farmer washes the product → LEL mapping: <*wash* verb> → Task: wash
Episode: The farmer brushes the product → LEL mapping: <*brush* verb> → Task: brush
Episode: The farmer ranks the quality level → LEL mapping: <*rank* verb> → Task: rank
Episode: The farmer chooses the destination → LEL mapping: <*choose* verb> → Task: choose
Episode: The farmer packs the product → LEL mapping: <*pack* verb> → Task: pack

Listing 6. Task identification

Rule 2. Task's Parameters Identification

Each subject and object used in the episodes of a Scenario is translated into a parameter in Task / Method model. Examples:

Episode: The farmer washes the product → LEL mapping: <*farmer* term>, <*product* object> →Task: wash (farmer, product)
Episode: The farmer brushes the product → LEL mapping: <*farmer* term>, <*product* object> → Task:brush (farmer, product)
Episode: The farmer ranks the quality level → LEL mapping: <*farmer* term>, <*quality_level* object> → Task:rank (farmer, quality_level)
Episode: The farmer chooses the destination → LEL mapping: <*farmer* term>, <*destination* object> → Task: choose (farmer, destination)
Episode: The farmer packs the product → LEL mapping: <*farmer* term>, <*product* object> → Task: pack (farmer, product)

Listing 7. Parameters identification

Rule 3. Episode's method

The achievement of Scenario's episodes is associated to a method in Task / Method model. Examples:

Achievement of episode: The farmer washes the product → Method: M11
Achievement of episode: The farmer brushes the product → Method: M12
Achievement of episode: The farmer ranks the quality level → Method: M13
Achievement of episode: The farmer chooses the destination → Method: M14
Achievement of episode: The farmer packs the product → Method: M15

Listing 8. Method identification

Rule 4. Sequence of tasks

The sequence of different lines in the episodes part of a Scenario determines the sequence of tasks in the control part of a method in the Task / Method model. The use of expressions like "then", "after", etc... in the episodes of a Scenario determines also a sequence of tasks in the method's control part. Examples:

Episodes: The farmer washes the product The farmer brushes the product The farmer ranks the quality level The farmer chooses the destination The farmer packs the product	→	Method: M1 Control Task: wash (farmer, product) Task:brush (farmer, product) Task:rank (farmer, quality_level) Task: choose (farmer, destination) Task: pack (farmer, product):

Listing 9. Sequence of tasks

Rule 5. Test Case Method

It is assumed in this work that each test case (Test cases part of scenario) corresponds to a binary achievement status: success or failure. If the achievement is succeeded, the execution of the test continue with the following episode. In a failure situation, the scenario will stop. This stop case will be represented by a method for the next task in which the precondition field correspond to the test case failure. For example:

```
Test case: Product correctly wahed / not correctly washed →
Method:  M12
Task:  brush(farmer,product) # next task
precondition: not Product_correctly_washed
Control:    message("not correctly washed, stop");
             Stop;
```

Listing 10. Test case method

Prototype

It has been developed an application to support the proposed approach of deriving tests from Scenarios. The application allows the collaboratively description of Scenarios and the LEL, and it also implements the rules to obtain tests described in Task / Method model. A Media Wiki (MediaWiki, 2018) platform is used as a repository of the Scenarios and the LEL, as well as a tool to derive the Tests. Many extensions have been added to Media Wiki. An ad-hoc collaborative catalog and editor was added to manage Scenarios and LEL. This tool is

enriched with a semantic Media Wiki extension that allow the semantic support and the creation of forms to make the CRUD operations (create, retrieve, update and deleted) in a more user-friendly way. Finally, the semantic extension also provides a query language for semantic searches in order to implement the derivation of tests.

Wikis are tools that allows users with no knowledge in programming to edit easily information on the web. This feature foster the participation and collaboration of non technical users. In traditionals wikis there are three different elements to build each page: (i) the code that can be edited by every user and it is stored in the server; (ii) a template that defines the elements and their position; and (iii) the HTML code provided in real time by the server with the page requested by the user, it is a combination of (i) and (ii).

There are many software systems to implement wikis. Media Wiki is an open source implementation in PHP that uses MySql database engine. Wikipedia and other projects of Wikimedia use Media Wiki. Media Wiki relies on the wikitext format. Thus, users with no knowledge of HTML or CSS can easily edit the pages. Another advantage of Media Wiki is the management of the links between pages. Although the destination of a link does not exist, the the link can also be written and Media Wiki shows it anyway, when the user clicks the link, Media Wiki allow to create the page. Another important feature is the configuration management. Media Wiki stores in a database all the different versions of each page, so, all previous versions can be accessed is necessary.

Media Wiki has been extended with the Semantic Media Wiki framework that allow to retrieve information from the Wiki in a similar way that data is retrieved from a database. In a Semantic Media Wiki each piece of data is represented by a triplets of: subject, predicate and object. Every element of this triplet corresponds to a category of the LEL glossary. And the triplet is also related to the template used in the description of the Scenarios. Some attributes as Title, actors and resources uses some the categorias, but the episodes are described with the triplet. Thus, Semantic Media Wiki framework provides the infrastructure to organize and store the information and a query language to implement the derivation rules in a straightforward way. Another technology added to Media Wiki is Page Forms, that is also called Semantic Forms. This extension allows to add, edit and retrieve data using customized forms. It is based on templates that are used in Pages called Special Pages.

