
Universidad Nacional de La Pampa

Facultad de Ingeniería

PROYECTO FINAL PARA LA CARRERA DE GRADO

**INGENIERÍA EN SISTEMAS**

# Aplicación práctica de SaST, una estrategia de prueba basada en escenarios y consciente del contexto

**Autor de Proyecto Final:** Marcos Eleicegui

**Director:** Dr. Pablo Becker. **Cátedra:** Ingeniería de Software II

**Co-Director:** Dr. Luis Olsina. **Cátedra:** Ingeniería de Software II

**Jurado:**

Dra. María de los Ángeles Martín. **Filiación institucional:** Facultad de Ingeniería - UNLPam

Dra. María Fernanda Papa. **Filiación institucional:** Facultad de Ingeniería - UNLPam

Mg. Hugo Alfredo Alfonso. **Filiación institucional:** Facultad de Ingeniería - UNLPam

**General Pico - La Pampa - Argentina**

**Fecha de aprobación:** 29/03/2022

# Abstract

Nowadays, software applications have become very complex since, in some particular situations, they depend on other systems or services to perform their functions correctly. In other words, this means that a system is not isolated and is influenced by context entities. Due to its inherent complexity, some existing software testing approaches are not effective enough to verify and validate particular situations in which context entities are relevant to consider and model.

Therefore, test designers should consider testing strategies that may be useful to model different situations in which context entities may influence the testable entity. In an attempt to contribute to this area, a new scenario-based testing strategy named *Situation-aware Scenario-based Testing Strategy* (SaST *Strategy*) has been developed.

This new strategy is used in this project to design test cases for a real Web App which interacts with a number of external services.

**Keywords:** Software Testing, Testing Strategy, Ontology, Context, Situation, Scenario.

# Resumen

Muchas aplicaciones de software no están aisladas, sino que su funcionamiento se ve influenciado por otras aplicaciones o servicios. Cuando esto ocurre, diseñar las pruebas de software no es una tarea sencilla. En estas situaciones, las estrategias de diseño de prueba basadas en escenarios son de mucha utilidad ya que los casos de prueba se diseñan a partir de posibles escenarios donde participan el objeto de prueba y las entidades del contexto que la rodean.

Recientemente se ha desarrollado una estrategia basada en escenarios llamada *Situation-aware Scenario-based Testing Strategy* (SaST *Strategy*). Esta cuenta con un proceso bien definido y métodos que sirven para especificar casos de prueba. Además, incluye una base conceptual robusta que sustenta sus especificaciones.

En este trabajo se hace uso de esta nueva estrategia para diseñar los casos de prueba para una Web App real que interactúa con varios servicios. Como resultado de la aplicación se obtuvo un conjunto de casos de prueba para distintos escenarios, demostrando así que la estrategia es de gran utilidad a la hora de diseñar los casos de prueba. Además, gracias a las plantillas de especificación que proporciona la estrategia se facilita documentar de manera completa los casos de prueba.

**Palabras clave:** Pruebas de software, Estrategia de prueba, Ontología, Contexto, Situación, Escenario.

# Agradecimientos

Agradezco principalmente a mi familia y a todas las personas que me acompañaron a lo largo de este camino. Al Dr. Pablo Becker, mi tutor, al Dr. Luis Olsina, mi co-tutor, por el apoyo brindado en la producción de este trabajo final, y al Ing. Guido Tebes, por el tiempo que dedicó a contestar todas mis dudas sobre la estrategia y por la ayuda al momento de conseguir acceso a documentación de referencia y distintas herramientas utilizadas en la elaboración de la tesina.

Al equipo de Agripay, por permitirme utilizar el proyecto como objeto de estudio y por acompañarme en esta última etapa.

A todos ellos, simplemente gracias.

# Tabla de contenidos

# Índice de figuras

# Índice de tablas

# Capítulo 1: Introducción

Al día de hoy, las aplicaciones de software han ido evolucionando y se han complejizado a un nivel en el que, en ciertas ocasiones, éstas dependen de otros sistemas o servicios para llevar a cabo su funcionamiento correctamente. Esto significa que, en estas situaciones particulares, un sistema no está aislado, sino que se ve influenciado por distintas entidades. Por ello, las aplicaciones deben ser conscientes de su contexto, lo cual incluye las entidades que la rodean y afectan su funcionamiento.

Cuando se produce una falla en un sistema de software, ésta puede impactar negativamente en la forma en que el usuario lo percibe. Por ejemplo, puede suceder que el usuario pierda confianza en la aplicación e incluso conducir al abandono de la misma. Además, si es una aplicación crítica, puede implicar grandes pérdidas económicas o, incluso peor, la pérdida de vidas humanas. Con el objetivo de prevenir dichas situaciones, los ingenieros de software deberían llevar a cabo procesos de Verificación y Validación (V & V) antes de poner el producto en producción (o antes de entregarlo al cliente).

V & V involucra un conjunto de actividades, herramientas, métodos y estrategias que dan soporte al aseguramiento de la calidad. Entre las actividades específicas de V & V se pueden mencionar las pruebas de software [1]. Estas brindan un conjunto de procesos, métodos y estrategias que ayudan a cumplir con los requisitos funcionales y no funcionales establecidos. Algunos tipos de métodos de prueba se documentan en [2], mientras que sus procesos se detallan en [3]. Además, las pruebas de software también cuentan con estrategias que sirven para validar sistemas (por ejemplo, pruebas de aceptación).

Con el fin de asegurar la calidad de forma sistemática es importante utilizar estrategias de prueba que den soporte a la V & V de sistemas de software. Según Olsina y Becker [4], las estrategias sistemáticas pueden utilizarse en una organización para alcanzar diferentes metas de negocio. Además, son consideradas recursos clave de la misma, ya que, definen un curso específico de acción a seguir, es decir, especifican qué hacer y cómo hacerlo. Consecuentemente, las estrategias bien especificadas deberían integrar tres capacidades, a saber: una especificación de procesos, una especificación de métodos y una base conceptual de dominio robusta, por ejemplo, mediante el uso de una ontología. Este principio de integración tiene como beneficio que una estrategia especificará explícitamente qué actividades se deben ejecutar y cómo llevarlas a cabo por medio de métodos, todo esto en el marco de un vocabulario de dominio común.

Hoy en día existen muchos métodos o técnicas de prueba, como las ilustradas en [2]. Algunos ejemplos son: las pruebas basadas en especificaciones (caja negra), pruebas basadas en la estructura (caja blanca), pruebas estáticas (*walkthrough*, *pair reviews*, etc.), pruebas *alpha*, entre otras. Sin embargo, no todas ellas son adecuadas para probar el software en situaciones donde es importante considerar la interacción del contexto que lo rodea. Según [5], la mayoría de las estrategias de prueba se crearon con el objetivo de probar el software de forma convencional, es decir, sin considerar el contexto y sus influencias. Un enfoque que puede ser útil para diseñar pruebas considerando el contexto son las pruebas basadas en escenarios, ya que al usar esta metodología se consideran modelos de situación, los cuales podrían capturar/modelar información importante de contexto.

Considerando las definiciones presentes en [1] y [6], se puede definir las pruebas basadas en escenarios como un método para el diseño de pruebas basadas en especificaciones, es decir, de caja negra, en donde los casos de prueba son diseñados a partir de escenarios. Estos escenarios deberían representar las distintas situaciones particulares que involucren al objeto principal de prueba y a las entidades de contexto relacionadas al mismo.

Recientemente se ha desarrollado una estrategia integrada de prueba llamada SaST Strategy (*Situation-aware Scenario-based Testing Strategy*) [7], la cual está semánticamente soportada por la arquitectura ontológica FCD-OntoArch (*Foundational, Core, and Domain Ontological Architecture for Sciences*) [8]. Además, cuenta con un proceso bien definido y métodos que sirven de guía para especificar los casos de prueba. Como se dijo anteriormente, según [4] una estrategia sistemática debería integrar una especificación de proceso y una especificación de método. En el caso de la estrategia SaST, dichas especificaciones son denominadas SaSTPro (Situation-aware Scenario-based Testing Process) y SaSTMe (Situation-aware Scenario-based Testing Method).

Con el objetivo de aplicar la estrategia SaST en un contexto de uso real, en este trabajo se sigue dicha estrategia para diseñar un conjunto de casos de prueba para una plataforma en desarrollo de la empresa Agripay. Particularmente, la plataforma en cuestión está destinada a gestionar contratos inteligentes y pagos digitales. El desarrollo de esta plataforma cuenta de distintos frentes, entre los cuales se encuentran una Web App, un conjunto de aplicaciones móviles para distintos tipos de dispositivos y una API back-end que brinda *endpoints* para la interacción con la base de datos y para la comunicación con otros servicios externos que necesiten ser utilizados por las distintas aplicaciones. En el contexto de cualquier aplicación de software es fundamental verificar que el comportamiento del sistema sea el esperado para asegurar la calidad del mismo. Para ello, en los diferentes frentes de desarrollo de Agripay se llevan a cabo pruebas de software unitarias.

Particularmente, el equipo de desarrollo de la Web App realiza pruebas *ad hoc* y en forma manual de las distintas funcionalidades con el fin de verificar la correctitud de las distintas funcionalidades que la Web App brinda. En una situación donde el correcto funcionamiento de una aplicación no depende sólo de sí misma sino de su interacción con una entidad externa, es de utilidad realizar las denominadas pruebas de integración, para evaluar ese comportamiento conjunto. Ahora, considerando que la Web App de Agripay puede verse influenciada por más de un servicio de la API backend al mismo tiempo, como así también por otros servicios externos con los que se pueda estar comunicando, es necesario un enfoque distinto a la hora de diseñar pruebas sobre la misma. Por lo tanto, se decidió diseñar las pruebas de integración entre la Web App y la API backend, haciendo uso de la estrategia SaST, la cual permite utilizar un enfoque basado en escenarios y consciente del contexto.

A partir de la aplicación de SaST se pretende mejorar la calidad de la Web App que fue tomada como objeto de prueba, diseñando pruebas de integración que verifiquen que ciertos componentes de la misma se comporten como deberían, de acuerdo a los requerimientos. Además, es importante contar con una estrategía sistemática que dé soporte a la producción de documentación completa y de utilidad, que luego se consuma para la futura implementación y ejecución de las pruebas diseñadas.

En este punto es importante mencionar que el alcance de esta tesina está limitado solo al diseño de las pruebas, y no a su implementación y ejecución. Esto debido a que, estas últimas etapas serán realizadas en el marco de la empresa cuando los tiempos de desarrollo lo ameriten y el líder del equipo de desarrollo Web lo considere adecuado.

La presente tesina está dividida en 4 capítulos. En el Capítulo 1 se realizó una breve introducción de la temática a desarrollar y la motivación de este trabajo. El Capítulo 2 contiene los principales fundamentos teóricos que ayudarán a la comprensión de este trabajo. En el mismo se hace una introducción a la estrategia SaST, ilustrando la arquitectura de ontologías que le brindan soporte semántico (FCD-OntoArch) y describiendo las especificaciones de proceso y de método de la estrategia (SaSTPro y SaSTMe, respectivamente). Además, contiene una breve reseña sobre trabajos relacionados. En el Capítulo 3 se presenta el desarrollo del caso práctico, es decir, la aplicación de la estrategia al problema real. En él se describe el objeto de estudio y el problema que se buscó solucionar con el desarrollo de este trabajo y se ilustra la aplicación de la estrategia para el diseño de casos de prueba. Por último, el Capítulo 4 presenta los resultados y las conclusiones obtenidas de este trabajo junto con las líneas de avance futuras para seguir asegurando la calidad de la aplicación de software.

# Capítulo 2: Marco teórico de la estrategia SaST

Como se mencionó en el capítulo anterior, una estrategia bien especificada debería integrar 3 aspectos, a saber: una especificación de proceso, una especificación de método y una base conceptual de dominio. Por lo tanto, en este capítulo se van a describir estos 3 aspectos para la estrategia SaST. Además, se incluye una descripción de algunos trabajos relacionados a estrategias de prueba y se indica en qué se diferencian éstos con la estrategia SaST.

## 2.1 Base conceptual que da soporte a la estrategia SaST

Para lograr una correcta comprensión de cualquier tema, es esencial conocer la semántica de cada uno de los términos que este involucra. Además, también es importante conocer cuáles son sus propiedades y relaciones con otros términos, como así también posibles restricciones del dominio. En otras palabras, es primordial que el tema a abordar cuente con una base conceptual robusta, tales como las ontologías.

En este sentido la estrategia SaST está soportada semánticamente por un conjunto de ontologías, las cuales pertenecen a una arquitectura ontológica denominada FCD-OntoArch [8], ilustrada en la Fig. 1. Ésta es una arquitectura ontológica de cinco capas, llamadas: *'foundational'*, *'core'*, '*top-domain*', '*low-domain*' e *'instance'*. Ontologías pertenecientes al mismo nivel pueden estar relacionadas entre sí, a excepción del nivel llamado '*foundational*' donde hay una única ontología llamada ThingFO [8]. Además, ontologías en niveles inferiores pueden ser enriquecidas semánticamente por ontologías de niveles superiores.

