Un enfoque híbrido LLM + OCL para la detección de inconsistencias en escenarios

Gabriela Pérez[1,2], Catalina Mostaccio[1], and Leandro Antonelli[1,3]

[1] LIFIA, Facultad de Informática, Universidad Nacional de La Plata
[2] UNAJ IIyA, Universidad Nacional Arturo Jauretche
[3] CAETI, Facultad de Tecnología Informática, Universidad Abierta Interamericana
{gperez, catty, lanto}@lifia.info.unlp.edu.ar

Resumen La especificación de requisitos mediante escenarios escritos en lenguaje natural es una práctica usada en ingeniería de requisitos. La forma narrativa de los escenarios dificulta su análisis sistemático, dado que los episodios suelen introducir actores y recursos de forma implícita y la consistencia entre la declaración estructural del escenario y su contenido narrativo raramente puede verificarse de manera automática. Este trabajo propone un enfoque híbrido que combina modelos de lenguaje de gran escala con invariantes formales expresadas en OCL para abordar este problema. Se define un metamodelo basado en los escenarios definidos por Leite, que formaliza los conceptos del dominio y las propiedades que toda especificación válida debe satisfacer. El proceso sigue un pipeline de cuatro etapas que incluye la construcción del modelo mediante un esquema de dos pasadas con intersección semántica, la resolución de referencias entre escenarios y la evaluación de invariantes OCL. El enfoque fue implementado en Python y validado sobre un corpus del dominio de huerta urbana, detectando inconsistencias concretas entre las declaraciones estructurales de los escenarios y el contenido narrativo de sus episodios.

Keywords: Ingeniería de Requerimientos · Escenarios · OCL · Validación formal · LLMs · Metamodelo

1. Introducción

Durante la ingeniería de requisitos, la representación de los comportamientos esperados del sistema constituye una actividad clave para comprender el dominio y alinear la visión entre los distintos involucrados. Entre las técnicas utilizadas con este propósito, los escenarios propuestos por [1] permiten describir situaciones del dominio mediante narrativas estructuradas que incluyen elementos como título, objetivo, contexto, actores, recursos y episodios. Esta forma de representación facilita la comunicación con los clientes al emplear lenguaje natural y centrarse en situaciones concretas del uso del sistema.

Sin embargo, el carácter narrativo de los escenarios también introduce ciertas limitaciones cuando se busca analizarlos de manera sistemática o verificar su

Proceedings of the 29th Workshop on Requirements Engineering (WER2026),
August 11-13, 2026, La Plata, Argentina.
https://doi.org/10.29327/1884517.29-16

consistencia. Al estar escritos en lenguaje natural, la identificación explícita de los elementos que los componen puede resultar ambigua o incompleta. En particular, los episodios suelen introducir actores o recursos que no siempre aparecen declarados previamente en las secciones estructurales del escenario, lo que dificulta la detección de inconsistencias y la verificación de propiedades estructurales. Esta situación limita la posibilidad de realizar chequeos automáticos de calidad o consistencia sobre los escenarios definidos durante el proceso de requisitos.

Una forma de abordar este problema consiste en representar la estructura de los escenarios mediante un metamodelo que formalice sus componentes fundamentales y las relaciones entre ellos. Sobre esta formalización es posible declarar invariantes que expresen propiedades que todo escenario válido debe satisfacer, habilitando así mecanismos de verificación automática. El proceso comienza con el análisis de escenarios escritos en lenguaje natural, a partir del cual se realiza un procesamiento preliminar que identifica los campos estructurales del escenario, como el título, objetivo, actores, recursos y el contexto. Posteriormente, los episodios se analizan mediante un modelo de lenguaje de gran escala, con el objetivo de inferir actores y recursos implícitos en la narrativa. Esa información se utiliza para instanciar el metamodelo propuesto. A partir de esta representación, se realizan diversos chequeos de consistencia, que permiten comprobar, por ejemplo, que los actores y recursos empleados en los episodios estén correctamente declarados, que las referencias sean coherentes dentro del escenario y que se cumplan las restricciones definidas por el metamodelo.

El trabajo presenta un enfoque híbrido que combina modelos de lenguaje a gran escala con un metamodelo formal para la detección de inconsistencias en escenarios de requisitos. La propuesta define un metamodelo con invariantes OCL para verificación automática de propiedades estructurales, un pipeline de análisis en cuatro etapas que integra extracción e inferencia desde los escenarios escritos en lenguaje natural, y una estrategia de procesamiento en dos pasadas que mejora la identificación de actores y recursos implícitos. Además, se implementa una herramienta que facilita la aplicación práctica del enfoque. De este modo, el método combina la expresividad del lenguaje natural con mecanismos de análisis estructural y verificación automática, aprovechando las capacidades actuales de los modelos de lenguaje para incorporar información implícita en la representación formal del escenario.