Input types is another extension used. It allows to define the basic elements and its attributes. Nevertheless, the episodes of the Scenarios are complex structures (they are a set of triplets), thus another extension was needed to deal with them. The Semantic Internal Object extension was used. Figure 1 shows a conceptual model that describes the types of Scenarios and LEL. Every LEL symbols has two attributes: notion and impact (behavioral responses). There are 4 categories of LEL Symbol, but only use 3 were used in this approach: Subject, Object and Verbs. The Scenarios is the core of the approach. It has a title, composed by a verb and an object. It also has an objective and a context, both are regular text. Then, actors and resources are important, because actors are mapped to subjects (subject category of the LEL and subjects in the semantic triplets), and resources are mapped to objects (object in the LEL and the triplets). The episodes are a set of expressions, in fact, they are full sentences with a subject and a predicate that is represented by a verb and an object. This full sentence is composed by symbols of the three categories, but they also are the semantic triplets.

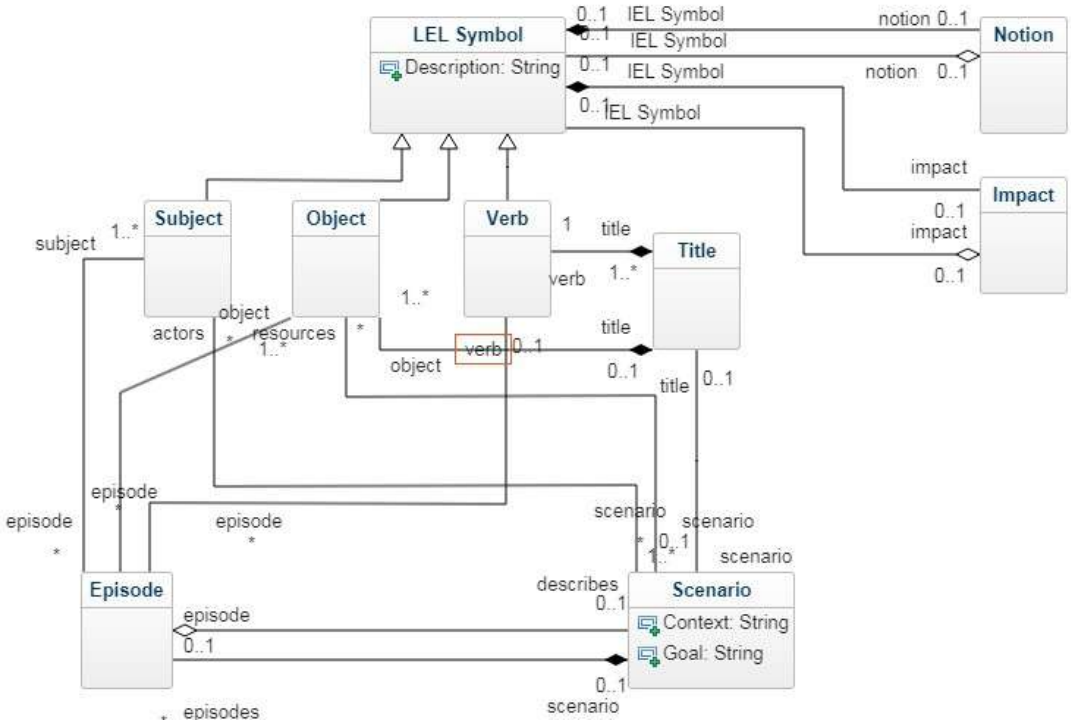


Figure 1. Scenario and LEL model

In order to define the entities and provide the functionality to edit them, it is necessary to define semantic properties, the templates, the forms and the categories. The semantic properties are the essential element in a semantic website because they determine the links between the information. The templates define how the data is shown in a page. The template translate the semantic information in editable data. Forms are used to add and edit data through one or more templates. Finally, categories determines the elements that the Media Wiki manages. Table 1 summarizes the configuration of categories, forms, templates and properties made to manage Scenarios and LEL.

Table 1. Configuration summary

Categories	**Forms**	**Templates**		**Properties**
Scenarios	Scenario	Scenarios		Its context is
				The objective is
		Episode	EpisodesData	EpisodeObject
				Who participates
				What does the participant do
				What does the participant uses

			ResourcesScenario		The resources are
			ActorsScenario		The actors are
LELSymbol	Subjects	Subject	Subject		ImpactProperty
					NotionProperty
	Objects	Object	Object		ImpactProperty
					NotionProperty
	Verbs	Verb	Verb		ImpactProperty
					NotionProperty

Figure 2 shows the form to visualize and navigate the information of an Scenario. The screenshot shows the same Scenario described in Listing 1. The title of the Scenario is the title of the page. Then, the Scenario has information about its context and its objective that is textual information (String). The actors and resources are LEL symbols, thus they are links because that words can be clicked to navigate to that information. Then, the episodes are represented by triplets, where each element of the triplets is a reference to a LEL. So, it can also be navigated. The figure also shows the link to edit the information. And there is also a link to view the history of the different modifications performed.