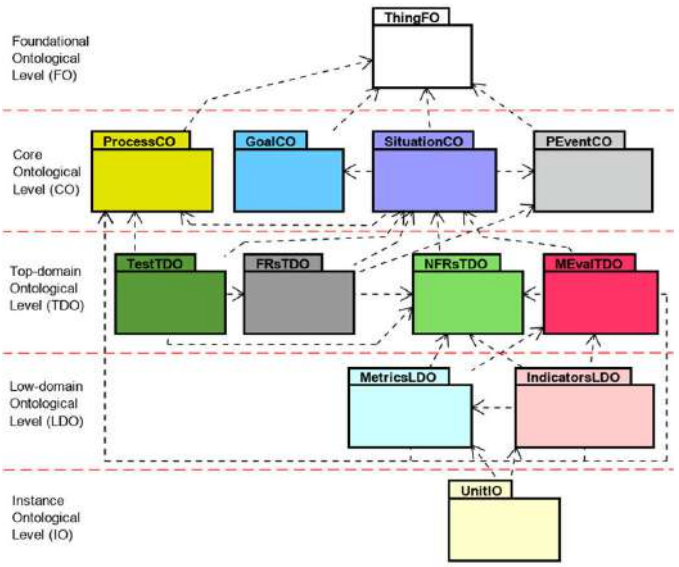



Fig. 1. Arquitectura ontológica de cinco capas llamada FCD-OntoArch. Imagen tomada de [8].

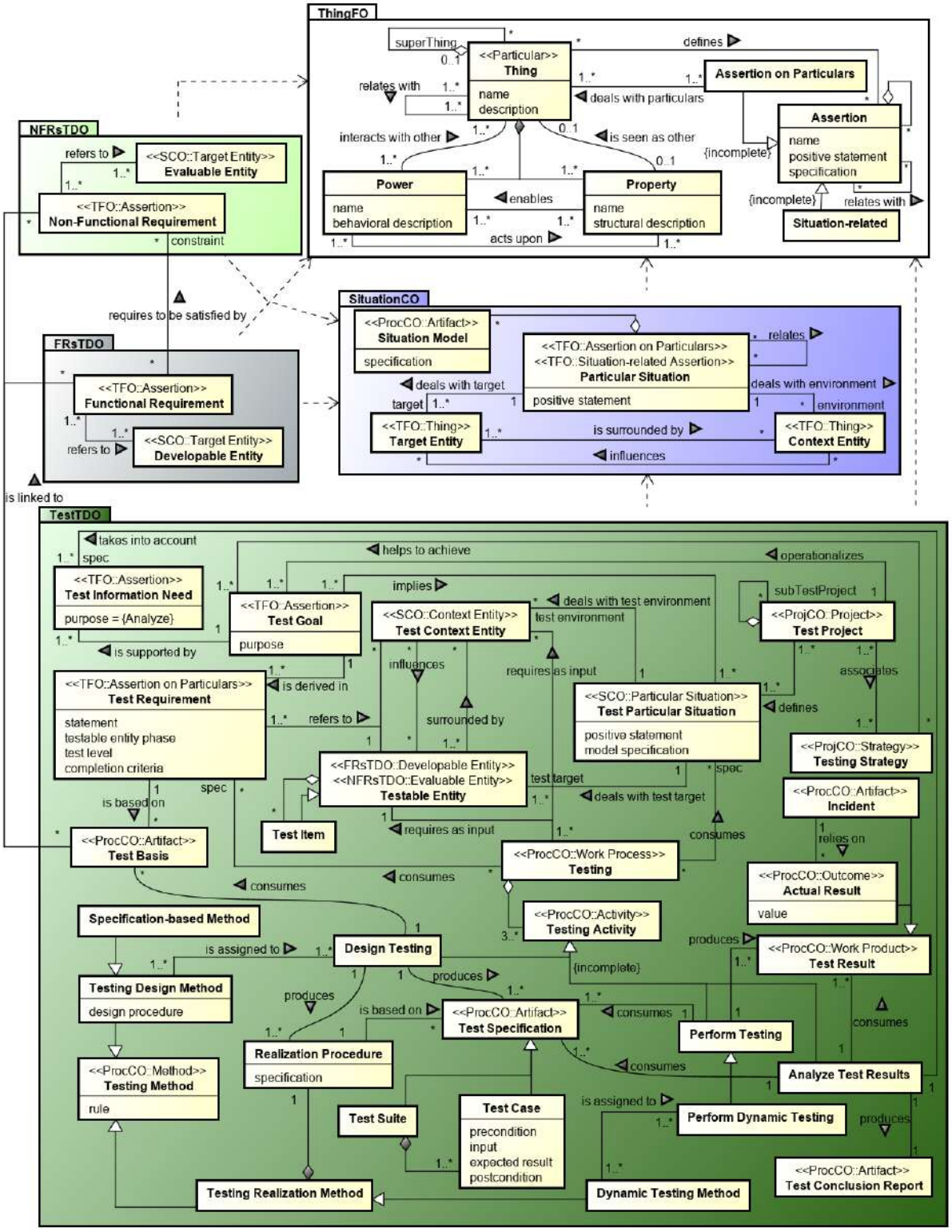



Fig. 2. Fragmento de algunas ontologías pertenecientes a FCD-OntoArch utilizadas en la estrategia SaST. Nótese que se utiliza la sigla TFO para Thing Foundational Ontology (ThingFO), SCO para Situation Core Ontology (SituationCO), ProcCO para Process Core Ontology, ProjCO para Project Core Ontology, NFRsTDO para Non-Functional Requirements Top-Domain Ontology y FRsTDO para Functional Requirements TDO.
Imagen tomada de [7].

Según Tebes *et al*. [7], la estrategia SaST hace uso de una ontología 'Top-domain' para pruebas de software llamada TestTDO [9] la cual enriquece semánticamente las especificaciones de método y de proceso de dicha estrategia. Además, en [7] los autores indican que una ontología '*core*' para situaciones llamada SituationCO [10] es muy importante para la estrategia SaST ya que enriquece semánticamente a TestTDO. Notar que SituationCO brinda los cimientos semánticos claves para modelar situaciones, las cuáles son un aspecto fundamental de la estrategia SaST.

Por otra parte, dado que la estrategia SaST está integrada por una especificación de proceso y una especificación de método, la ontología 'core' llamada ProcessCO [11] también es considerada importante para la estrategia. A continuación se hace un pequeño análisis de la arquitectura mencionada y del modo en que se relacionan semánticamente los distintos términos que la forman.

Algunos términos de ThingFO [8], tales como 'Thing' y 'Assertion' enriquecen términos en ontologías de niveles inferiores. Por ejemplo, como se muestra en la Fig. 2, los términos 'Target Entity' y 'Particular Situation' pertenecientes a SituationCO son enriquecidos por 'Thing' y 'Assertion on Particulars', respectivamente. 'Thing' representa un objeto particular, tangible o intangible, de un universo particular dado, pero no una categoría. Una Cosa ('Thing'), no es un objeto particular sin sus Propiedades ('Properties') y sus Capacidades ('Powers'), por lo tanto, este trío emerge simultáneamente para formar una unidad. Además una Afirmación ('Assertion') es una declaración explícita y positiva que alguien hace sobre algo relativo a una Cosa o sus categorías, basado en pensamientos, hechos, intuiciones y/o intenciones. Para tener valor científico, toda afirmación debe ser verificada y/o validada con evidencia teórica o empírica. Las afirmaciones pueden ser representadas y modeladas en especificaciones de lenguajes formales, semiformales o informales. El lector puede acceder a las definiciones de todos los conceptos de ThingFO en https://arxiv.org/abs/2107.09129.

SituationCO [10] incluye el término Situación Particular ('Particular Situation'), que es una Afirmación sobre Particulares relacionada a una Situación ('Situation-related Assertion on Particulars'), la cual indica de forma explícita y específica la combinación de circunstancias particulares, episodios y relaciones/eventos abarcando las Entidades Objetivo ('Target Entities') y las Entidades de Contexto ('Context Entities') que las rodean. Además, las Situaciones Particulares, o escenarios, pueden ser especificadas en un Modelo de Situación ('Situation Model'), que es un Artefacto ('Artifact' [11]). El lector puede encontrar todas las definiciones de los conceptos de SituationCO en https://arxiv.org/abs/2107.10083.

A continuación se describen algunos de los términos clave que enriquecen la estrategia SaST, los cuales forman parte de la ontología TestTDO. Notar que, para TestTDO, todas sus definiciones de términos, propiedades y relaciones no taxonómicas en conjunto con las especificaciones de los axiomas están documentados en https://arxiv.org/abs/2104.09232.

Como se puede ver en la Fig. 2, Un Proceso de Prueba (‘Testing Process’) está compuesto por al menos tres Actividades de Prueba (‘Testing Activities)’, denominadas: Diseñar la Prueba (‘Design Testing’), Ejecutar la Prueba (‘Perform Testing’) y Analizar los Resultados de Prueba (‘Analyze Test Results’). También se podrían considerar otras actividades, como las relacionadas a establecer un ambiente de prueba.

El objetivo de la actividad Diseñar la Prueba es producir Especificaciones de Prueba (‘Test Specifications’) como así también Procedimientos de Realización (‘Realization Procedures’). Las Especificaciones de Prueba pueden ser diseñadas utilizando las Bases de Prueba (‘Test Basis’), Requerimientos de Prueba (‘Test Requirements’) y/o Situaciones Particulares de Prueba (‘Test Particular Situations’). Las bases de prueba son toda aquella documentación que puede ser utilizada como insumo para diseñar las pruebas, como podrían ser las especificaciones de requerimientos, el código fuente de la aplicación de software, etc. Por su parte, los Requerimientos de Prueba indican qué debe ser verificado o validado de un Objeto de Prueba (‘Test Object’). Finalmente un Escenario de Prueba, tiene la semántica de ‘Particular Situation’ de SituationCO, y asocia la o las Entidades de Prueba (‘Testable Entity’) con una o más Entidades de Contexto de Prueba (‘Test Context Entities’). Las Entidades de Prueba forman parte del denominado Objeto principal de Prueba (‘Test Target’), mientras que las Entidades de Contexto de prueba forman parte del Ambiente de Prueba (‘Test Environment’), en caso de existir ya que es opcional. Dado que las distintas situaciones pueden ser modeladas en especificaciones formales, semiformales o informales, entonces un Escenario de Prueba puede ser especificado en escenarios de casos de uso, diagramas de comunicación, en lenguaje natural, entre otras representaciones.

La actividad Diseñar la Prueba tiene asignada un Método de Diseño de Prueba (‘Testing Design Method’). Un método basado en especificaciones, es decir, una técnica de prueba de caja negra, es un tipo de método de diseño de prueba, y un tipo más específico de ésta es, por ejemplo, un método de prueba basado en escenarios. Notar que un método de diseño de prueba tiene reglas y un procedimiento de diseño. Entendiendo por procedimiento de diseño a un conjunto ordenado de instrucciones u operaciones, que especifican cómo debe ser llevada a cabo cada subactividad/tarea de la actividad de diseño de prueba.

Un tipo de especificación de prueba es un Caso de Prueba (‘Test Case’). Este documento es utilizado principalmente en la actividad Ejecutar la Prueba Dinámica (‘Perform Dynamic Testing’) y contiene precondiciones (preconditions), entradas (inputs), resultados esperados (expected results) y postcondiciones (postconditions). Una precondición es cualquier tipo de restricción o condición que debe ser evaluada como verdadera antes de que la entrada del caso de prueba sea utilizada en la actividad Ejecutar la Prueba. En cambio, una postcondición es una restricción que debe ser evaluada como verdadera después de que la entrada haya sido utilizada y se haya obtenido el Resultado Real (‘Actual Result’).

## 2.2 Especificación de método para la estrategia SaST

El método de la estrategía SaST, denominado Situation-aware Scenario-based Testing Method (SaSTMe), brinda soporte para el diseño y documentación de casos de prueba, capturando información relevante sobre el escenario de prueba como pueden ser la o las entidades que serán el objetivo de la prueba (Test Targets) y las entidades de contexto que se relacionen con éstas (si las hubiese). A continuación, para ilustrar el soporte semántico que brindan las ontologías mencionadas anteriormente, se remarcan los términos provenientes de ontologías con palabras capitalizadas, propiedades en itálica, y relaciones no-taxonómicas subrayadas.

La Fig. 3 contiene la plantilla de SaSTMe, la misma apunta a capturar toda la información relevante de un escenario de prueba, de modo que, los diseñadores de pruebas deben documentar sus declaraciones positivas (*positive statement*) y sus especificaciones de modelos (*model specifications*), para cada escenario. Ambos pueden ser especificados en lenguaje natural, pero los modelos de situación (Situation Model) que describen el escenario tendrán más valor si son especificados formalmente mediante la utilización de algún diagrama estandarizado, como pueden ser los diagramas de secuencia o los diagramas de comunicación UML, entre otros.