El resto del trabajo se organiza de la siguiente manera. La sección 2 describe los trabajos relacionados con el enfoque propuesto. La sección 3 presenta los conceptos fundamentales sobre escenarios, OCL y modelos de lenguaje de gran escala. La sección 4 define el metamodelo de escenarios y las invariantes OCL asociadas. La sección 5 describe el enfoque híbrido y el pipeline de cuatro etapas. La sección 6 presenta la implementación e ilustra el funcionamiento del enfoque mediante un ejemplo concreto del dominio de huerta urbana. Finalmente, la sección 7 presenta las conclusiones y las líneas de trabajo futuro.

2. Trabajos Relacionados

Algunos trabajos han propuesto modelos conceptuales para representar la estructura de los escenarios y sus elementos. Los escenarios han sido ampliamente reconocidos como una estrategia efectiva para comprender el comportamiento del sistema [2], [3], [4]. En particular, en [5] se describe una estrategia de derivación de escenarios a partir del Léxico Extendido del Lenguaje (LEL), donde los escenarios se construyen identificando actores, recursos y episodios a partir de símbolos del léxico. Este enfoque permite estructurar la información contenida en los escenarios y establecer relaciones entre elementos como actores, recursos, episodios y estados del dominio, dando lugar a un modelo conceptual cercano a una representación entidad–relación. Sin embargo, estas propuestas se centran principalmente en el proceso de derivación y documentación, mientras que la verificación de la consistencia estructural de los escenarios continúa realizándose en gran medida de forma manual.

Por otro lado, diversos trabajos han abordado el análisis de consistencia en escenarios escritos en lenguaje natural. En particular, [6] proponen un enfoque automatizado que combina procesamiento de lenguaje natural, redes de Petri y técnicas de visualización para verificar propiedades estructurales y conductuales. Si bien este enfoque aborda problemáticas similares a las consideradas en este trabajo, requiere que los escenarios estén expresados en una forma restringida del lenguaje natural y no incorpora mecanismos de verificación formal basados en un metamodelo explícito. Por su parte, [7] propone un enfoque que combina la definición de escenarios utilizando OWL con consultas semánticas en SPARQL para identificar inconsistencias, redundancias y problemas de calidad en la descripción de escenarios colaborativos en el sector agronindustrial. Sin embargo, este método requiere que el analista sea quien prepare y defina el escenario en términos de ontologías.

En los últimos años, el uso de modelos de lenguaje de gran escala (LLMs) ha ganado creciente atención en el ámbito de la ingeniería de software, especialmente en tareas que involucran la comprensión y transformación de lenguaje natural. Diversos estudios han demostrado su potencial para asistir en una amplia variedad de tareas de ingeniería de requisitos, desde la elicitación y validación hasta la generación de casos de prueba [8]. En este contexto, han comenzado a surgir enfoques que integran LLMs en procesos de verificación formal, con el objetivo de reducir la brecha entre requisitos informales y modelos verificables.

En [9], se propone un enfoque de verificación formal asistida por LLMs que automatiza la transformación de requisitos en lenguaje natural en propiedades verificables mediante su integración con herramientas de model checking. A diferencia de dicho enfoque, que prioriza la automatización end-to-end del proceso, nuestro trabajo adopta una estrategia híbrida que combina LLMs con invariantes formales expresadas en OCL sobre un metamodelo de escenarios. Mientras que este tipo de propuestas utiliza LLMs para generar directamente aserciones verificables sobre código, nuestro enfoque restringe su rol a la interpretación semántica del lenguaje natural, delegando la verificación en un modelo formal explícito. Esto permite un mayor control sobre la consistencia estructural de los

escenarios y reduce el riesgo de errores derivados de interpretaciones incorrectas, a costa de una menor automatización del proceso completo.

3. Background

El presente trabajo se apoya en tres áreas conceptuales que se describen a continuación. Por un lado, los escenarios como artefactos de especificación de requisitos. Por otro, el lenguaje OCL como mecanismo de expresión de invariantes sobre metamodelos. Finalmente, los modelos de lenguaje de gran escala como herramienta de interpretación de texto en lenguaje natural.