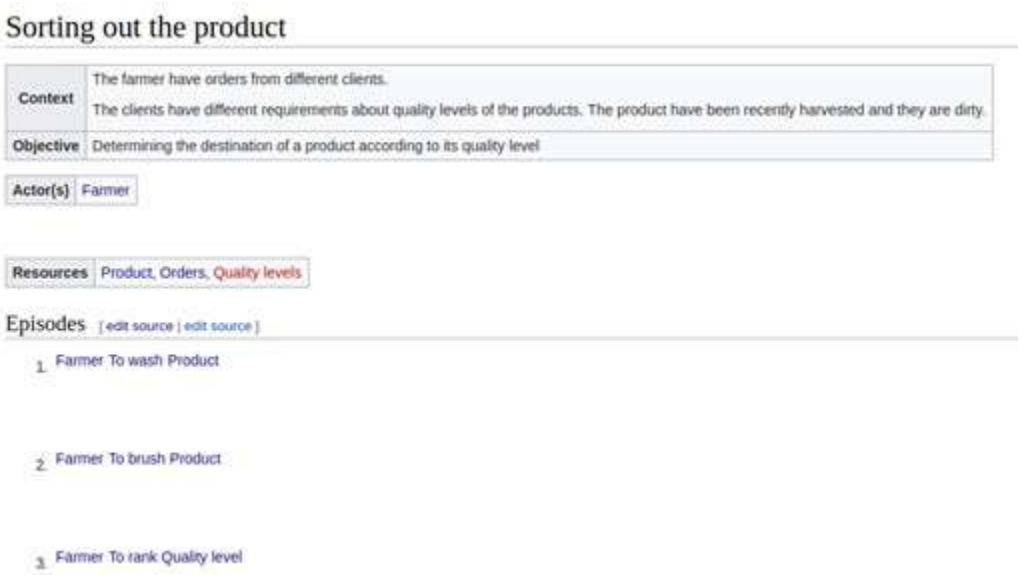


Figure 2. Scenario navigation form

Figures 3 and 4 show the forms to edit the same Scenario. Figure 3 shows the attributes context and objective, where the information is plain text. Then, actors and resources are LEL elements, that is why they are entered as text, but they are converted to tokens. This translation is done automatically during the edition. Figures 4 shows the forms to edit the episodes. Since episodes has a specific structure (triplets of subject, predicate and object), the form displays a template to enter an actor (the subject), a verb (the predicate) and the resource (the object). The Scenario may have many episodes, so the form has a button to replicate this template.

Figure 3. Scenario edition form

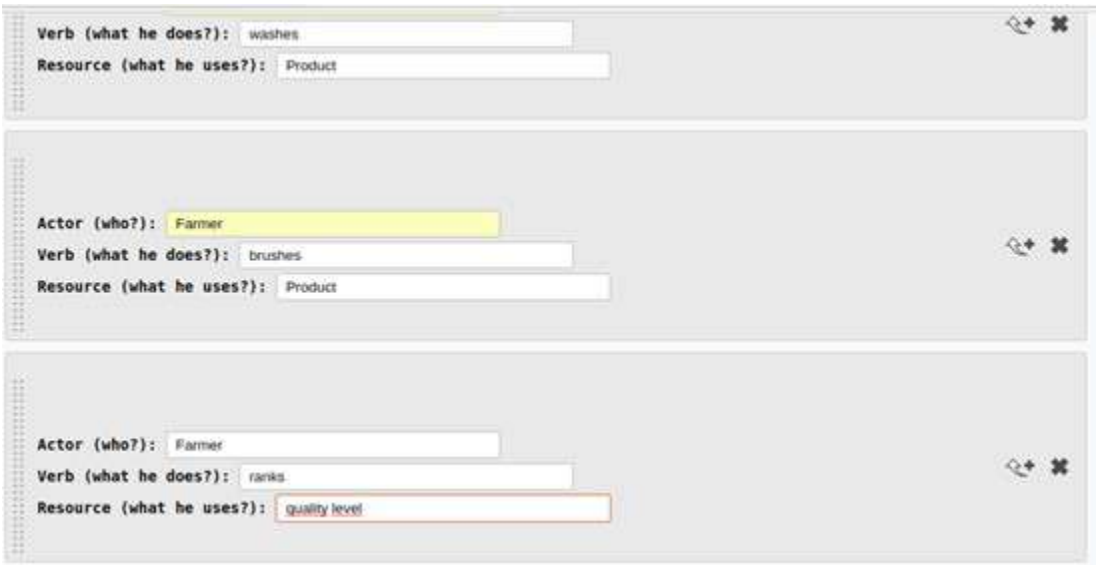


Figure 4. Scenario episodes edition form

Figures 5 shows a form with all the LEL symbols defined in the catalog. This page is an index automatically calculated. The list displayed is a query similar to the needed to apply the rules to derive de Tests. The page is configured to retrieve all the elements that belong to LELSymbol category, in particular objects.