SITUATION-AWARE SCENARIO-BASED TESTING METHOD (SASTME) TEMPLATE SPECIFICATION

| |
|---|
| **Test Particular Situation:**<br>*-positive statement*:<br>-(Particular) **Situation Model** *specification*: |
| **Testable Entities** (and **Test Items**, if applicable)**:** *indicate what are the test targets* |
| **Test Context Entities:** (not mandatory) |
| **Test Requirement:** |
| **Test Basis**: |
| **Test Cases**:<br>*-input*: *-expected result*:<br>*-preconditions*: *-postconditions*: |

Fig. 3. Plantilla de especificación de SaSTME.
Imagen tomada de [7].

Una vez definido el escenario de prueba (Test Particular Situations), el siguiente paso es analizar el mismo buscando identificar entidades a probar (Testable Entities) y entidades de contexto de prueba (Test Context Entities) que influencien (influence) a las anteriores.

Además, la plantilla captura los requerimientos de prueba (Test Requirements) junto con información de la base de prueba (Test Basis) que puede ser de utilidad para diseñar los resultados esperados (*expected results*) de los casos de prueba (Test Cases).

Finalmente, haciendo uso de la información incluída en la plantilla, los diseñadores de pruebas producirán casos de prueba (Test Cases). Es importante remarcar que, en un escenario de prueba, las entidades de contexto relacionadas pueden influenciar las condiciones (*conditions*) y las entradas (*inputs*) de los casos de prueba [12]. Por lo tanto, los diseñadores de pruebas deben analizar cada modelo de situación (Situation Model) en particular para diseñar las entradas (*inputs*) y/o condiciones (*conditions*) de los casos de prueba (Test Cases) que serán de utilidad para verificar o validar de forma efectiva la o las entidades de prueba (Testable Entities) en función a las entidades de contexto de prueba (Test Context Entities) que lo rodean (surrounding).

## 2.3 Especificación de proceso para la estrategia SaST

Como se muestra en la Fig. 4, se utiliza el lenguaje SPEM (*Software & Systems Process Engineering Metamodel*) [13] para especificar el proceso de la estrategia SaST, denominado SaSTPro (*Situation-aware Scenario-based Testing Process*).

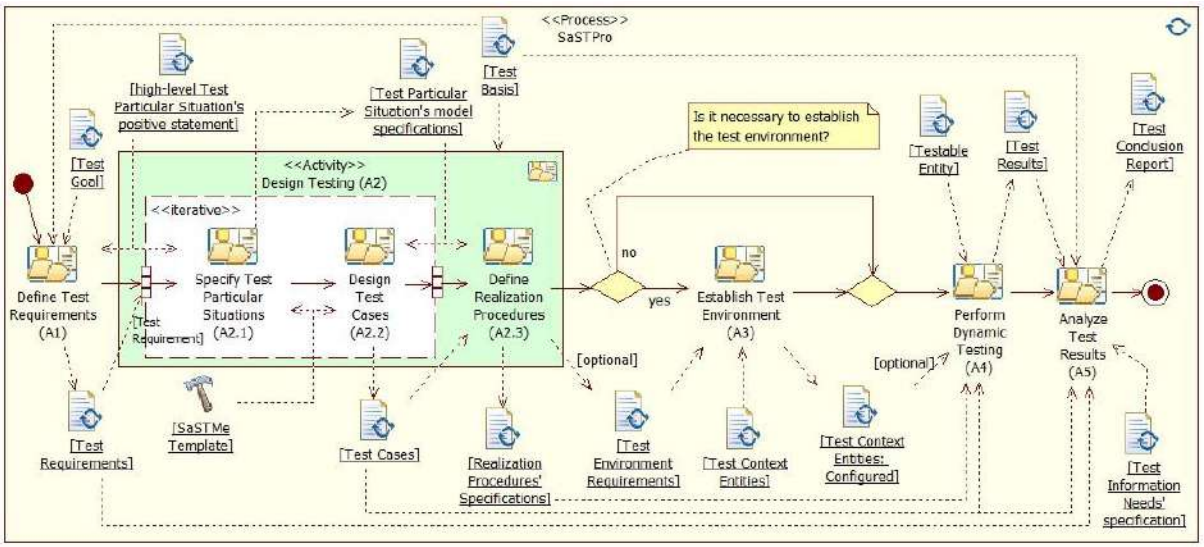



Fig. 4. Especificación de las perspectivas funcional y de comportamiento de SaSTPro en lenguaje SPEM. Imagen tomada de [7].

A continuación se describen las principales actividades de SaSTPro junto con los artefactos que las mismas producen/consumen. Notar que la semántica de los términos de proceso tales como actividad (activity), artefacto (artifact), entre otros, puede encontrarse en la ontología denominada Process Core Ontology (ProcessCO) [11].

Según el proceso de la Fig. 4, la primera actividad a ejecutar es Definir los Requerimientos de Prueba (“Define Test Requirements (A1)”). Esta actividad consume dos documentos que pueden provenir del proyecto de prueba (‘Test Project’): Objetivo de Prueba (‘Test Goal’) y Declaración positiva de Escenario de Prueba (‘Test Particular Situation’s *positive statement’*). Además, también se pueden usar las Bases de Prueba (‘Test Basis’) como especificaciones de requisitos funcionales y no funcionales. El objetivo de A1 es producir requerimientos de prueba (‘Test Requirements’), que declaran qué debe ser verificado/validado de una entidad de prueba (‘Testable Entity’).

La siguiente actividad que se debe llevar a cabo es Diseñar la Prueba (“Design Testing (A2)”) la cual está formada por tres sub-actividades. La primera de ellas es Especificar Escenarios de Prueba (“Specify Test Particular Situations (A2.1)”), en la cual se deben producir modelos de escenario o situación (‘Situation Models’) haciendo uso de la (‘Test Basis’). Recordar que, estos modelos de escenario pueden estar especificados en, por ejemplo, diagramas UML de comunicación, secuencia, casos de uso, entre otros. También es importante volver a resaltar que un modelo de escenario asocia la o las entidades a probar (‘Testable Entities’) con las entidades de contexto (‘Context Entities’), según corresponda. Finalmente, toda la información producida es registrada según la plantilla de especificación de SaSTMe (Fig. 3).

Una vez desarrollados los modelos de escenario, el siguiente paso es producir los casos de prueba (‘Test Cases’) en la actividad Diseñar los Casos de Prueba (“Design Test Cases (A2.2)”). Notar que un caso de prueba puede tener entradas (‘*inputs*’), precondiciones (‘*preconditions’*), postcondiciones (‘*postconditions*’) y un resultado esperado (‘*expected result’*). Estos elementos, para cada caso de prueba, son diseñados analizando los modelos de escenario y se registran utilizando la plantilla de SaSTMe. También es importante considerar la base de prueba (‘Test Basis’), ya que generalmente es útil para establecer los resultados esperados. A este punto es importante remarcar que las actividades A2.1 y A2.2, es decir, la especificación de escenarios de prueba y el diseño de los casos de prueba, pueden ser desarrolladas de forma iterativa para cada requerimiento de prueba (‘Test Requirement’).

La siguiente subactividad es Definir los Procedimientos de Realización (“Define Realization Procedures (A2.3)”). Esta actividad utiliza toda la información presente en la plantilla de especificación de SaSTMe (modelos de escenarios de prueba y los casos de prueba) para producir especificaciones de procedimientos de realización (‘Realization Procedures’ *specifications*). Por ejemplo, cuando se llevan a cabo pruebas dinámicas que involucran casos de prueba, una especificación de procedimiento de realización puede ser una secuencia de casos de prueba en un cierto orden de ejecución y cualquier otra acción asociada que pueda ser requerida para cumplir con las precondiciones (‘*preconditions*’) y/o postcondiciones (‘postconditions’). Además, en esta actividad se pueden identificar

requerimientos del entorno o ambiente de prueba (‘Test Environment Requirements’), en caso que sea necesario establecer uno.

Los requerimientos del entorno de prueba y las entidades de contexto de prueba son consideradas en la actividad Establecer el Entorno de Prueba (“Establish Test Environment (A3)”) con el objetivo de configurar el entorno donde finalmente se ejecutarán las pruebas diseñadas. Notar que la actividad A3 es una actividad opcional.

Una vez finalizada la actividad A3, o en caso de no ser necesario el establecimiento de un entorno de prueba, la siguiente actividad a desarrollar es Ejecutar las Pruebas Dinámicas (“Perform Dynamic Testing (A4)”). En esta, los procedimientos de realización (‘Realization Procedures’), definidos en A2.3, son ejecutados haciendo uso de las entidades de prueba (‘Testable Entities’) y, opcionalmente, las entidades de contexto de prueba (‘Test Context Entities’). También pueden ser utilizadas en esta actividad las especificaciones de casos de prueba, principalmente cuando se llevan a cabo pruebas manuales. Como resultado, A4 produce resultados de prueba (‘Test Results’), los cuales pueden ser los resultados reales obtenidos al ejecutar la prueba (‘Actual Results’), o bien incidentes (‘Incidents’).

Finalmente, se ejecuta la actividad Analizar los Resultados de Prueba (“Analyze Test Results (A5)”), donde los resultados de prueba son analizados con el fin de determinar si se cumplieron los requerimientos de prueba (‘Test Requirements’) y fueron logradas las Metas de Necesidad de Información de Prueba (‘Test Information Need Goals’). Este tipo de metas u objetivos proveen información de utilidad a la hora de decidir si el objetivo de prueba (‘Test Goal’) fue logrado o no. Finalizando el proceso de SaSTPro, el análisis es documentado en el informe o reporte de conclusión de prueba (‘Test Conclusion Report’).

Para finalizar la presentación del proceso de la estrategia SaST, es importante remarcar que la descripción de las actividades que lo conforman fueron mostradas en una forma puramente secuencial con el objetivo de mantener la legibilidad, a excepción de las subactividades A2.1 y A2.2 que fueron indicadas como iterativas por cada requerimiento de prueba. En la práctica, otras actividades también pueden ser iterables, por ejemplo, en caso de que sea necesario contemplar más escenarios de prueba para llegar a satisfacer un requerimiento de prueba inconcluso, las actividades A2 a A5 pueden ser iteradas hasta lograrlo. Además, como se mencionó en el capítulo anterior, el alcance de esta tesina considera sólo la ejecución de las actividades A1, A2.1 y A2.2

## 2.4 Trabajos relacionados

Pocos trabajos se han encontrado que documenten estrategias que sean de utilidad para verificar sistemas de software y den soporte para la documentación de entornos de prueba. Por ejemplo, en [5], los autores indican que la mayoría de las metodologías de pruebas de software fueron desarrolladas con el objetivo de probar el software de forma convencional (es decir, sin considerar entidades de contexto) y pocos enfoques fueron diseñados para el propósito de probar sistemas "conscientes del contexto".

Un trabajo reciente relacionado está documentado en [12]. En este trabajo, los autores proponen CATS#, que es una evolución de la técnica de diseño CATS (Context-Aware Test Suite). Su técnica está destinada a ayudar a los ingenieros de software con la especificación de casos de prueba sensibles al contexto. Al utilizar esta técnica, los casos de prueba pueden capturar el contexto en sí y también su variación. Esta técnica de diseño de pruebas consciente del contexto puede tener un impacto práctico, ya que también modela explícitamente tres escenarios en los cuales el contexto puede influir en las entradas de los casos de prueba, sus condiciones o en ambos. Sin embargo, el trabajo no modela la situación como la estrategia SaST, ya que CATS# no captura explícitamente cuáles son las entidades de contexto que intervienen en un escenario sino que más bien solo captura sus influencias.

A su vez, otros trabajos similares donde se apliquen estrategias de prueba son [14-18]. A diferencia del tipo de aplicación a probar en el contexto de Agripay, donde tanto las entidades a probar como las entidades de contexto son todas categorizadas como software, los trabajos documentados en [14-16] se centran en las pruebas de software que controlan el trabajo automático de automotores y los documentados en [17-18] pertenecen al dominio del Internet de las Cosas (IoT), por lo que las entidades de contexto pertenecen más bien al mundo del hardware, como pueden ser sensores de temperatura, presión, geolocalización, etc.

# Capítulo 3: Aplicación de las actividades de diseño de la estrategia SaST

En este capítulo se llevará a cabo la aplicación práctica de la estrategia SaST para el diseño de casos de prueba para una Web App real que interactúa con varios servicios.

En la Sección 3.1 se hace una breve presentación de la Web App que se está desarrollando en Agripay y los Servicios de la API que fueron seleccionados como objetos de prueba. Por su parte, en la Sección 3.2 se describe el conjunto de documentos que conforman la Test Basis considerada para este trabajo, la cuál será de gran utilidad en la siguiente sección al momento de definir los distintos escenarios y casos de prueba. Finalmente, en la Sección 3.3, se definen los requerimientos de prueba, se plantean los distintos escenarios para cada uno de ellos y se hace el diseño de los casos de prueba.