3.1. Escenarios

Los escenarios son artefactos que permiten explicar cómo funciona un sistema a través de la narración de historias que describen situaciones del dominio. Este enfoque resulta efectivo porque permite incorporar detalles que son esenciales para una comprensión clara y completa de su funcionamiento. Tanto desarrolladores como expertos del dominio pueden utilizarlos sin necesidad de aprender formalismos complejos, lo que facilita la comunicación entre ellos. En [1] se define un escenario como una estructura compuesta por los siguientes atributos: (i) un título; (ii) un objetivo que debe alcanzarse mediante la ejecución del escenario; (iii) un contexto, que establece el punto de partida del escenario y se describe mediante precondiciones y postcondiciones; (iv) los recursos, que corresponden a objetos físicos o información que debe estar disponible; (v) los actores, que son los agentes que realizan las acciones; (vi) el conjunto de episodios, que describe la secuencia de acciones que ocurren en el escenario; y (vii) las excepciones, que representan situaciones alternativas o condiciones de falla. Cada episodio describe acciones realizadas por los actores utilizando los recursos disponibles y representa la evolución del escenario a lo largo del tiempo.

Los escenarios no son descripciones aisladas, sino que forman una red de escenarios. En lugar de describir todo el comportamiento en un único escenario, se construyen varios escenarios más pequeños que representan situaciones específicas del dominio y se relacionan entre sí formando una red que permite describir el comportamiento del sistema de manera modular y reutilizable.

Generalmente, a partir de las primeras entrevistas se identifican escenarios iniciales. Luego, al detallar los episodios, algunas acciones se expanden a nuevos escenarios. Estos nuevos escenarios se referencian desde los episodios originales. Esto también permite evitar repetir comportamientos similares.

3.2. OCL y verificación de metamodelos

El Object Constraint Language (OCL) es un lenguaje declarativo, parte del estándar UML, diseñado para expresar restricciones sobre modelos y metamodelos que no pueden representarse mediante notación gráfica [10]. OCL permite

definir invariantes, es decir, reglas que deben ser satisfechas por todas las instancias de un tipo, y precondiciones y postcondiciones de operaciones. En el enfoque propuesto, OCL se emplea como lenguaje de especificación de invariantes sobre el metamodelo de escenarios. Dado que la evaluación de invariantes OCL se implementa directamente en Python sobre las instancias del metamodelo, el enfoque no requiere herramientas MDE específicas, lo que facilita su integración en flujos de trabajo existentes de ingeniería de requisitos.

3.3. Grandes Modelos de Lenguaje

Los modelos de lenguaje de gran escala (LLMs) son sistemas basados en la arquitectura Transformer [13], entrenados sobre grandes volúmenes de texto y capaces de generar y transformar lenguaje natural. En el ámbito de la ingeniería de requisitos, han comenzado a aplicarse en tareas como la extracción de requisitos a partir de documentos no estructurados, la clasificación de requisitos funcionales y no funcionales, la detección de ambigüedades y la generación de casos de prueba [8]. Su capacidad para analizar el contexto semántico de oraciones complejas los convierte en herramientas que pueden ayudar a reducir el esfuerzo manual en la fase de análisis. No obstante, presentan limitaciones cuando se emplean de forma aislada en tareas que requieren razonamiento lógico estricto. Dado que se basan en el aprendizaje de patrones estadísticos del lenguaje y no en la aplicación explícita de reglas formales, carecen de mecanismos internos de verificación que garanticen la validez de las inferencias. Como consecuencia, pueden producir resultados inconsistentes, sensibles a variaciones en los prompts, o directamente incorrectos al razonar sobre estructuras formales. En este contexto, su uso resulta más adecuado dentro de enfoques híbridos, donde el LLM se encarga de la interpretación semántica del lenguaje natural, mientras que un sistema formal independiente valida dichas representaciones y garantiza la corrección de las inferencias [9].

4. Metamodelo

El metamodelo propuesto formaliza la estructura conceptual de los escenarios mediante un conjunto de clases, asociaciones e invariantes OCL. Cada escenario se representa como una instancia de la clase Escenario, compuesta por actores, recursos, episodios, condiciones y excepciones. Los escenarios se agrupan en un Corpus que centraliza el vocabulario global y habilita la verificación de propiedades inter-escenario. La Figura 1 presenta el diagrama de clases del metamodelo.