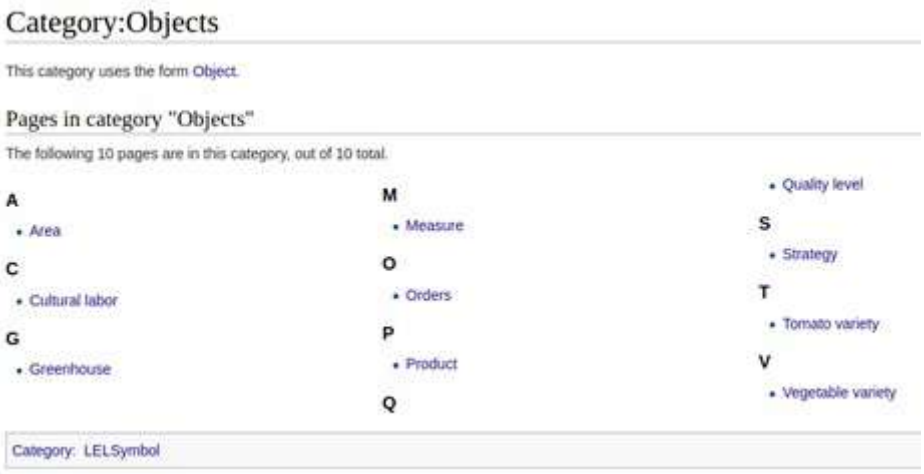


Figure 5. LEL symbols list of elements

Finally, the resulting test from the previous Scenario is displayed in Figure 6. An example of source code to make queries is listed in Listing 11. Part of the source code is wiki language and there are also semantic queries. The queries are the blocks that begin with #ask. In the blocks the categories described in table 1 are used to find specific elements of the Scenario.

Method: M1

Task: Sorting_Out (product)

Control:
wash (farmer, product);
brush (farmer, product);
rank (farmer, quality_levels);
choose (farmer, destination);
pack (farmer, products);

Method: M11

Task: bush (farmer, product)
precondition:
not Product_correctly_washed

Control:
message("not correctly washed, stop");
stop;

Method: M12

Task: brush (farmer, product)
Precondition:
not Product_correctly_brushed

Control:
message("not correctly brushed, stop");
stop;

and so on…

Figure 6. Task Method / Model test visualization form

```
= Method:  M11 =
<b>Task:</b>
       [[ {{#ask:
       [[Category:Scenarios]]
       [[EpisodesData in::Episode]]
       |?WhatDoesTheParticipantDo
       }} ]]
       ([[
       {{#ask:
       [[Category:Scenarios]]
       [[EpisodesData in::Episode]]
       |?WhoParticipate
       }}]],
       [[
       {{#ask:
       [[Category:Scenarios]]
       [[EpisodesData in::Episode]]
       |?WhatDoesTheParticipantUses
       }} ]]) <br>
<b>precondition:</b> <br> not Product_correctly_washed <br>
<b>Control:</b> <br>  message("not correctly washed, stop");<br>
```

```
stop;<br>
```

Listing 11. Source code for Method definition

Conclusions

In this paper we have shown a new way of generating tests from well-known requirements artifacts, by presenting a set of rules implemented in a wiki based platform. We are now working in completing the set of rules, by adding rules required to translate more complex decision (switch or case situations instead of if then) as well as iterative behaviour (loops or iterations like for each). We also plan to take benefit of the semantic support in order to improve the rules and not only the syntactic analysis. This work will certainly benefit the software development in the last step of the global process, i.e. the tests step. Nevertheless, in order to show how this work is useful and as perspective of this work we intend to apply the developed tool and process to real cases.

Acknowledgements

Authors of this publication acknowledge the contribution of the Project 691249, RUC-APS: Enhancing and implementing Knowledge based ICT solutions within high Risk and Uncertain Conditions for Agriculture Production Systems (www.ruc-aps.eu), funded by the European Union under their funding scheme H2020-MSCA-RISE-2015

References

(Boucher and Mussbacher, 2015) Boucher M., Mussbacher G. (2017) "Transforming Workflow Models into Automated End-to-End Acceptance Test Cases," in *Proceedings - 2017 IEEE/ACM 9th International Workshop on Modelling in Software Engineering, MiSE 2017*, 2017, pp. 68–74.

(Camilleri et al., 2003) Camilleri G., Soubie J.L., Zalaket J. (2003) "TMMT: Tool Supporting Knowledge Modelling," in *Knowledge-Based Intelligent Information and Engineering Systems*, vol. 2773, pp. 45–52.

(Carroll, 2000) Carroll, J.M. (2000) Five reasons for scenario-based design. Interacting with computers 13.1. doi: 10.1016/S0953-5438(00)00023-0. pp 43-60.

(Forsberg et al., 2005) Forsberg K., Mooz H., Cotterman H. (2005) *Visualizing Project Management,* 3rd edition, John Wiley and Sons, New York, NY.