## 3.1 Presentación de los objetos de prueba y entidades de contexto

Actualmente, en la empresa Agripay está en desarrollo un proyecto el cual se basa en la creación de una plataforma para gestión de contratos inteligentes y pagos digitales. Dicho desarrollo abarca tres frentes distintos: Web App, App Mobile y API backend. Por la velocidad de crecimiento de la API y lo ajustado de los tiempos de desarrollo, es imperativa la realización de pruebas de integración que ayuden a validar y verificar el correcto funcionamiento en conjunto de los sistemas frontend (Web App y App Mobile) y la API backend. Particularmente, como miembro del equipo de desarrollo de la Web App, me centré en la integración de la aplicación web con la API backend.

La API brinda servicios accesibles desde el frontend mediante distintos endpoints. Notar que, un endpoint es el path o url mediante la cual se hace la petición al servicio. Estos servicios son consumidos por las aplicaciones frontend para obtención y escritura de datos sobre una base de datos compartida. Teniendo en cuenta esto, es de primera necesidad asegurar que las interacciones con la API sean correctas, o dicho de otra forma, que durante la ejecución de alguna funcionalidad en particular de las distintas aplicaciones frontend, las respuestas obtenidas de las peticiones a la API backend sean las esperadas por el usuario.

Como se mencionó, las pruebas se diseñaron para encontrar defectos en la interacción entre la Web App y la API backend. Además, considerando que el alcance del presente trabajo se limita al diseño de los casos de prueba, se pondrá especial énfasis en la

Actividad A2 de la estrategia SaST (recordar Fig. 4). Finalmente, por temas de confidencialidad del proyecto de Agripay, los datos utilizados en la aplicación práctica de la estrategia fueron modificados a fin de no mostrar información sensible para la empresa, como pueden ser por ejemplo, las urls reales de los endpoints de la API.

En el caso particular de este trabajo, el objeto de estudio no es un ente individual sino más bien un conjunto de entidades. Como el objetivo es verificar el funcionamiento conjunto de la Web App y la API backend, los objetos de prueba serán un conjunto formado por un componente de la Web App y uno o más servicios de la API backend. Con motivo de agilizar la lectura, se utilizarán las siglas WC y AS para hacer referencia a 'Web Component' (componente de la Web App) y 'Api Service' (servicio de la API backend), respectivamente.

Como objeto de prueba seleccionado se tiene, por un lado, el WC listar-contratos, cuya funcionalidad es la de mostrar por pantalla un listado de los contratos de una empresa consumiendo para ello el AS listar-contratos que genera dicho listado consultando una base de datos externa. Notar que el AS listar-contratos también es un objeto a ser probado. La interfaz del componente web proporciona una botonera que permite realizar distintas acciones sobre un contrato, como puede ser ver el detalle del mismo, rechazarlo o firmarlo. También incluye un panel desplegable que permite utilizar filtros para obtener un conjunto de contratos en un cierto rango de fechas, contratos firmados o no firmados, entre otros. Por ejemplo, utilizando estos filtros se puede indicar al WC que se quiere obtener todos los contratos firmados que sean posteriores al 2018 y hasta la actualidad.

A su vez, otro objeto de prueba es el WC firmar-contratos, el cual, como su nombre lo indica, permite firmar digitalmente un contrato. El mismo posee una interfaz de vista previa del documento del contrato (en pdf) para que el usuario pueda llevar a cabo su lectura y brinda un botón para iniciar el proceso general de firma. Luego, para poder cumplir su función, consume los servicios AS enviar-sms, el cual genera un segundo factor de autenticación al momento de realizar la firma mediante un servicio externo de envío de mensajes de texto. Además del AS enviar-sms se necesita consumir el AS firmar-contratos para realizar la firma digital del contrato. Notar que el AS firmar-contratos, el cual es un objeto de prueba, utiliza un servicio externo para generar un estampado blockchain para la trazabilidad del contrato. En este punto, es importante mencionar que tanto el AS enviar-sms junto servicio externo que éste utiliza, como así también, el servicio utilizado para generar la estampa blockchain por parte del AS firmar-contratos, son todos objetos de contexto.

## 3.2 Definición de la Test Basis

Como bien lo indica [7], es de mucha utilidad para gran cantidad de las actividades que conforman a SaSTPro (ver Fig. 4), tomar en consideración lo indicado en la ‘Test Basis’, además de los otros documentos que puedan consumir cada una de las actividades.

Como se mencionó en el Capítulo 2, se conoce como ‘Test Basis’ a toda aquella documentación que puede ser utilizada como insumo para diseñar las pruebas, como podrían ser las especificaciones de requerimientos, el código fuente de la aplicación de software, etc. En otras palabras, es la base de conocimiento relativa al objeto de prueba, es decir, toda aquella documentación que pueda ser utilizada como referencia a la hora de lograr una buena interpretación del mismo. Por lo tanto, la ‘Test Basis’ es de utilidad para el diseño de escenarios y/o casos de prueba que prueben el comportamiento esperado del objeto de prueba en una cierta situación.

Para la definición de la ‘Test Basis’, el primer paso fue hacer un análisis de la aplicación Web de Agripay, identificando y documentando los componentes de la misma junto con los servicios de la API que cada uno consume. Luego, junto al líder del equipo de desarrollo fontend de Agripay, se diseñó el esquema de impacto mostrado en la Tabla 1. En el mismo se muestra una clasificación de los WC según su grado de impacto sobre el funcionamiento final de la aplicación. Además se indica para cada tipo de clasificación una exhaustividad de prueba, la cual describe textualmente la profundidad de análisis que cada clasificación amerita y un listado de las distintas respuestas de la API que se consideran como suficientes para probar un componente con ese nivel de impacto.

| Impacto | | | | | |
|---|---|---|---|---|---|
| # | 1 | 2 | 3 | 4 | 5 |
| Nombre | Bajo | Medio | Alto | Muy Alto | Crítico |
| Descripción | *Componente prescindible*: No compromete la experiencia de usuario | *Componente soporte*: No compromete la experiencia de usuario directamente, aunque puede comprometer el funcionamiento de otros componentes | Componente de funcionalidad: Compromete de forma directa la experiencia de usuario | *Componente de negocio*: Es una funcionalidad clave del producto | *Componete de aplicación*: Compromete directamete la utilización de la aplicación |
| Respuestas de la API a considerar | | | | | |
| 200 | NO | SI | SI | SI | SI |
| 403 | NO | NO | NO | SI | SI |
| 404 | NO | NO | Solo si hay parámetros en la url | Solo si hay parámetros en la url | SI |
| 405 | NO | NO | NO | NO | SI |
| 422 | NO | NO | NO | NO | SI |
| Errores sin código provenientes de servicios externos | NO | NO | NO | SI | SI |

Tabla 1. Esquema de clasificación según el nivel de impacto para los componentes de la Web App de Agripay. Además, se muestran las respuestas de la API a considerar en las pruebas según el nivel de impacto.

Algunas respuestas de la API que serán tomadas a consideración son por ejemplo las de los códigos 200, 403 y 404. Las respuestas con códigos 200 indican una respuesta exitosa, en donde la API pudo resolver exitosamente la petición al servicio. Por otro lado, las respuestas con códigos 403 y 404 indican casos de error. Más específicamente, un código

403 indica un error por falta de permisos al momento de ejecutar una acción, y un código 404 indica un error en el procesamiento de una ruta por haber proporcionado un endpoint inexistente o con errores en los parámetros que lo forman.

Un listado completo de todas las posibles respuestas de la API quedan a disposición del lector en el Anexo I: Test Basis. No todas las respuestas posibles indicadas en el anexo se ven reflejadas en la tabla de impacto por considerarlas ajenas al objetivo o fuera del alcance de este proyecto. Un ejemplo de esto pueden ser los errores de conexión que puedan darse por una falla en los servicios de comunicación, estos fueron dejados de lado por considerarlos ajenos al objetivo de prueba. Como así también, los errores con código 500, los cuales indican un error interno en la API, que fueron descartados por dos razones. Primero por ser inviables en un entorno de prueba ya que para lograr su replicación habría que modificar uno de los objetos a probar (en este caso la API), lo cual restaría valor a las pruebas a diseñar. Y segundo por ser internos de la misma, es decir específicos a ella, lo cual los relaciona más a pruebas unitarias de la API que a las pruebas de integración que se busca diseñar.

Teniendo en cuenta la descripción de los objetos de prueba seleccionados que se hizo en la Sección 3.1, contando nuevamente con la ayuda del líder del equipo de desarrollo Web y en función a lo descrito en la Tabla 1, se categorizaron los objetos de prueba seleccionados según su nivel de impacto (ver Tabla 2).

| Componentes de la Web App Front End (WC) | | | |
|---|---|---|---|
| NOMBRE | SERVICIOS API | DESCRIPCIÓN | IMPACTO |
| WC listar-contratos | AS listar-contratos | Muestra una lista de los contratos de un usuario en relación a una empresa y presenta una botonera de acciones posibles a realizar sobre una instancia de contrato. | Alto (3) |
| WC firmar-contrato | AS firmar-contratos<br>AS enviar-sms | Hace una vista previa del documento pdf del contrato a firmar y muestra un botón que da la opción al usuario de firmar el contrato. | Muy Alto (4) |

Tabla 2. Descripción de los componentes de la Web App sobre los que se aplicará la estrategia. Además, se muestran los servicios de la API relacionados y el nivel de impacto asignado a cada uno de ellos. Los niveles de impacto fueron descritos en la Tabla 1.

Además, en la Tabla 2, se hace una recopilación de la información relacionada a los componentes de la aplicación Web que serán tomados como objetos principales de prueba (‘Test Targets’). En ella se puede ver, por un lado, el WC listar-contratos, el cual muestra

una lista de los contratos de un usuario en relación a una empresa, consumiendo para ello el AS listar-contratos. Fue categorizado con un nivel de impacto alto (3) por ser un componente de funcionalidad, lo que quiere decir que, un fallo en el mismo compromete notablemente la experiencia de usuario.

A su vez, la Tabla 2 muestra el grado de impacto del WC firmar-contrato, el cual tiene un nivel de impacto muy alto (4). Recordar que este componente tiene como función permitir a los usuarios finales firmar los contratos, para lo cual debe generar una vista previa del documento pdf del contrato y un botón para iniciar el proceso de firma digital. Además, para poder cumplir su función, consume los servicios AS firmar-contratos y AS enviar-sms. El WC firmar-contrato fue categorizado con un nivel de impacto muy alto (4) por ser considerado un componente de negocio, es decir, la funcionalidad que el componente firmar-contrato brinda a la Web App no es secundario o de apoyo, sino uno de los requerimientos funcionales del producto final.

Por su parte, en la Tabla 3, se encuentra la información relacionada a los servicios de la API que, junto al WC que los consume, podrán formar parte de los objetos principales de prueba (‘Test Targets’), o bien ser considerados entidades de contexto (‘Text Context Entity’), según el papel que cumplan en cada interacción.

| Servicios de la API Back End (AS) | | | |
|---|---|---|---|
| NOMBRE | ENDPOINT | DESCRIPCIÓN | AGENTES EXTERNOS |
| AS listar-contratos | base-url/users/**user_id**/companies/**company_id**/contracts | GET: Obtiene el listado de contratos vinculados a una empresa y un usuario en particular (puede contener parámetros de filtrado de resultados tales como rango de fecha, empresas emisora del contrato, entre otros) | - |
| AS firmar-contrato | base-url/users/**user_id**/companies/**company_id**/contracts/**contract_id**/sign | POST: Marca como firmado un contrato digitalmente, requiere como parámetro un código de verificación enviado por sms. | Servicio de estampado Blockchain |
| AS enviar-sms | /base-url/users/**user_id**/send-sms | POST: Envía un código de verificación sms al número de teléfono del usuario indicado por url. Funcionalidad complementaria como segundo factor de autorización en muchos servicios de la API. | Servicio de envío de SMS |

Tabla 3. Descripción de los servicios de la API backend y sus endpoints asociados. Nótese que se marcan en negrita los parámetros de URL, los cuales se deben instanciar con datos al momento de su utilización.

Observar que, para cada servicio, se documenta su endpoint de acceso, una breve descripción del método HTTP necesario para comunicarse con él y un listado de los agentes externos con los que se comunica. Por ejemplo, el servicio AS listar-contratos, tiene como endpoint “base-url/users/user_id/companies/company_id/contracts”, el método HTTP que utiliza es el GET para obtener el listado de los contratos vinculados a la empresa y al

usuario indicados en los parámetros de ruta company_id y user_id respectivamente y para ello no se comunica con ningún agente externo.

Finalmente, al contar con un entorno de prueba preexistente, se tomaron algunos datos de la base de datos de prueba de dicho entorno, que serán de utilidad en el capítulo siguiente. Con ellos se llevó a cabo una descripción informal simplificada a la cual se le dió el nombre de 'Test Data Pool' (Fig. 5). En él se pueden identificar datos de usuarios ('Users'), empresas ('Companies') y relaciones entre los mismos ('Companies X Users'); datos de permisos ('Permissions') y los permisos asignados a los distintos usuarios ('Permissions X Users') y datos de contratos ('Contracts').