El metamodelo distingue dos tipos de episodios, los episodios simples, que describen acciones directas realizadas por actores sobre recursos, y los episodios complejos, que representan referencias a otros escenarios del corpus. Esta distinción permite modelar la composición jerárquica de escenarios. Las condiciones se representan mediante la clase Condición y se utilizan para precondiciones y postcondiciones de escenarios. Las condiciones asociadas a las excepciones no se

consideran en este primer trabajo y se plantean como trabajo futuro. Las excepciones se modelan como instancias de la clase Excepción, asociadas al episodio en el que pueden ocurrir. Sobre este metamodelo se definen invariantes OCL que expresan las propiedades que toda especificación válida debe satisfacer. A continuación, se describen las principales clases del metamodelo junto con sus asociaciones e invariantes.

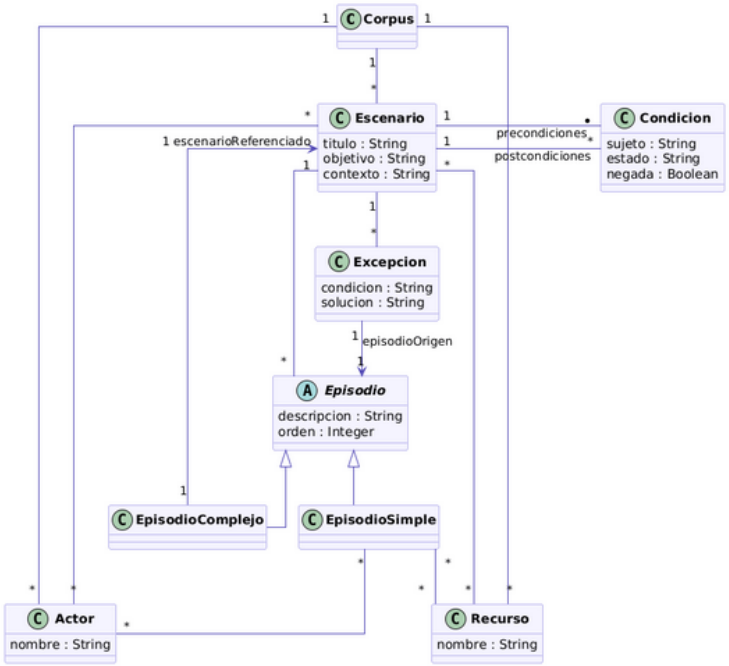

Figura 1. Metamodelo propuesto para un corpus de escenarios.

4.1. Corpus

Un Corpus representa el contenedor principal de un conjunto de escenarios. Contiene todos los escenarios y mantiene el vocabulario global de actores y recursos utilizados en ellos.

Asociaciones

- **escenarios:** Conjunto de escenarios que pertenecen al corpus.
- **actores:** Actores globales utilizados en los escenarios del corpus.
- **recursos:** Recursos compartidos entre los escenarios del corpus.

Restricciones

[inv-1] Los escenarios contenidos en el corpus deben tener títulos únicos.

```
inv EscenariosConTitulosUnicos:
  self.escenarios->isUnique(e | e.titulo.toLower().trim())
```

[inv-2] Los recursos contenidos en el corpus deben tener nombres únicos.

```
inv RecursosConNombresUnicos:
  self.recursos->isUnique(r | r.nombre.toLower().trim())
```

[inv-3] Los actores contenidos en el corpus deben tener nombres únicos.

```
inv ActoresConNombresUnicos:
  self.actores->isUnique(r | r.nombre.toLower().trim())
```

[inv-4] El conjunto de actores del Corpus es la unión de los actores de todos los escenarios.

```
inv ActoresCorpusCompletos:
  self.actores->asSet() = self.escenarios
  ->collect(e | e.actores)->flatten() ->asSet()
```

[inv-5] El conjunto de recursos del Corpus es la unión de los recursos de todos los escenarios.

```
inv RecursosCorpusCompletos:
  self.recursos->asSet() = self.escenarios
  ->collect(e | e.recursos)->flatten()->asSet()
```

4.2. Escenario

Un escenario describe una situación específica dentro del sistema, incluyendo su objetivo, el contexto en el que ocurre y una secuencia ordenada de episodios que representan la interacción entre actores y recursos.

Asociaciones

- **actores:** Actores que participan en el escenario.
- **recursos:** Recursos utilizados en el escenario.
- **episodios:** Secuencia de episodios que describen el desarrollo del escenario.
- **precondiciones:** Condiciones que deben cumplirse antes de que el escenario pueda ejecutarse.
- **postcondiciones:** Condiciones que deben cumplirse después de la ejecución del escenario.
- **excepciones:** Conjunto de excepciones asociadas al escenario, cada una vinculada a un episodio específico en el que puede ocurrir.