(Huaikou and Ling, 2000) Huaikou M., Ling L. (2000) "A test class framework for generating test cases from Z specifications," in *Sixth IEEE International Conference on Engineering of Complex Computer Systems, ICECCS 2000*, pp. 164–171.

(Husain et al., 2015) Hussain A. *et al.*, (2015) "Review on formalizing use cases and scenarios: Scenario based testing," *2015 Int. Conf. Emerg. Technol.*, vol. 3, no. 3, p. 1.

(Lei and Wang, 2016) Lei H., Wang Y. (2016) “A model-driven testing framework based on requirement for embedded software,” in *11th International Conference on Reliability, Maintainability and Safety (ICRMS)*, pp. 1–6.

(Leite and Franco, 1993) Leite J.C.S.dP., Franco A.P.M. (1993) “A strategy for conceptual model acquisition”, In Requirements Engineering conference. IEEE. doi:10.1109/ISRE.1993.324851, pp 243–246.

(Leite et al., 1997) Leite J.C.S.dP., Rossi G., Balaguer F., Maiorana V., Kaplan G., Hadad G., Oliveros A. (1997) “Enhancing a requirements baseline with scenarios”, In requirements Engineering. Vol 2.4. doi: 10.1109/ISRE.1997.566841. pp 184-198.

(Lipka et al., 2015) Lipka R., Potuak T., Brada P., Hnetynka P., Vinarek J., (2015) “A Method for Semi-Automated Generation of Test Scenarios Based on Use Cases,” in *Proceedings - 41st Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2015*, pp. 241–244.

(MediaWiki, 2018) Media Wiki (2018) available at: https://www.mediawiki.org (accessed September 1st, 2018)

(Nomura et al.m 2014) Nomura N., Kikushima Y., Aoyama M. (2014) “A Test Scenario Design Methodology Based on Business Context Modeling and Its Evaluation,” *21st Asia-Pacific Softw. Eng. Conf.*, vol. 1, pp. 3–10.

(Nogueira et al. 2012) Nogueira, S., Sampaio, A., & Mota, A. (2014). Test generation from state based use case models. Formal Aspects of Computing, 26(3), 441-490.

(Pandit et al., 2016) Pandit P., Tahiliani S., Sharma M. (2016) “Distributed agile: Component-based user acceptance testing,” in *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*, pp. 1–9.

(Philip et al., 2017) Philip G., Dsouza M., Abidha V. P. (2017) “Model based safety analysis: Automatic generation of safety validation test cases,” in *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*, pp. 1–10.

(Potts, 1995) Potts C. (1995) “Using schematic scenarios to understand user needs”. In proceedings of the 1st conference on Designing interactive systems: processes, practices, methods, & techniques. ACM. doi: 10.1145/225434.225462.

(Sarmiento et al., 2014) Sarmiento E., Leite J.C.S.dP., Almentero E. (2014) “C&L: Generating model based test cases from natural language requirements descriptions”. In Requirements Engineering and Testing (RET), 2014 IEEE 1st International Workshop on. IEEE, 2014. doi: 10.1109/RET.2014.6908677

(Sarmiento and Leite, 2016) Sarmiento E., Leite J.C.S.dP., Almentero E., Sotomayor G. (2016) “Test Scenario Generation from Natural Language Requirements Descriptions based on Petri-Nets,” *Electron. Notes Theor. Comput. Sci.*, vol. 329, pp. 123–148.

(Trichet and Tchounikine, 1999) Trichet F., Tchounikine P. (1999) “DSTM: A framework to operationalise and refine a problem solving method modeled in terms of tasks and methods,” *Expert Syst. Appl.*, vol. 16, no. 2, pp. 105–120.

(Vieira et al,. 2006) Vieira, M., Leduc, J., Hasling, B., Subramanyan, R., & Kazmeier, J. (2006, May). Automation of GUI testing using a model-driven approach. In Proceedings of the 2006 international workshop on Automation of software test (pp. 9-14). ACM.

ICDSST – PROMETHEE DAYS 2018 PROCEEDINGS – THE EWG-DSS 2018 INTERNATIONAL CONFERENCE ON DECISION SUPPORT SYSTEM TECHNOLOGY & PROMETHEE DAYS 2018:
Jason Papathanasiou, ???
(editors)
Heraklion, Greece, 22-25 May 2018

A Modelling Approach to Generating User Acceptance Tests

Leandro Antonelli[1], Guy Camilleri[2], Julián Grigera[1,3], Mariángeles Hozikian[1], Cécile Sauvage[4] and Pascale Zarate[5]

[1]LIFIA, Facultad de Informática, UNLP, Argentina
[2]SMAC group, IRIT, 118 route de Narbonne, 31062 Toulouse Cedex 9, France
[3]Also at CIC, Buenos Aires, Argentina
[4]FEDACOVA, C/ Isabel la Católica 6. Pta9-10, 46004 Valencia, Spain
[5]ADRIA group, IRIT, Université de Toulouse, 2 rue du Doyen Gabriel Marty, 31042 Toulouse Cedex 9, France
{camiller,zarate}@irit.fr, {leandro.antonelli, julian.grigera, marian.hozikian}@lifia.info.unlp.edu.ar, internacional@fedacova.org

ABSTRACT

Software testing, in particular acceptance testing, is a very important step in the development process of any application since it represents a way of matching the users' expectations with the finished product´s capabilities. Typically considered as a cumbersome activity, many efforts have been made to alleviate the burden of writing tests by, for instance, trying to generate them automatically. However, testing still remains a largely neglected step.