**Users**

| *user_id* | *name* | *surname* | *email* | *phone-number* |
|---|---|---|---|---|
| 1 | Jhon | Doe | jhondoe@example.net | 230251895 |
| 2 | Anna | Flick | annaf@example.net | 2302889512 |

**Companies**

| *company_id* | *business name* |
|---|---|
| 1 | Test S.A |
| 2 | Prueba S.R.L |

**Companies x Users**

| *company_id* | *user_id* |
|---|---|
| 1 | 1 |
| 2 | 2 |

**Permissions**

| *permission_id* | *name* |
|---|---|
| 1 | list-contracts |
| 2 | sign-contracts |

**Permissions x Users**

| *permission_id* | *user_id* |
|---|---|
| 1 | 1 |
| 1 | 2 |
| 2 | 1 |

**Contracts**

| *contract_id* | *company_id* | *contract_type* | *date* | *signed* |
|---|---|---|---|---|
| 1 | 1 | 1 | 06/03/2021 | 0 |
| 2 | 1 | 3 | 04/04/2021 | 1 |
| 3 | 2 | 3 | 11/10/2021 | 0 |
| 4 | 2 | 2 | 20/02/2021 | 0 |
| 5 | 1 | 4 | 14/07/2021 | 1 |

Fig. 5. Test Data Pool: datos extraídos de la base de datos de prueba preexistente.

Un resumen de lo mencionado hasta el momento, indicaría que la Test Basis considerada para ésta aplicación de la estrategia SaST queda conformada por los siguientes artefactos:

- El esquema de clasificación de los WC según su nivel de impacto mostrado en la Tabla 1.
- Las respuestas de la API a considerar para cada nivel de impacto, presentes también en la Tabla 1.

- Las descripciones generales de los WC y AS, documentados en las Tablas 2 y 3.
- El extracto de la base de datos de prueba que conforma el ‘Test Data Pool’, presente en la Fig. 5.

## 3.3 Diseño de los casos de prueba utilizando SaST

En esta sección se hace uso de las actividades descritas en la especificación de proceso de SaST (recordar Fig. 4) con el objetivo de diseñar las pruebas. En la Subsección 3.3.1 se presentan los documentos que SaSTPro indica como necesarios para la ejecución de la actividad A1, para luego, ver cómo éstos son utilizados en la producción de los documentos donde se formalizan los Requerimientos de prueba (‘Test Requirements’). A continuación, en la Subsección 3.3.2, se dispone un modelo de escenario genérico descrito con un diagrama de comunicación UML con el fin de presentar al lector la forma en que se especificarán los escenarios. Luego, se indican nuevamente los documentos requeridos y la forma en que estos fueron utilizados, en este caso para el desarrollo de las subactividades que componen la actividad A2. En esta subsección se puede ver como se iteran las subactividades A2.1 y A2.2 para la producción de los documentos correspondientes a Especificaciones de modelo de escenario (‘Test Particular Situation's model specification’) y Casos de Prueba (‘Test Cases’).

### 3.3.1 Definir los Requerimientos de Prueba (A1)

Para realizar la primera actividad de SaSTPro, Definir los Requerimientos de Prueba (“Define Test Requirements (A1)” en Fig. 4), se comenzó por analizar el objetivo de prueba (ver Tabla 4) y la declaración del escenario (ver Tabla 5). Notar que ambos documentos fueron producidos de antemano junto al líder del equipo de desarrollo frontend web.

| **Objetivo de prueba (‘Test Goal’)** |
| --- |
| Verificar el correcto funcionamiento, de forma integrada, de los componentes de la Web App frontend y los servicios de la API backend que cada uno de ellos consume. |

Tabla 4. Documento de SaSTPro: Test Goal.

| **Declaración de la situación particular (‘High-level Test Particular Situation’s positive statement’)** |
| --- |
| Los componentes Web App frontend de listado y firma de contrato junto con los servicios relacionados de la API backend deberían funcionar correctamente de forma integrada. |

Tabla 5. Documento de SaSTPro: High-level Test Particular Situation’s positive statement.

Se puede observar, en la declaración del escenario ilustrada en la Tabla 5, que se deben verificar los componentes WC listar-contratos y firmar-contrato de manera integrada con los servicios que consumen. Entonces, considerando lo anterior, se decidió producir dos requerimientos de prueba (‘Test Requirements’). El primero, enfocado en verificar la correctitud funcional del WC listar-contratos en relación a su interacción con el AS listar-contratos, se lo llamó “TR-Integration-ListContracts”. El segundo, con el objetivo de verificar la correctitud funcional del WC firmar-contratos en relación a su interacción con el AS firmar-contratos, se lo denominó “TR-Integration-SignContract”. Además, tanto el objetivo de prueba (Tabla 4) como la declaración del escenario (Tabla 5), sugieren que el nivel de prueba (test level) a considerar en ambos requisitos es de integración. A su vez, es importante mencionar que todos los objetos a ser probados están actualmente en la fase de desarrollo del ciclo de vida del software.

Para el primer requisito (TR-Integration-ListContracts), se produjo la siguiente declaración (‘Statement’): “Se requiere verificar la correctitud funcional del WC listar-contratos en relación a su interacción con el AS listar-contratos”. Como se mencionó, el test level para este requisito es ‘integración’. Luego, para establecer el ‘Completion Criteria’, se consideró el contenido de las Tablas 1 y 2 (ambos artefactos con semántica de ‘Test Basis’). A partir de esto, se identificó que el WC listar-contratos tiene un nivel de impacto alto (3) y que para dicho impacto deben ser consideradas las respuestas de la API de código 200 y 404 (este último ya que el AS listar-contratos posee parámetros en la url) y con ello, se estableció el ‘Completion Criteria’ para ese requerimiento indicando que se debe diseñar al menos un caso de prueba para una respuesta con código 200 y otro para una respuesta con código 404. Para este requisito se identificaron como entidades a probar al WC listar-contratos y al AS listar-contratos. Finalmente tomando lo descrito en la Tabla 3 (nuevamente con semántica de ‘Test Basis’), se detectó que la única entidad de contexto es la base de datos de prueba.

Por otra parte, para el segundo requisito (TR-Integration-SignContract), la declaración (‘Statement’) producida es la siguiente: “Se requiere verificar la correctitud funcional del WC firmar-contratos en relación a su interacción con el AS firmar-contratos”. Nuevamente, el test level para este requisito es ‘integración’. De forma análoga al anterior, para establecer el ‘Completion Criteria’, se tomaron en cuenta los datos presentes en las Tablas 1 y 2. En base a esto se identificó que el WC firmar-contratos tiene un nivel de impacto muy alto (4) y que para dicho impacto se deben considerar las respuestas de la API con códigos 200, 403, 404 (éste nuevamente por tener el AS parámetros en la ruta del endpoint), y errores provenientes de servicios externos. Luego, utilizando la Tabla 3 para identificar los servicios externos con los que se comunica el AS firmar-contratos, se estableció el ‘Completion Criteria’ con la premisa: “Se debe diseñar al menos un caso de prueba para respuestas de la API con códigos 200, 403, 404; al menos un caso para un error en el servicio externo de

SMS y otro para un error en el servicio externo de estampado blockchain. Se tomaron como entidades a probar para este requisito al WC firmar-contratos y el AS firmar-contratos. Además, considerando la Tabla 3, se detectó que las entidades de contexto son el AS enviar-sms, el servicio externo de envío de SMS, el servicio externo de estampado blockchain y la base de datos de prueba.

Finalmente, el lector puede ver en la Tabla 6 toda la información de los dos requisitos de prueba producidos, la cual fue estructurada siguiendo el modelo de TestTDO (ver Test Requirement en Fig. 2).

**Label**: TR-Integration-ListContracts.
**Statement**: Se requiere verificar la correctitud funcional del **WC listar-contratos** en relación a su interacción con el **AS listar-contratos**.
**Testable entity phase**: Desarrollo.
**Test level**: Integración.
**Completion Criteria**: Se debe desarrollar al menos un caso de prueba para cada una de las siguientes situaciones:

- El servicio de la API respondió correctamente y el WC recibe un código 200 acompañado de un mensaje 'Success' y la lista de contratos filtrada y ordenada según lo esperado.
- Sucedió un error de parámetros en la URL, el WC recibe un código de error 404 acompañado de un mensaje 'Param Error.'

**Refers to Testable Entities**: WC listar-contratos, AS listar-contratos.
**Refers to Test Context Entities**: Test Database (Test DB)

---

**Label**: TR-Integration-SignContract
**Statement**: Se requiere verificar la correctitud funcional del **WC firmar-contratos** en relación a su interacción con el **AS firmar-contratos**.
**Testable entity phase**: Desarrollo
**Test level**: Integración
**Completion Criteria**: Se debe desarrollar al menos un caso de prueba para cada una de las siguientes situaciones:

- El servicio de la API respondió correctamente y el WC recibe un código 200 acompañado de un mensaje 'Success' y el contrato seleccionado figura como correctamente firmado.
- Sucedió un error de parámetros en la URL, el WC recibe un código de error 404 acompañado de un mensaje 'Param Error'.
- El usuario no tiene el permiso para firmar el contrato, el WC recibe un código de error 403 acompañado de un mensaje 'Authorization Error: Permission not granted'.
- Falló la entidad de contexto que hace el envío de SMS, el WC recibe un error sin código con el mensaje 'SMS sending failed'.

| |
|---|
| • Falló la entidad de contexto que hace el estampado blockchain para la trazabilidad de la firma, el WC recibe un error sin código con el mensaje ‘Stamping failed, sign cancelled’.<br><br>**Refers to Testable Entities**: WC firmar-contrato, AS firmar-contrato.<br>**Refers to Test Context Entities**: AS enviar-sms, Servicio externo de envío de SMS, Servicio externo de estampado blockchain, Test DB |

Tabla 6. Documento de SaSTPro: Requerimientos de prueba (‘Test Requirements’).

### 3.3.2 Diseñar la Prueba (A2)

La actividad A2, Diseñar la Prueba (‘Design Testing’), propone primero el desarrollo iterativo de las subactividades A2.1, Especificar las Situaciones Particulares de Prueba (‘Specify Test Particular Situations’), y A2.2, Diseñar los Casos de Prueba (‘Design Test Cases’), para cada ‘Test Requirement’, con el objetivo de diseñar los escenarios y cada uno de los casos de prueba asociados a cada uno de ellos. Luego, finaliza con la subactividad A2.3, Definir Procedimientos de Realización (‘Define Realization Procedure’), en la cual se especifica el modo en que las pruebas deben de ejecutarse.

Como se muestra en la Fig. 4, la subactividad A2.1 produce un modelo de ‘situación particular’ o ‘escenario’, el cual es conveniente que se especifique en un lenguaje formal [7]. Por ello, para este trabajo se decidió utilizar diagramas de comunicación UML para modelar los escenarios. Se optó por utilizar diagramas de comunicación y no otro, como podrían ser diagramas de secuencia, ya que se los consideró más fáciles de entender a la hora de comunicar una situación particular. A modo de ejemplo, en la Fig. 6 se plantea un modelo de escenario genérico que contiene los diferentes elementos que luego se utilizarán en los distintos modelos producidos en A2.1, con el motivo de facilitar al lector su interpretación.