Restricciones

[inv-6] Todo escenario debe tener título.

```
inv TituloNoVacio:
  self.titulo <> ''
```

[inv-7] Todo escenario debe tener un objetivo.

```
inv ObjetivoNoVacio:
  self.objetivo <> ''
```

[inv-8] Todo escenario debe tener al menos un episodio.

```
inv TieneAlMenosUnEpisodio:
  self.episodios->size() > 0
```

[inv-9] Todo actor que participa en un episodio debe estar declarado en el escenario, y todo actor declarado en el escenario debe participar en al menos un episodio.

```
inv ActoresConsistentes:
  self.actores->asSet() = self.episodios
      ->select(ep | ep.oclIsTypeOf(EpisodioSimple))
      ->collect(ep | ep.oclAsType(EpisodioSimple).actores)
      ->flatten()->asSet()
```

[inv-10] Todo recurso utilizado en un episodio debe estar declarado en el escenario, y todo recurso declarado en el escenario debe ser utilizado por al menos un episodio.

```
inv RecursosConsistentes:
  self.recursos->asSet() = self.episodios
      ->select(ep | ep.oclIsTypeOf(EpisodioSimple))
      ->collect(ep | ep.oclAsType(EpisodioSimple).recursos)
      ->flatten()->asSet()
```

[inv-11] No pueden coexistir condiciones contradictorias dentro de un mismo conjunto de precondiciones. Es decir, no se puede contener simultáneamente una condición y su negación.

```
inv SinContradicionesEnPrecondiciones:
  self.precondiciones->forAll(c1, c2 |
    c1 <> c2 and c1.sujeto = c2.sujeto and c1.estado = c2.estado
      implies c1.negada = c2.negada)
```

[inv-12] Igual que la regla anterior, pero en el conjunto de postcondiciones.

```
inv SinContradicionesEnPostcondiciones:
  self.postcondiciones->forAll(c1, c2 |
    c1 <> c2 and c1.sujeto = c2.sujeto and c1.estado = c2.estado
      implies c1.negada = c2.negada)
```

4.3. EpisodioComplejo

Un episodio complejo es subclase de un Episodio, y representa una referencia a otro escenario, permitiendo reutilizar la definición de ese escenario dentro de la secuencia de episodios.

Asociaciones

- **escenarioReferenciado:** Escenario que es invocado o reutilizado por el episodio.

Para evitar dependencias recursivas entre escenarios, el metamodelo prohíbe la existencia de ciclos en el grafo de referencias entre escenarios. Esta restricción se formaliza mediante una invariante OCL que verifica que un escenario no sea alcanzable transitivamente a partir de sí mismo a través de episodios complejos. De esta forma se garantiza que las referencias entre escenarios formen un grafo acíclico.
[inv-13] Un escenario no puede ser alcanzado transitivamente a partir de sí mismo a través de episodios complejos (no hay ciclos en el grafo de referencias).

```
inv SinCiclos:
  not self.escenariosReferenciados
  ->closure(e | e.escenariosReferenciados) ->includes(self)

-- Operación auxiliar
def: escenariosReferenciados : Set(Escenario)
 self.episodios ->select(ep | ep.oclIsKindOf(EpisodioComplejo))->
collect(ep | ep.oclAsType(EpisodioComplejo).escenarioReferenciado)
->asSet()
```

5. Enfoque híbrido. LLMs y OCL

El enfoque propuesto combina el procesamiento de lenguaje natural mediante modelos de lenguaje de gran escala (LLM) con la evaluación formal de invariantes expresadas en OCL, con el objetivo de detectar inconsistencias en escenarios escritos en lenguaje natural. El proceso se organiza en un pipeline de cuatro etapas que transforma escenarios no estructurados en instancias del metamodelo y las somete a verificación automática.

5.1. Visión general del pipeline

La Figura 2 ilustra las cuatro etapas del pipeline. La entrada es un conjunto de escenarios en formato JSON, donde cada entrada contiene los campos estructurales del escenario en texto libre. La salida es un reporte de invariantes que indica, para cada escenario, qué propiedades se satisfacen y cuáles presentan inconsistencias.

En la primera etapa se extraen los campos estructurales declarados explícitamente en cada escenario: título, objetivo, contexto, actores, recursos y episodios. En la segunda etapa se invoca al LLM para inferir los actores y recursos mencionados implícitamente en los episodios, completando así la información necesaria para instanciar el metamodelo. En la tercera etapa se resuelven las referencias entre escenarios, vinculando los episodios complejos con los objetos Escenario correspondientes y asociando las excepciones a sus episodios de origen. Finalmente, en la cuarta etapa se evalúan las invariantes OCL sobre cada instancia construida.