In this paper we propose taking advantage of existing requirement artifacts to semi-automatically generate acceptance tests. In particular, we use Scenarios, a requirement artifact used to describe business processes and requirements, and Task/Method models, a modelling approach taken from the Artificial Intelligence field. In order to generate acceptance tests, we propose a set of rules that allow transforming Scenarios (typically expressed in natural language), into Task/Methods that can in turn be used to generate the tests.

Using the proposed ideas, we show how the semi-automated generation of acceptance tests can be implemented by describing an ongoing development of a proof of concept web application designed to support the full process.

Keywords: User Acceptance Tests, Scenarios, Task/Method model, Agriculture Production Systems

INTRODUCTION

Developing software still remains a very complex process involving several actors and consisting of different steps. The testing step remains as one of the biggest problems, and it

is frequently avoided. As a consequence, the resulting system can fail to meet users' expectations, rendering it useless. Our objective is to develop a strategy to make the testing step easier, generating User Acceptance Tests (UATs) in a semi-automatic way from requirements artifacts. To do this, we combine two modelling approaches: Scenarios, from the requirement engineering field and Task/Method models, from the Artificial Intelligence field, particularly knowledge-based systems [1]. We provide rules to automatically translate scenarios to task/method models from which UATs can be generated.

This work is applied to the RUC-APS project. RUC-APS is a H2020 RISE-2015 project, aiming at Enhancing and implementing Knowledge based ICT solutions within high Risk and Uncertain Conditions for Agriculture Production Systems. In this context we will use a scenario based on agriculture production. The rest of the paper is organized as follows: we first introduce related work, then present the background introducing scenarios and the Task/method paradigm. In the third part we define our approach and in the fourth part we demonstrate a first proof of concept. Finally, we show our conclusions and future work.

RELATED WORK

Test cases may be generated from requirements, designs and source code. In particular, the use of abstract artifacts like UML diagrams, helps defining User Acceptance Tests, and much research has been done in this direction. Two approaches can be distinguished in this area: those that consider the relationships between elements (inter-scenario dependency), and those that consider the variations within each element (intra-scenario).

Inter-scenario dependency approaches provide a high-level organization of the artifact to cover different dependencies between them. Boucher et al. [2] transform workflow models (Use Case Maps) into Acceptance Test Cases that can be automated with the JUnit framework. Nomura et al. [3] model the business context in a matrix representing the dependency between business process, from which they design test scenarios from the perspective of Personas to cover the different situations. Sarmiento et al. [4] propose a similar approach using scenarios.

Intra-scenario approaches focus on the detail of some artifact and analyze its steps or elements to design tests. Pandit et al. [5] automatically design UATs from acceptance criteria written in the Given-When-Then template. These criteria are divided in steps, and dependencies amongst steps are arranged in a dependency graph. Lipka et al. [6] derive test scenarios from use cases stated in natural language, enriched with annotations to connect the specification with the source code of the application.

BACKGROUND

Scenarios

Scenarios can be used in different stages of software development, from clarifying business process and describing requirements, to providing the basis of acceptance tests [7]. There is a distinction between application domain (the real world) and the application software (the machine)[8]: during business process modelling and requirements elicitation, Scenarios describe events in the world, while in system specification, they describe events in the machine. Scenarios are stories about people and the activities they perform to reach certain goals, parting from a setting and counting with some resources. Their description ranges from visual

(storyboards) to narrative (structured text) [9]. Leite et al. [10] propose a template with six attributes to describe Scenarios in a textual way:

- **Title**, it is the name of the scenario to identify it.
- **Goal, conditions and restrictions** to be reached after the execution of the Scenario.
- **Context, conditions and restrictions** that are satisfied and constitute the starting point of the Scenario execution.
- **Actors** and **agents** that perform actions during the Scenario to traverse the path from the context to reach the goal.
- **Resources, products** and **elements** used by the actors to perform action.
- **Episodes**: steps executed by the actors using the resources beginning at the context to reach the goal.