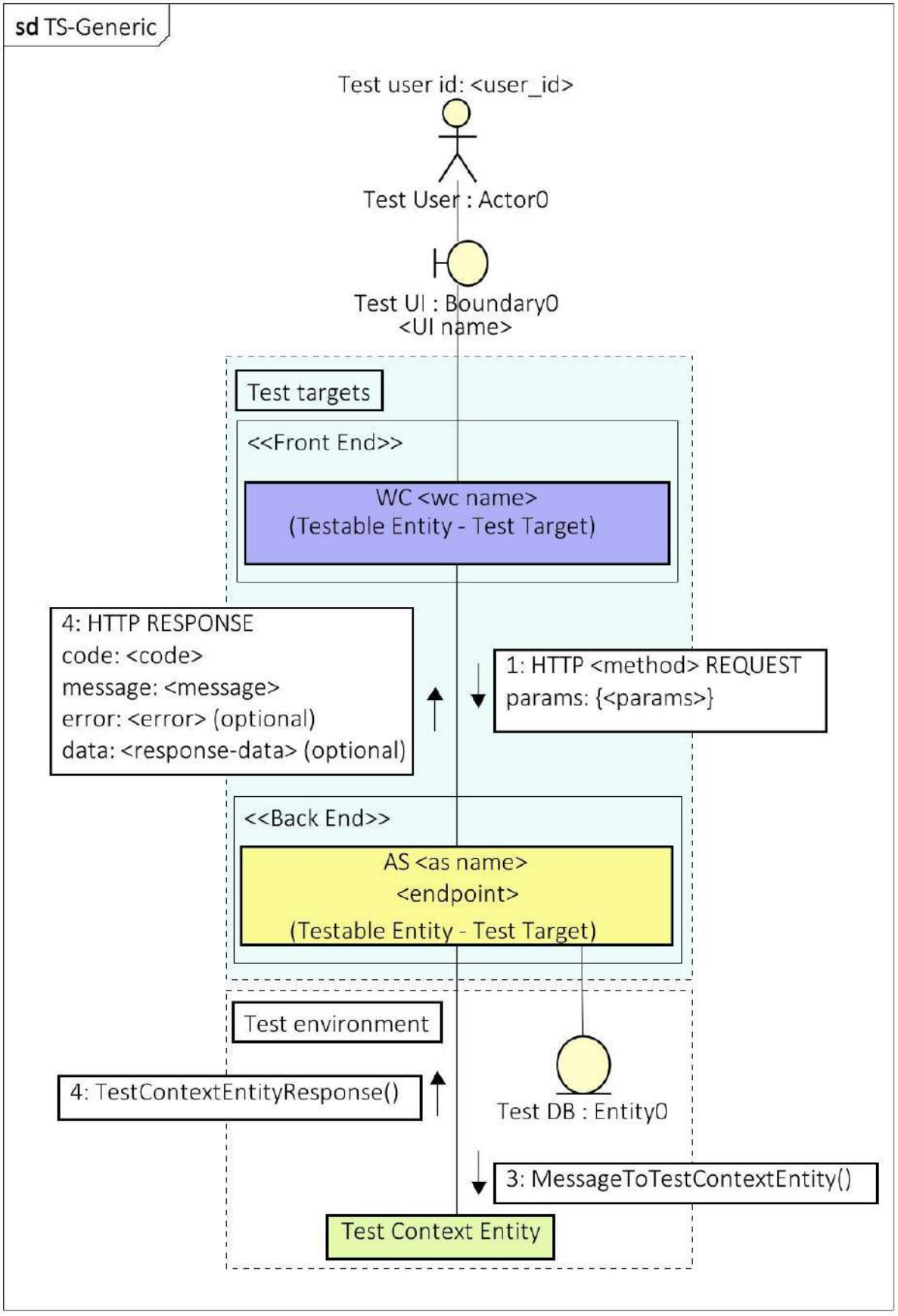



Fig. 6. Especificación de un escenario de prueba genérico usando un diagrama de comunicación UML.


➔ **Actor:** Dónde <*user_id*> es el id que figura en la base de datos de test para el usuario que está logueado al momento de ejecutar la prueba.


➔ **Interfaz gráfica:** Dónde <UI name> es el nombre con el cual se referencia a la interfaz gráfica por la cual el usuario interacciona con el componente web.

➔ **Front End:** Representado por una caja de línea sólida y titulada con la etiqueta <<*Front End*>>.

➔ **Back End:** Representado por una caja de línea sólida con la etiqueta <<*Back End*>>.

➔ **Componente Web:** Representado por una caja de color violeta con borde sólido y línea gruesa, donde <*wc name*> es el nombre del componente de la aplicación web que interviene en la prueba.

➔ **Servicio de la API:** Representado por una caja de color amarillo, con borde sólido y línea gruesa, donde <*as name*> es el nombre del servicio de la API que interviene en la prueba; <*endpoint*> es la url de comunicación con el servicio de la API, el cual será detallado en la descripción de cada *escenario*.

Nota: Los servicios de la API presentes en el diagrama pueden ser más de uno dependiendo de la funcionalidad analizada.

➔ **Mensajes:** Representados por una flecha que indica la dirección del mensaje y un cuadro de texto con borde sólido y línea gruesa, donde:

- ➢ <*method*> es el método HTTP con el que se realiza el request, y puede ser GET|POST|PUT|DELETE entre otros;
- ➢ <*params*> son los parámetros que se envían en el request, los mismos serán detallados en la descripción de cada *escenario*;
- ➢ <*code*> es el código de respuesta de la API, (una lista completa de los mismos está a disposición del lector en Anexo I: Test Basis);
- ➢ <*message*> es el mensaje de respuesta de la API, será detallado en cada *escenario;*
- ➢ <*error*> es el array de errores que responde la API, marcado como opcional ya que figura únicamente en casos de error, será detallado en cada *escenario* si corresponde;
- ➢ <*data*> es el array de datos que responde la API, marcado como opcional ya que figura únicamente en casos de success, será detallado en cada *escenario* si corresponde.

➔ **Focos u Objetos de Prueba (Test targets):** Representado con una caja de color celeste y borde punteado que contiene todos los test targets.

➔ **Contexto de Prueba (Test environment):** Representado con una caja transparente y de borde punteado que contiene todos los elementos que se consideran parte del contexto.

➔ **Entidad de contexto de Prueba (Test Context Entity):** Representado con una caja de color verde y borde sólido y linea gruesa.


➔ **Base de datos de Prueba (Test DB):** Parte del entorno de prueba.

Como se dijo anteriormente, la actividad A2 propone la iteración de las subactividades A2.1 y A2.2 para cada uno de los requerimientos de prueba obtenidos como resultado de la actividad A1. A continuación, se abordará el paso a paso solo de la primera iteración, es decir, la ejecución de las actividades A2.1 y A2.2 para el primero de los requerimientos identificado con la etiqueta *TR-Integration-ListContracts* (ver Tabla 6). Acto seguido y con motivos ilustrativos, se mostrarán las especificaciones de uno de los escenarios y sus casos de prueba correspondientes al requerimiento identificado como *TR-Integration-SignContract.* El lector puede encontrar los demás documentos producidos a lo largo de la segunda iteración en el Anexo II: Especificaciones de escenarios y casos de prueba.

Considerando el criterio de completitud descrito en la especificación del requerimiento *TR-Integration-ListContracts* (Tabla 6), a saber: “Se debe desarrollar al menos un caso de prueba para cada una de las siguientes situaciones: a) El servicio de la API respondió correctamente y el WC recibe un código 200 acompañado de un mensaje ‘Success’ y la lista de contratos filtrada y ordenada según lo esperado; b) Sucedió un error de parámetros en la URL, el WC recibe un código de error 404 acompañado de un mensaje ‘Param Error’.”, se identificaron dos escenarios a plantear, uno para un caso de éxito, es decir, una respuesta correcta con un código 200 (este escenario fue denominado TS-Integration-ListContracts-S200); y otro para un caso de error en uno de los parámetros de ruta, es decir, una respuesta de error con un código 404 (este escenario fue denominado TS-Integration-ListContracts-S404). Luego, teniendo en cuenta las entidades a probar (es decir, el WC listar-contratos y AS listar-contratos) y las entidad de contexto (Test Database) nombradas en la misma especificación de requerimiento de la Tabla 6, se elaboró una descripción detallada de ambos escenarios para finalmente especificarlos utilizando la plantilla de SaSTMe (recordar Fig. 3). En la Tabla 7 se encuentra la situación detallada para el caso de respuesta de un código 200, y en la Tabla 8 para el código 404. Además, se formalizaron ambas especificaciones utilizando diagramas de comunicación UML como se puede observar en las Fig. 7 y Fig. 8, respectivamente.

**Test Particular Situation:** TS-Integration-ListContracts-S200.
**Positive statement:** Con el objetivo de obtener correctamente un listado de contratos, en el WC listar-contratos se genera un request HTTP con método GET al endpoint base-url/users/user_id/companies/company_id/contracts, instanciando las variables user_id y company_id con los ids del usuario logueado y una empresa vinculada al usuario, respectivamente.
Con motivos de acotar el resultado obtenido, el request puede contener cualquier combinación de los siguientes parámetros de filtrado:

- signed: 0|1
- contract_type: integer
- dateFrom: Una fecha en formato 'YYYY-mm-dd'.
- dateTo: Una fecha en formato 'YYYY-mm-dd'.
  (Utilizando estos últimos dos en conjunto se puede filtrar los contratos por el rango que ambos forman, utilizarlos por separado no genera resultados apreciables)

El AS listar-contratos recibe la petición, la procesa y genera una respuesta con la siguiente estructura:
response:{code:200, message:'Success', response data: Objeto JSON}
Donde el Objeto JSON corresponde a un conjunto de contratos.
**(Particular) Situation Model Specification:** ver Fig. 7.
**Testable Entities:** WC listar-contratos (test target) y AS listar-contratos (test target)
**Test Context Entities:** Test DB.
**Test Requirement:** TR-Integration-ListContracts.
**Test Basis:** 'Descripción de las posibles respuestas de la API' (Tabla 1) , 'Test Data Pool' (Fig. 5), 'Descripción de los componentes de la Web App frontend' (Tabla 2) y 'Descripción de los servicios de la API backend' (Tabla 3).

Tabla 7. Documento de SaSTPro: Especificación de modelo de escenario ('Test Particular Situation's model specification') para el requisito 'TR-Integration-ListContracts' y el código de respuesta 200.

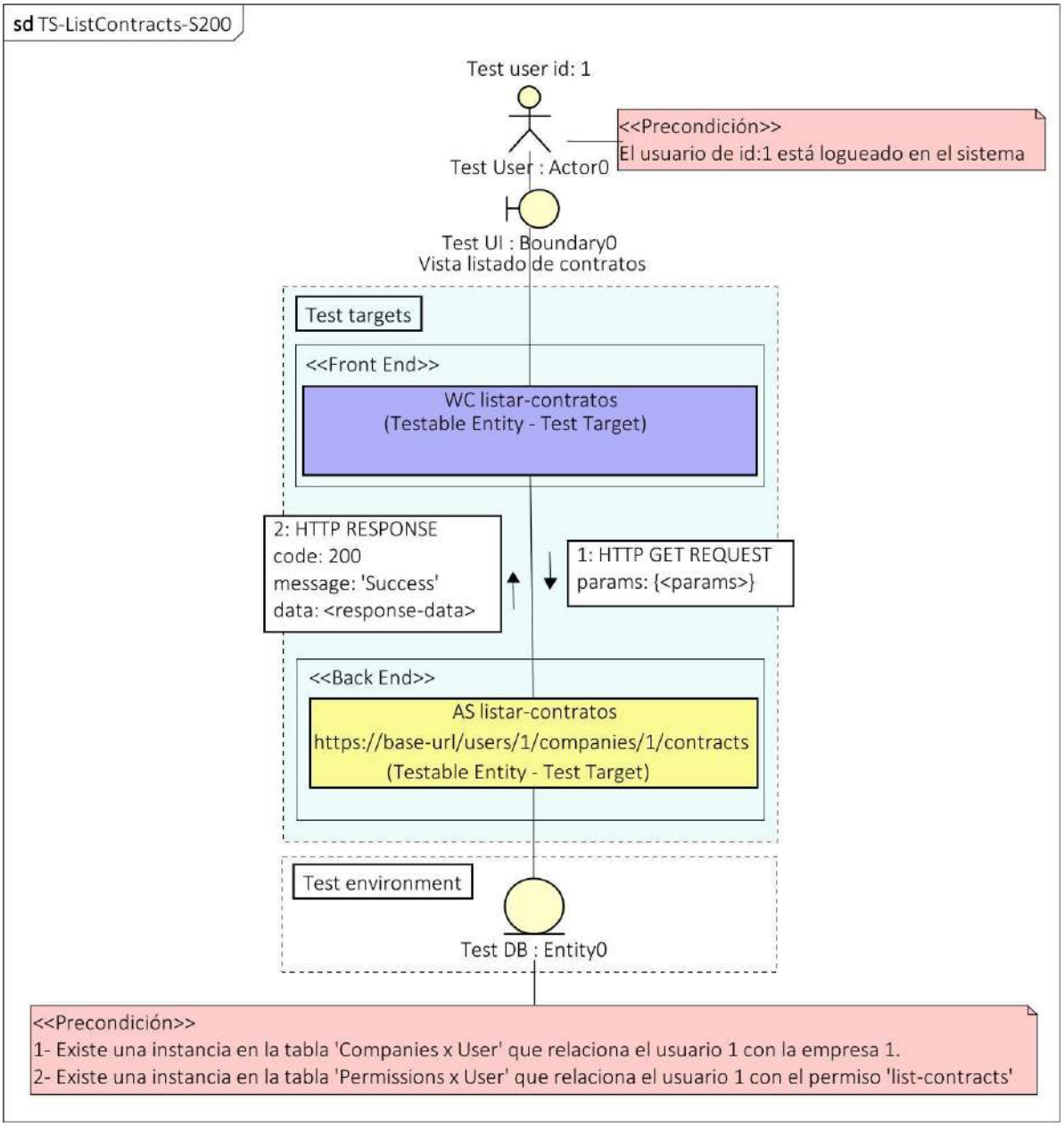



Fig. 7. Diagrama de comunicación UML, especificación formal del escenario TS-Integration-ListContracts-S200.