Figura 2. Visión general del pipeline de cuatro etapas

5.2. Extracción de información estructural

En esta etapa se toma como entrada cada escenario en formato JSON y se extraen los campos estructurales declarados explícitamente: título, objetivo, contexto, actores, recursos y episodios. Esta información se encuentra disponible de forma directa en el texto del escenario y no requiere inferencia. Los campos extraídos se utilizan para inicializar la instancia del metamodelo correspondiente a cada escenario.

Durante este proceso se realiza además la unificación de entidades a nivel de corpus. Si un mismo actor o recurso aparece declarado en múltiples escenarios, se lo trata como una única instancia compartida dentro del Corpus, evitando duplicaciones y garantizando la consistencia del vocabulario global.

5.3. Inferencia de actores y recursos en episodios

La segunda etapa se centra en la inferencia de actores y recursos mencionados implícitamente en los episodios. Para este propósito se utilizó Gemma 3 (12b)[14], ejecutado en forma local mediante Ollama [15]. Este modelo analiza el escenario y extrae de manera automática los elementos implícitos que luego se incorporan al metamodelo. Dado que la redacción es en lenguaje natural, los actores y recursos que intervienen en cada uno de los escenarios no se encuentran explícitamente identificados, sino que aparecen como menciones implícitas dentro de la narrativa. Para extraer esta información, se diseñó inicialmente un esquema de análisis episodio por episodio, en el que se invoca al LLM con un prompt para identificar los actores que realizan la acción y los recursos utilizados, empleando exactamente las palabras presentes en el texto sin inferir entidades no mencionadas. Esta estrategia presentó una limitación sistemática: al analizar

cada episodio de forma aislada, el LLM no dispone de contexto suficiente para distinguir si un objeto mencionado es un recurso externo o un artefacto producido por una acción anterior. Por ejemplo, en el escenario Trasplantar, el episodio "el huertero coloca una semilla en cada orificio" llevó al modelo a identificar orificio como recurso, cuando los orificios fueron creados en el episodio anterior. Este tipo de errores contamina la evaluación de las invariantes INV-9 e INV-10. Para resolver esta limitación se adoptó un esquema de dos pasadas con intersección semántica. En la primera pasada se mantiene el análisis episodio por episodio, preservando el nivel de detalle de la extracción. En la segunda pasada el LLM recibe el escenario completo y produce una lista general de actores y recursos, pudiendo razonar sobre el contexto completo y descartar artefactos intermedios. El resultado final de cada episodio se obtiene a partir de la intersección entre ambas pasadas, conservando únicamente las entidades identificadas en el análisis individual que también aparecen en el análisis global. La comparación se realiza mediante matching semántico con embeddings vectoriales [11], [12] lo que permite resolver variaciones morfológicas como diferencias de número o errores tipográficos presentes en el texto original.

5.4. Resolución de referencias

Una vez construidas todas las instancias del metamodelo, se resuelven las referencias entre escenarios. Los episodios complejos, que invocan a otro escenario, se vinculan con el objeto Escenario correspondiente mediante su título. De manera similar, las excepciones se asocian al episodio de origen dentro del mismo escenario. Esta etapa es necesaria para poder evaluar posteriormente las invariantes que involucran relaciones entre escenarios, como la detección de ciclos en el grafo de referencias.

5.5. Evaluación de invariantes OCL

La última etapa aplica el conjunto de invariantes OCL definidas sobre el corpus instanciado. Las invariantes están implementadas directamente en Python sobre las instancias del metamodelo, sin requerir herramientas MDE específicas. Cada invariante evalúa una propiedad estructural del escenario y produce un resultado booleano acompañado de un mensaje de detalle en caso de fallo. Las invariantes INV-9 e INV-10 son especialmente relevantes en el contexto del enfoque, ya que verifican la consistencia entre la declaración estructural del escenario y la información inferida por el LLM en la etapa anterior. Si estas invariantes no se cumplen, indica que el LLM identificó actores o recursos en los episodios que no fueron declarados en el encabezado del escenario, o que hay entidades declaradas que no aparecen en ningún episodio.

6. Implementación y evaluación

El enfoque descripto en la sección anterior fue implementado como una herramienta de escritorio desarrollada en Python, que toma como entrada un archivo

JSON con escenarios en lenguaje natural y produce como salida un reporte de invariantes y una visualización interactiva de los escenarios. La herramienta integra un pipeline de procesamiento, donde identifica actores y recursos declarados, e infiere de los episodios los actores y recursos. Luego evalúa las invariantes OCL definidas en el metamodelo.