The text descriptions in Scenarios follow a fixed structure. In particular, episodes must be written with full sentences describing the subject, the action they perform, and if necessary the resource used. The following example describes a Scenario for farmer packing products. The example also includes the cases to consider for testing the scenario. These test cases do not belong to the original structure of the scenario:

Scenario: Packing the products
Goal: Put the products in boxes in order to distribute them
Context: The products have recently been harvested
Resources: Products, Box
Actors: Farmer
Episodes:
The farmer washes the products
The farmer brushes the products
The farmer determines the destination of the products
The farmer determines the quality levels of the products according to the destination
The farmer determines the appropriate box according to the destination
The farmer chooses the products that satisfy the quality levels
The farmer packs the chosen products in the box
Test cases:
Temperature forecast obtained / not obtained
Sun radiation forecast obtained / not obtained
Rain forecast obtained / not obtained
There is no best date to plant

Task/Method Paradigm

The task/method paradigm is a knowledge modelling paradigm (mainly from the artificial intelligence field [11], [12]) that sees reasoning as a task. Knowledge is expressed in a declarative way, making it easy to process by execution engines or planners [1]. A task/method model is composed by a **domain model** and a **reasoning model**. The former describes the objects of the world being used (directly or indirectly) by the latter, similarly to an application ontology. It is often described in UML language and implemented with OO languages. The reasoning model describes how a task can be performed. It uses two modelling primitives:

1. **Task**: it is a transition between two world state families (an action) and is defined by the following fields: *Name, Par, Objective* and *Methods*.
2. **Method**: it describes one way of performing a task. A method is characterized by the following fields: *Heading, Prec, Effects, Control* and *Subtask.*

The task's field *Name* specifies the name of the task. The field *Par* contains the list of parameters, that is, all objects handled by the task. For example, in a task *Pack*, the parameter list could be *(farmer, products)* which are domain objects (from domain model) used by the

task *Pack*. We will write *Pack(farmer, products)*. The list of methods which can be applied to perform a task is described in the field *Methods*. A terminal task is a directly executable task. The method's field *Prec* contains conditions that must be satisfied to apply the method. The execution order of subtasks is described in the *Control* field, and sub-tasks are recorded in the *Subtask* field. Note that, by essence, Task/Method models are hierarchical. Here we explained only the fields used in this work, see [1] for a full reference.

APPROACH

The proposed approach consists in representing scenarios in the form of Task/Method models. Being a modelling paradigm, the building of Task/Method models requires modelling effort. On one hand, the integration of this modelling activity during the definition of scenarios facilitates an early identification of misunderstandings between stakeholders. Moreover, as Task/Method models are operational models, they can be executed to generate test cases. On the other hand, building a task/method model at early stages shouldn't take much effort. To reduce this effort, we propose a semi-automatic translation of scenarios to task/method models. We use scenarios expressed in natural language, since it's the natural way to describe them.

In the packing example presented previously, the translation process would produce the following Task/Method model for the general scenario and the first subscenario, respectively:

Method: M1
Task: Packing(products)
Control:

wash(farmer, products);

brush(farmer, products);

determines(farmer, destination, products);

determines(farmer,quality_levels,products,destination);

determine(farmer, appropriate_box, destination);

choose(farmer, products, quality_level);

pack(farmer,products,box);

Method: M21
Task: brush(farmer,products) (IR2b)
Precondition:
not Product_correctly_washed
Control:
message("not correctly washed, stop");
stop;

In the next subsection, we present the translation rules, and a proof of concept.

Translation Rules

The translation of scenarios to task/method is performed thanks to the following rules:

Rule 1. Tasks Identification: each verb in the Scenario's episodes is translated into a task in Task/Method model. Each Scenario title is also a task in Task/Method model. Examples:

Episode: The farmer washes the products → Task: Wash

Episode: The farmer brushes the products → Task: Brush

Episode title: Packing the products → Task: Pack

Rule 2. Task's Parameters Identification: each actor and resource used in the episodes of a Scenario is translated by a parameter in Task / Method model. Examples:

Episode: The farmer wash the products → Task: Wash(farmer, product)

Episode: The farmer brush the products → Task: Brush(farmer, product)

Rule 3. Episode's method: the episodes part of a scenario is translated by a method in Task / Method model. Examples:

Episodes: The farmer wash the products

… → Method: M1

Rule 4. Sequence of tasks: the sequence of different lines in the episodes part of a Scenario determines the sequence of tasks in the control part of a method in the Task / Method model. The use of expressions like "then", "after", etc... in the episodes of a Scenario determines also a sequence of tasks in the method's control part. Examples:

Episodes:

The farmer washes the products *or* The farmer washes the products, then brushes the products

The farmer brushes the products

Method: M1

Control: wash(farmer, product); brush(farmer, product)

Rule 5. Test Case Method: In this work, we assume that each test case (Test cases part of scenario) corresponds to the achievement status (succeed or fail) of a task. In a failure situation, the scenario will stop. This stop case will be represented by a method for the next task in which the precondition field correspond to the test case failure. For example:

Test case: Temperature forecast obtained / not obtained →

Method: M21

Task: bush(farmer,products) # next task

precondition: not Product_correctly_washed

Control: message("not correctly washed, stop"); stop;

The natural language used in the expression of scenarios is limited, we also assume that episodes part only contains a "nominal" way to achieve a scenario, i.e. we consider that execution of every task succeeds. In the test cases part, only failures of task achievement are considered. With these assumptions and a few translation rules, it is possible to automatically translate scenarios such as the packing scenario. Of course, this automatic translation has to be used as a preliminary design and it should be analyzed and enriched by a test case designer.