Como se puede ver en la Tabla 7, la situación particular etiquetada con el nombre TS-Integration-ListContracts-S200, plantea un escenario de éxito donde, el WC listar-contratos genera un request HTTP con el método GET al AS listar-contratos haciendo uso de su endpoint (base-url/users/**user_id**/companies/**company_id**/contracts descrito en Tabla 3), instanciando para ello los parámetros de url user_id con el id del usuario logueado en la plataforma y company_id con la empresa a la cual el usuario está vinculado. Opcionalmente, se pueden agregar parámetros de filtrado al request los cuales pueden ser: signed: 0|1 (para obtener solo los contratos firmados o sin firmar), contract_type: integer (para obtener solo los contratos que correspondan con el tipo dado), dateFrom y dateTo: Una fecha en formato ‘YYYY-mm-dd’ (para obtener solo los contratos generados en ese

rango de fecha). Finalmente, el AS listar-contratos genera una respuesta con código 200, acompañada de un objeto JSON que contendrá la lista de contratos de la empresa indicada y filtrada o no, según se le hayan indicado o no parámetros de filtrado.

Nótese que para lograr una respuesta de la API con código 200, se deben utilizar como user_id y company_id un par de ids que estén relacionados en la tabla 'Companies x Users' del 'Test Data Pool' (Fig. 5).

Por su parte, la situación particular etiquetada con el nombre TS-Integration-ListContracts-S404 se muestra en la Tabla 8. En esta se plantea un escenario de error donde el request generado por el WC listar-contratos posee un error en el parámetro de ruta 'company_id', es decir, el id de la empresa utilizado para generar el request no corresponde con el id de la empresa relacionada al usuario indicado con el parámetro 'user_id' en la url del endpoint del AS listar-contratos (base-url/users/**user_id**/companies/**company_id**/contracts). En función a ello, el AS listar-contratos genera una respuesta con un código de error 404 y un mensaje de error identificando el parámetro donde se encontró el error.

Nótese que para lograr una respuesta de la API con código 404 se deben utilizar como user_id y company_id un par de ids que **no** estén relacionados en la tabla 'Companies x Users', o bien, utilizar como company_id y contract_id un par de ids que **no** estén relacionados en la tabla 'Contracts' del 'Test Data Pool' (Fig. 5).

| |
|---|
| **Test Particular Situation:** TS-Integration-ListContracts-S404.<br>**Positive statement:** Con el objetivo de comprobar que se está bloqueando correctamente el acceso de un usuario a información correspondiente a otros usuarios, en el WC listar-contratos se genera un request HTTP con el método GET al endpoint base-url/users/user_id/companies/company_id/contracts, instanciando la variable user_id con el id del usuario logueado y la variable company_id con un id que no corresponde con el id de una empresa a la cual el usuario está vinculado.<br>El AS listar-contratos recibe la petición, la procesa y genera una respuesta con la siguiente estructura:<br>response:{code:404, message:'Param Error', errors:{company_id:'not associated'}}<br>**(Particular) Situation Model Specification:** ver Fig. 8.<br>**Testable Entities:** WC listar-contratos (test target) y AS listrar-contratos (test target)<br>**Test Context Entities:** Test DB.<br>**Test Requirement:** TR-Integration-ListContracts.<br>**Test Basis:** 'Descripción de las posibles respuestas de la API' (Tabla 1) , 'Test Data Pool' (Fig. 5), 'Descripción de los componentes de la Web App frontend' (Tabla 2) y 'Descripción de los servicios de la API backend' (Tabla 3). |

Tabla 8. Documento de SaSTPro: Especificación de modelo de escenario ('Test Particular Situation's model specification') para el requisito 'TR-Integration-ListContracts' y el código de respuesta 404.

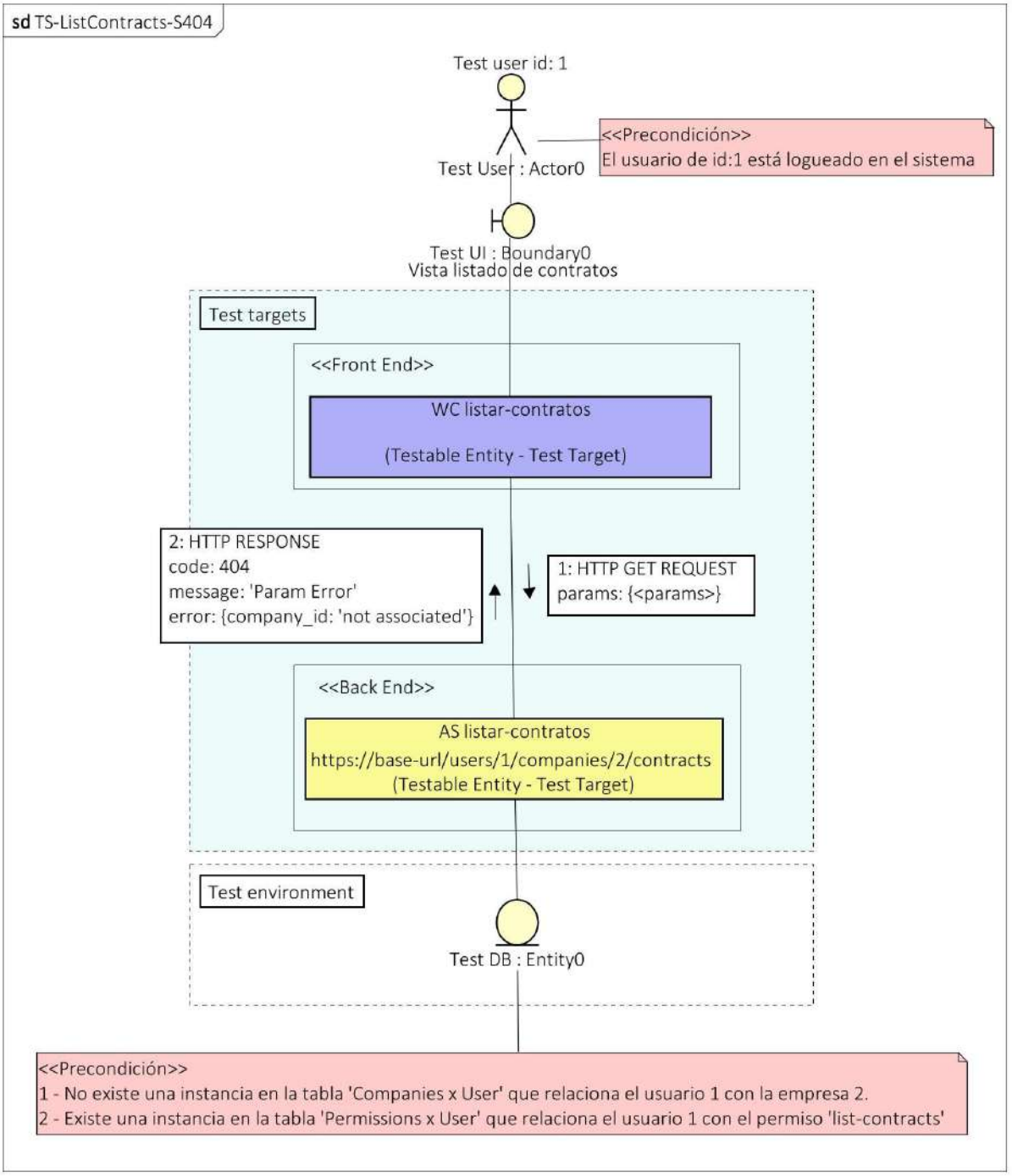



Fig. 8. Diagrama de comunicación UML, especificación formal del escenario TS-Integration-ListContracts-S404.

Habiendo obtenido las especificaciones de los distintos escenarios, se pasó a la subactividad A2.2, cuyo objetivo es diseñar los casos de prueba para cada escenario. Para esto, fue de gran ayuda considerar nuevamente la Test Basis, en esta oportunidad más específicamente el 'Test Data Pool'. Como se mencionó en el apartado 3.2, el test data pool es un subconjunto de los datos presentes en la base de datos de prueba de Agripay, el cual incluye sólo aquellos datos de importancia para la prueba a realizar, los cuales están representados mediante el uso de tablas.

Utilizando nuevamente la plantilla de SaSTMe, y con el fin de ilustrar un template completo, en la Tabla 9 se muestra el contenido de la Tabla 7 más los casos de prueba asociados al escenario “TS-Integration-ListContracts-S200”. Por su parte, en la Tabla 10, se muestran solamente los casos de prueba asociados al escenario documentado en la Tabla 8 (escenario “TS-Integration-ListContracts-S404”).

**Test Particular Situation:** TS-Integration-ListContracts-S200.
**Positive statement:** Con el objetivo de obtener correctamente un listado de contratos, en el WC listar-contratos se genera un request HTTP con método GET al endpoint base-url/users/user_id/companies/company_id/contracts, instanciando las variables user_id y company_id con los ids del usuario logueado y una empresa vinculada al usuario, respectivamente.
Con motivos de acotar el resultado obtenido, el request puede contener cualquier combinación de los siguientes parámetros de filtrado:

- signed: 0|1
- contract_type: integer
- dateFrom: Una fecha en formato ‘YYYY-mm-dd’.
- dateTo: Una fecha en formato ‘YYYY-mm-dd’.
  (Utilizando estos últimos dos en conjunto se puede filtrar los contratos por el rango que ambos forman, utilizarlos por separado no genera resultados apreciables)

El AS listar-contratos recibe la petición, la procesa y genera una respuesta con la siguiente estructura:
response:{code:200, message:’Success’, response data: Objeto JSON}
Donde el Objeto JSON corresponde a un conjunto de contratos.
**(Particular) Situation Model Specification:** ver Fig. 7.
**Testable Entities:** WC listar-contratos (test target) y AS listrar-contratos (test target)
**Test Context Entities:** Test DB.
**Test Requirement:** TR-Integration-ListContracts.
**Test Basis:** ‘Descripción de las posibles respuestas de la API’ (Tabla 1) , ‘Test Data Pool’ (Fig. 5), ‘Descripción de los componentes de la Web App frontend’ (Tabla 2) y ‘Descripción de los servicios de la API backend’ (Tabla 3).
**Test Case:** TC-Integration-ListContracts-S200-1
**Preconditions:** El usuario de id = 1 debe estar correctamente logueado, poseer el permiso para listar contratos, estar vinculado a la empresa de id =1 y estar posicionado en la vista de listado de contratos de la aplicación Web.
**Input:** { usuario id: 1, company id: 1, params: null }
**Expected Result:** code: 200, message:’Success’, response data: Objeto JSON correspondiente a:

| Contracts | | | | |
|---|---|---|---|---|
| ***contract_id*** | ***company_id*** | ***contract_type*** | ***date*** | ***signed*** |
| 1 | 1 | 1 | 06/03/2021 | 0 |
| 2 | 1 | 3 | 04/04/2021 | 1 |
| 5 | 1 | 4 | 14/07/2021 | 1 |

---

**Test Case:** TC-Integration-ListContracts-S200-2
**Preconditions:** El usuario de id = 1 debe estar correctamente logueado, poseer el permiso para listar contratos, estar vinculado a la empresa de id = 1 y estar posicionado en la vista de listado de contratos de la aplicación Web.
**Input:** { usuario id: 1, company id: 1, params: {contract_type: 3} }
**Expected Result:** code: 200, message:'Success', response data: Objeto JSON correspondiente a:

| Contracts | | | | |
|---|---|---|---|---|
| ***contract_id*** | ***company_id*** | ***contract_type*** | ***date*** | ***signed*** |
| 2 | 1 | 3 | 04/04/2021 | 1 |

---

**Test Case:** TC-Integration-ListContracts-S200-3
**Preconditions:** El usuario de id = 1 debe estar correctamente logueado, poseer el permiso para listar contratos, estar vinculado a la empresa de id =1 y estar posicionado en la vista de listado de contratos de la aplicación Web.
**Input:** { usuario id: 1, company id: 1, params: {signed: 1} }
**Expected Result:** code: 200, message:'Success', response data: Objeto JSON correspondiente a:

| Contracts | | | | |
|---|---|---|---|---|
| ***contract_id*** | ***company_id*** | ***contract_type*** | ***date*** | ***signed*** |
| 2 | 1 | 3 | 04/04/2021 | 1 |
| 5 | 1 | 4 | 14/07/2021 | 1 |

---

Tabla 9. Plantilla SaSTMe completa para el escenario TS-Integration-ListContracts-S200.

**Test Requirement:** TR-Integration-ListContracts.
**Test Particular Situation:** TS-Integration-ListContracts-S404
**Test Case:** TC-Integration-ListContracts-S404-1
**Preconditions:** El usuario de id = 1 debe estar correctamente logueado, poseer el permiso para listar contratos, no estar vinculado a la empresa de id = 2 y estar posicionado en la vista de listado de contratos de la aplicación Web.
**Input:** { usuario id: 1, company id: 2, params: null }
**Expected Result:** code: 404, message:'Param Error', errors: {company_id: 'not associated'}

Tabla 10. Documentos de SaSTPro: Casos de Prueba ('Test Cases') para el escenario TS-Integration-ListContracts-S404.