La Figura 3 muestra la herramienta implementada. A la izquierda puede verse un grafo con los escenarios, las referencias entre ellos y los actores que participan. A la derecha se muestran dos paneles, uno con los resultados de la evaluación de las invariantes sobre el escenario seleccionado, y el otro con el resultado a nivel corpus.

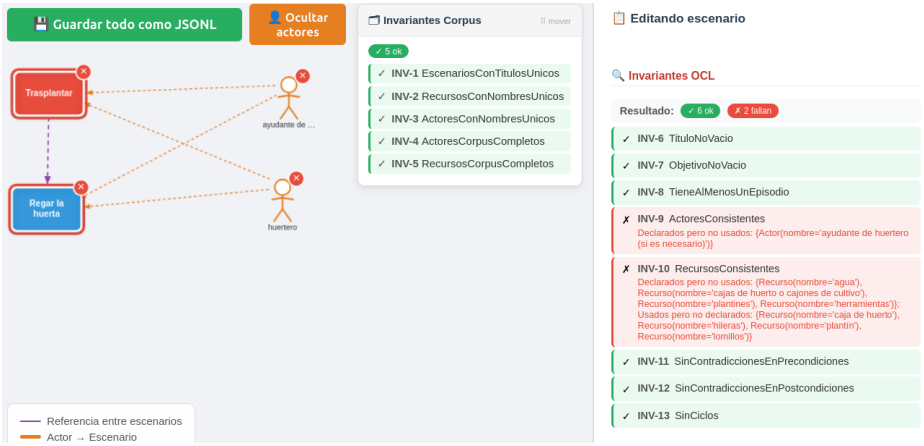

Figura 3. Herramienta implementada

El enfoque fue validado sobre un corpus del dominio de huerta urbana compuesto por ocho escenarios: Sembrar semillas de tomate, Realizar Injerto, Injertar, Sembrar el semillero, Abonar el semillero, Regar el semillero, Trasplantar y Regar la huerta. Este dominio fue seleccionado por su riqueza terminológica y por la presencia de relaciones inter-escenario, lo que permite poner a prueba tanto las invariantes locales como las propiedades del corpus.

Las invariantes INV-1, INV-2, INV-3, INV-4, INV-5, INV-6, INV-7, INV-8, INV-13 se satisfacen en la totalidad de los escenarios, lo cual era esperado dado que estas propiedades dependen de campos estructurales que se extraen directamente del encabezado del escenario sin intervención del LLM, y los escenarios se encontraban completos. Las invariantes críticas del enfoque, INV-9 e INV-10, que verifican la consistencia entre la declaración estructural y el contenido narrativo, presentan inconsistencias en siete de los ocho escenarios respectivamente. Las inconsistencias en la invariante INV-9 responde a dos causas. La primera es la presencia de actores en episodios que no están declarados en el encabezado. Por ejemplo, en Injertar aparece el huertero en episodios aunque el encabeza-

do solo declara al ingeniero agrónomo. La segunda causa es la inversa, donde actores declarados que no participan en ningún episodio, lo que indica una sobredeclaración en el encabezado. Varios episodios están además redactados en forma impersonal, como "Encalar la tierra" o "Esperar en promedio de cinco a ocho días", lo que impide al LLM inferir un actor participante, generando una diferencia con los actores declarados. Las inconsistencias detectadas por INV-10 responden principalmente a la identificación de recursos utilizados en los episodios que no fueron declarados en el encabezado del escenario. Por ejemplo, en uno de los escenarios el episodio "El huertero pulveriza con agua el almácigo" introduce el agua como recurso sin que esté declarado previamente. En otro caso, el encabezado declara "herramientas" como recurso pero los episodios referencian herramientas específicas que no coinciden con esa declaración genérica. Si bien estos casos requieren una corrección manual, el enfoque demuestra su utilidad como herramienta de apoyo para mejorar la calidad y consistencia de la redacción.

Por otro lado, la validación se realizó sobre un corpus reducido de ocho escenarios pertenecientes a un único dominio. Si bien esto permite demostrar la operatividad del enfoque y caracterizar los tipos de inconsistencias detectables, limita la generalización de los resultados. El enfoque está orientado a situaciones donde la revisión manual es poco viable, como corpus de gran tamaño donde verificar la consistencia de forma exhaustiva resulta impráctico. La extensión a un corpus mayor y dominios variados queda como línea de trabajo futuro.