PROOF OF CONCEPT

We are developing a web application where users can describe Scenarios and obtain the Task/Method model that implements the UATs. Figure 1 depicts the client/server architecture: a client module allows describing the Scenarios and see the Task/Method model obtained from them. The server module derives the Task/Method from the Scenarios. Derivation relies on a Natural Language Processor and a set of rules. Rules determine scenarios' processing, elements to be identified, and how they must be interrelated to produce the Task/Method. The NLP processes Scenarios, obtaining the elements determined by the rules.

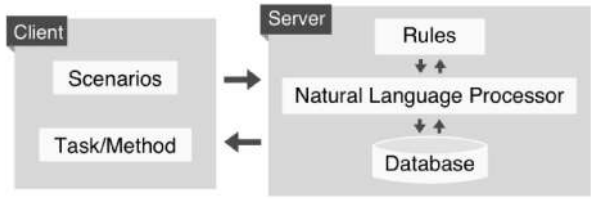


Figure 1. Architecture of the application

The system is being implemented with Node.js (https://nodejs.org) for the backend, and Angular 2 (https://angular.io) for the client. This will allow the user to define the scenarios and trigger the mechanism to obtain the Task/Method executing the rules specified in the server side through the Stanford Natural Language Processing Framework (https://nlp.stanford.edu). After that, the final translation will be shown to the client.

CONCLUSION

In this paper we have shown a new way of generating acceptance tests from well-known requirements artifacts, by presenting a set of rules to guide the implementation of semi-automated solutions and shown the first steps towards a supporting tool. We are now working in completing the ruleset, by adding the rules required to translate iterative episodes into tasks. For example, each verb used in the episode of a Scenario that describes iteration should be written as a while expression in in Task/Method model: while <condition> <block>. This would help to support other scenarios that require iterative tasks, e.g. "For each order, determine the time needed to take the products to the destination". We also plan to publish the web application in order to experiment with the presented ideas in real development settings, so we can assess the benefits of semi-automatically generated UATs.

ACKNOWLEDGEMENTS

Authors of this publication acknowledge the contribution of the Project 691249, RUC-APS: Enhancing and implementing Knowledge based ICT solutions within high Risk and Uncertain Conditions for Agriculture Production Systems (www.ruc-aps.eu), funded by the European Union under their funding scheme H2020-MSCA-RISE-2015

REFERENCES

[1] G. Camilleri, J.-L. Soubie, and J. Zalaket, "TMMT: Tool Supporting Knowledge Modelling," in *Knowledge-Based Intelligent Information and Engineering Systems*, vol. 2773, 2003, pp. 45–52.

[2] M. Boucher and G. Mussbacher, "Transforming Workflow Models into Automated End-to-End Acceptance Test Cases," in *Proceedings - 2017 IEEE/ACM 9th International Workshop on Modelling in Software Engineering, MiSE 2017*, 2017, pp. 68–74.

[3] N. Nomura, Y. Kikushima, and M. Aoyama, "A Test Scenario Design Methodology Based on Business Context Modeling and Its Evaluation," *2014 21st Asia-Pacific Softw. Eng. Conf.*, vol. 1, pp. 3–10, 2014.

[4] E. Sarmiento, J. C. S. P. Leite, E. Almentero, and G. Sotomayor Alzamora, "Test Scenario Generation from Natural Language Requirements Descriptions based on Petri-Nets," *Electron. Notes Theor. Comput. Sci.*, vol. 329, pp. 123–148, 2016.

[5] P. Pandit, S. Tahiliani, and M. Sharma, "Distributed agile: Component-based user acceptance testing," in *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*, 2016, pp. 1–9.

[6] R. Lipka, T. Potuak, P. Brada, P. Hnetynka, and J. Vinarek, "A Method for Semi-Automated Generation of Test Scenarios Based on Use Cases," in *Proceedings - 41st Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2015*, 2015, pp. 241–244.

[7] I. Alexander and N. Maiden, "Scenarios, Stories, and Use Cases: The Modern Basis for System Development," *IEEE Comput. Control Eng.*, vol. 15, no. 5, pp. 24–29, 2004.

[8] M. Jackson, "The world and the machine," in *Proceedings of the 17th international conference on Software engineering - ICSE '95*, 1995, pp. 283–292.

[9] R. Young, *The requirements engineering handbook*. 2004.

[10] A. Hussain *et al.*, "Review on formalizing use cases and scenarios: Scenario based testing," *2015 Int. Conf. Emerg. Technol.*, vol. 3, no. 3, p. 1, 2015.

[11] F. Trichet and P. Tchounikine, "DSTM: A framework to operationalise and refine a problem solving method modeled in terms of tasks and methods," *Expert Syst. Appl.*, vol. 16, no. 2, pp. 105–120, 1999.

[12] G. Schreiber, H. Akkermans, A. Anjewierden, R. De Hoog, N. R. Shadbolt, and B. Wielinga, *Knowledge Engineering and Management: The CommonKADS Methodology*, vol. 99. 2000.