7. Conclusiones y trabajo futuro

Este trabajo presentó un enfoque híbrido que combina modelos de lenguaje de gran escala con invariantes formales expresadas en OCL para la detección automática de inconsistencias en escenarios de requisitos. Se trata de un trabajo exploratorio que busca demostrar la viabilidad del enfoque, sin pretender aún resultados definitivos.

La validación sobre un corpus de ocho escenarios del dominio de huerta urbana permitió demostrar la operatividad del enfoque y caracterizar los tipos de inconsistencias que el sistema es capaz de detectar. Los resultados muestran que las invariantes INV-9 e INV-10, las más relevantes para los objetivos del enfoque, detectaron inconsistencias en la mayoría de los escenarios analizados. Estas inconsistencias corresponden a tres categorías concretas: actores introducidos en episodios sin estar declarados en el encabezado, actores declarados que no participan en ningún episodio, y recursos con discrepancias entre la declaración estructural y el contenido narrativo. El esquema de dos pasadas con intersección semántica demostró ser efectivo para reducir artefactos intermedios incorrectamente identificados como recursos. Sin embargo, se observó que los resultados son sensibles a variaciones en los prompts utilizados, lo que constituye una limitación a considerar en trabajos futuros.

Como trabajo futuro se identifican varias líneas de extensión. En primer lugar, incorporar la inferencia de precondiciones y postcondiciones al pipeline. En

segundo lugar, extender la evaluación a corpus de mayor tamaño y dominios distintos. En tercer lugar, explorar estrategias de prompting más sofisticadas para episodios redactados en forma impersonal. Finalmente, incorporar el tratamiento de excepciones al pipeline.

Referencias

1. Julio Cesar Sampaio do Prado Leite, Gustavo Rossi, Federico Balaguer, Vanesa Maiorana, Gladys Kaplan, Graciela Hadad, Alejandro Oliveros: Enhancing requirements baseline with scenarios. Requirements Engineering Journal, vol. 2, no. 4, pp. 184–198 (1997).
2. Carroll, J. M.: Five reasons for scenario-based design. Proceedings of the 32nd Annual Hawaii International Conference on Systems Sciences, (1999).
3. Antonelli, L., Delle Ville, J., Dioguardi, F., Fernandez, A., Tanevitch, L., Torres, D.: An Iterative and Collaborative Approach to Specify Scenarios using Natural Language. Workshop on Requirements Engineering (WER) 2022.
4. Alexander, I., Maiden, N.: Scenarios, stories, and use cases: the modern basis for system development. Computing Control Engineering Journal 15(5), 24–29 (2004).
5. Leite, J.C.S.P., Hadad, G., Doorn, J. y Kaplan, G. A scenario construction process. *Requirements Engineering*, 5(1):38–61, 2000.
6. Sarmiento-Calisaya, E. y Leite, J.C.S.P. Early analysis of requirements using NLP and Petri-nets. *Journal of Systems and Software*, 208:111901, 2024. DOI: https://doi.org/10.1016/j.jss.2023.111901
7. Leandro Antonelli, Diego Torres, Mariángeles Hozikian, Jorge E. Hernández: Semantic Support for Scenarios to Improve Communication in Agribusiness. PRO-VE 2019: 447-456
8. Zadenoori, Mohammad and Dąbrowski, Jacek and Alhoshan, Waad and Zhao, Liping and Ferrari, Alessio. (2025). Large Language Models (LLMs) for Requirements Engineering (RE): A Systematic Literature Review. 10.48550/arXiv.2509.11446.
9. Wang, W., Farrell, M., Cordeiro, L.C. y Zhao, L. Supporting Software Formal Verification with Large Language Models: An Experimental Study. En: *33rd IEEE International Requirements Engineering Conference (RE 2025)*, Valencia, Spain, September 1–5, 2025.
10. OCL metamodelo. https://www.omg.org/spec/OCL/2.4/PDF. Accedido en marzo 2026.
11. Reimers, N., Gurevych, I. : Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). (2019)
12. Hugging Face, https://huggingface.co/, Accedido en marzo 2026.
13. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, L., Polosukhin, I. : Attention is all you need - 31st Conference on Neural Information Processing Systems, pages 5998–6008 (NIPS 2017), Long Beach, CA, USA. (2017).
14. Gemma Team, Aishwarya Kamath, Johan Ferret, et al. Gemma 3 Technical Report. *arXiv preprint arXiv:2503.19786*, 2025. https://arxiv.org/abs/2503.19786
15. Ollama. Plataforma para ejecutar modelos abiertos de forma local, 2023. https://github.com/ollama/ollama. Accedido en marzo 2026.