Como se puede ver en la Tabla 9, al escenario TS-Integration-ListContracts-S200, especificado en Tabla 7, se le agregaron tres casos de prueba a los cuales se los llamó TS-Integration-ListContracts-S200, TS-Integration-ListContracts-S200-2 y TS-Integration-ListContracts-S200-3. Recordar que en la Tabla 6 se indica como 'Completion Criteria' para el requerimiento TR-Integration-ListContracts que "se debe diseñar al menos un caso de prueba para el caso de respuesta 200". Pero, considerando lo indicado en la Tabla 7, a saber, "es posible indicar distintos parámetros de filtrado a la API a la hora de realizar la consulta por el listado de contratos", se decidió diseñar más de un caso de prueba con el fin de contemplar la utilización de distintos parámetros de filtrado. Además, para cada uno de estos casos de prueba, siguiendo el método SaSTMe, se deben indicar precondiciones ('preconditions'), entradas ('inputs') y resultados esperados ('Expected Results'). De modo que, se comenzó por definir las precondiciones. Para ello se utilizó la nota de "precondición" presente en el modelo de escenario TS-Integration-ListContracts-S200 (ver Fig. 7), donde se indica que, "El usuario de id = 1 debe estar correctamente logueado, poseer el permiso para listar contratos, estar vinculado a la empresa de id =1 y estar posicionado en la vista de listado de contratos de la aplicación Web". A su vez, el contar con estas precondiciones ayudó a identificar el conjunto de datos a tomar del "Test Data Pool" para definir las entradas de cada prueba. Estas entradas se definieron de la siguiente manera: { usuario id: 1, company id: 1, params: null }, para el caso de prueba TS-Integration-ListContracts-S200; { usuario id: 1, company id: 1, params: {contract_type: 3} }, para TS-Integration-ListContracts-S200-2 y { usuario id: 1, company id: 1, params: {signed: 1} }, para TS-Integration-ListContracts-S200-3. Finalmente, teniendo en cuenta las entradas para los distintos casos, se logró definir los resultados esperados. Estos resultados se expresaron como "code: 200, message:'Success', response data: Objeto JSON correspondiente a una tabla subconjunto de la tabla 'Contracts' del 'Test Data Pool' ". Particularmente, para el caso de prueba TS-Integration-ListContracts-S200, donde no se indicaron parámetros de filtrado, el subconjunto se definió como aquel formado por todos los contratos cuyo atributo company_id fuera igual a 1. Para el caso de prueba TS-Integration-ListContracts-S200-2, donde se utilizó como filtro {contract_type: 3}, el subconjunto fue el formado por todos los contratos con los atributos company_id igual a 1 y

contract_type igual a 3. Y por último, para el caso de prueba TS-Integration-ListContracts-S200-3, donde se indicó como parámetro de filtrado {signed:1}, el subconjunto que se definió como parte del resultado esperado fue aquel formado por todos los contratos ya firmados, es decir, aquellos cuyo atributo signed fuera igual a 1.

Por su parte, para el escenario TS-Integration-ListContracts-S404, se diseñó un único caso de prueba (Tabla 10). Para el mismo, y de forma análoga a los anteriores, se utilizó la nota 'Precondición' del modelo de dicho escenario (ver Fig. 8) para establecer las precondiciones, las cuales fueron, "El usuario de id = 1 debe estar correctamente logueado, poseer el permiso para listar contratos, no estar vinculado a la empresa de id = 2 y estar posicionado en la vista de listado de contratos de la aplicación Web". Nuevamente, teniendo en cuenta las precondiciones definidas, se tomó del "Test Data Pool" el conjunto de datos a utilizar como entrada para la prueba y se lo definió como: { usuario id: 1, company id: 2, params: null }. Recordar que la tabla 'Companies x Users' del "Test Data Pool" (ver Fig. 5) no cuenta con una entrada que relacione al usuario 1 con la empresa 2, de modo que, el resultado esperado para este caso de prueba será el de un error de código 404 y se lo expresó de la siguiente manera: code: 404, message:'Param Error', errors: {company_id: 'not associated'}.

Recordar que, todo lo anterior corresponde a la primera iteración realizada de las subactividades A2.1 y A2.2 de SaSTPro (ver Fig. 4). Del mismo modo, en la segunda iteración y para el requisito de prueba TR-Integration-SignContract, con el fin de establecer los distintos escenarios se consideró lo descrito en su 'Completion Criteria' (Tabla 6). En el mismo se indica que se debe diseñar al menos un caso de prueba para cada una de las siguientes situaciones: a) El servicio de la API respondió correctamente y el WC recibe un código 200 acompañado de un mensaje 'Success' y el contrato seleccionado figura como correctamente firmado; b) Sucedió un error de parámetros en la URL, el WC recibe un código de error 404 acompañado de un mensaje 'Param Error'; c) El usuario no tiene el permiso para firmar el contrato, el WC recibe un código de error 403 acompañado de un mensaje 'Authorization Error: Permission not granted'; d) Falló la entidad de contexto que hace el envío de SMS, el WC recibe un error sin código con el mensaje 'SMS sending failed'. e) Falló la entidad de contexto que hace el estampado blockchain para la trazabilidad de la firma, el WC recibe un error sin código con el mensaje 'Stamping failed, sign cancelled'. Para la última de las situaciones recién mencionadas, se definió un escenario al cual se lo llamó TS-Integration-SignContracts-SStamping, al mismo se lo describió utilizando un diagrama de comunicación UML y finalmente se diseñó un caso de prueba para el mismo al cual se lo llamó TC-Integration-SignContracts-SStamping. Para el diseño de este caso de prueba, se tomó nuevamente lo descrito en la nota "Precondición" del modelo de escenario para establecer las precondiciones, con ello se identificó del "Test Data Pool" el conjunto de datos a utilizar como entrada y finalmente se estableció el resultado

esperado como un mensaje de error notificando la falla en el servicio externo de estampado blockchain. En la Tabla 11 se puede ver la plantilla completa de SaSTMe para este caso de prueba. Además, como se indicó al comienzo de esta sección, el lector puede encontrar los documentos que contienen las definiciones de los demás escenarios, como así también los diseños de los casos de prueba para cada uno de ellos producidos durante esta iteración, en el Anexo II.

**Test Particular Situation:** TS-Integration-SignContracts-SStamping
**Positive Statement:** Con el objetivo de comprobar que se está procesando correctamente una falla externa relacionada al estampado blockchain, en el WC firmar-contratos se genera un request HTTP con el método POST al endpoint /base-url/users/user_id/send-sms, instanciando la variable user_id con el id del usuario logueado.

El AS enviar-sms recibe la petición, la procesa, genera un código de validación de 6 dígitos e indica al servicio externo de sms que lo envíe al número de teléfono asociado al usuario recibido en el request.

El servicio externo de SMS responde correctamente y notifica al AS, el cual genera una respuesta para el WC con la siguiente estructura:
{code:200, message:'Success: SMS sended'}.

El WC firmar-contratos procesa la respuesta de la API y en función al success hace visible al usuario un formulario para ingresar el código de verificación.

El usuario ingresa correctamente el código recibido y presiona el botón "firmar"
(Se supone que el usuario va a ingresar correctamente el código recibido ya que el objeto principal de prueba está en la comunicación front-end/back-end por lo que los errores humanos quedan fuera de consideración).

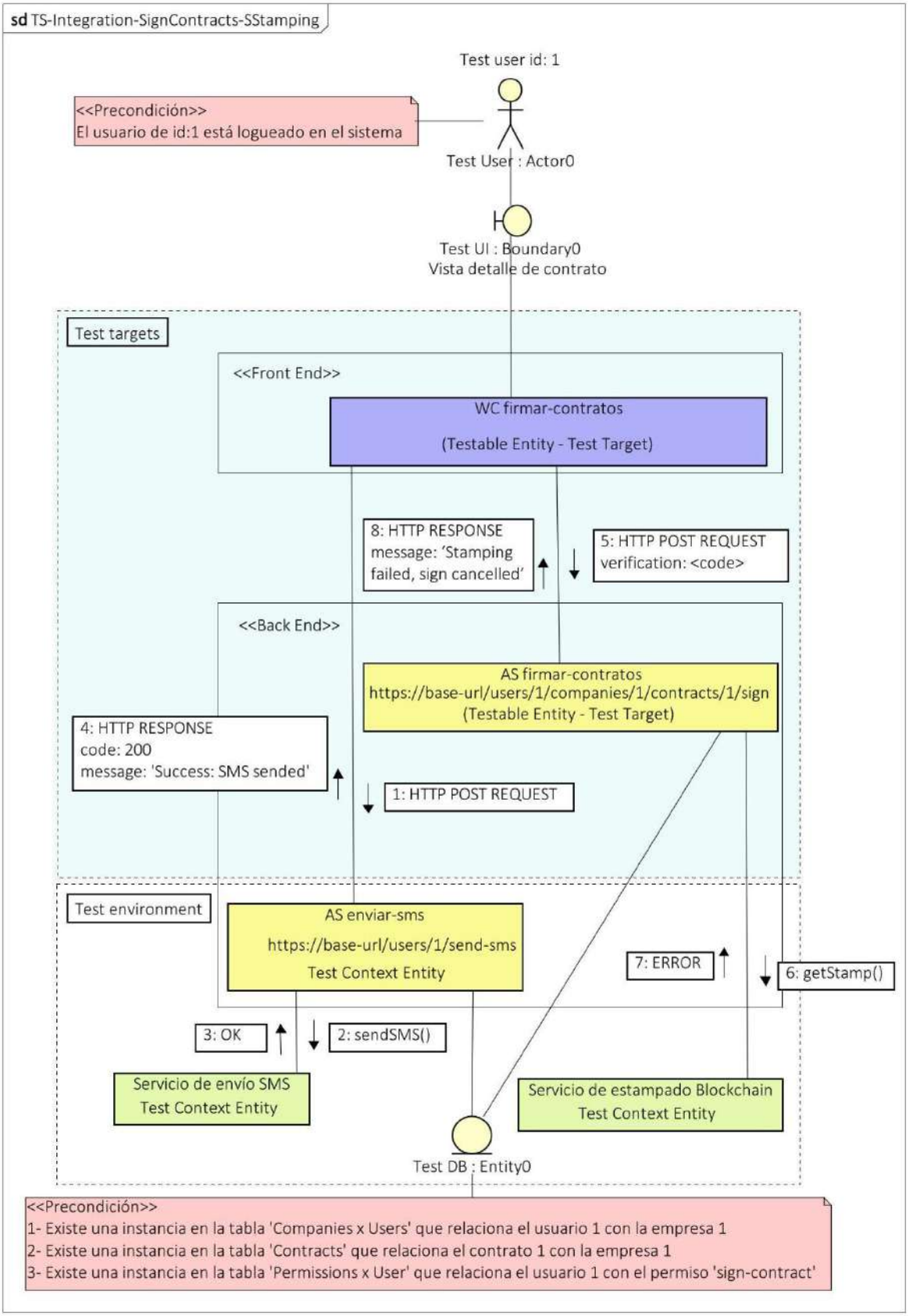
El WC firmar-contratos genera un request HTTP con método POST al endpoint base-url/users/**user_id**/companies/**company_id**/contracts/**contract_id**/sign, instanciando las variables user_id, company_id y contract_id con los ids del usuario logueado, una empresa vinculada al usuario y el id del contrato correspondiente a la empresa que va a ser firmado, respectivamente.

El AS firmar-contratos recibe la petición, la procesa e indica al servicio externo de estampado blockchain que genere una estampa y su correspondiente QR.

El servicio externo de estampado responde con algún error o no responde.

El AS genera una respuesta para el WC con la siguiente estructura:
response:{message: 'Stamping failed, sign cancelled'}.

**(Particular) Situation Model Specification:**




**Testable Entities:** WC firmar-contratos (test target) y AS firmar-contratos (test target)
**Test Context Entities:** Test DB, AS enviar-sms, Servicio externo de envío de SMS, Servicio externo de estampado blockchain.

**Test Requirement:** TR-Integration-SignContracts.
**Test Basis:** ‘Descripción de las posibles respuestas de la API’ (Tabla 1) , ‘Test Data Pool’ (Fig. 5), ‘Descripción de los componentes de la Web App frontend’ (Tabla 2) y ‘Descripción de los servicios de la API backend’ (Tabla 3).
**Test Case:** TC-Integration-SignContracts-SStamping.
**Preconditions:** El usuario de id = 1 debe estar correctamente logueado, poseer el permiso para firmar contratos, estar vinculado a la empresa de id = 1 y estar posicionado en la vista de firma de contrato para el contrato de id 1 de la aplicación Web.
**Input:** { usuario id: 1, company id: 1, contract id: 1 }
**Expected Result:** message: ‘Stamping failed, sign cancelled.

Tabla 11. Plantilla SaSTMe completa para el escenario TS-Integration-SignContracts-SStamping.

Como se mencionó al comienzo de este capítulo, con esto se dió por finalizada la aplicación de la estrategia, al menos en lo que a este trabajo final se refiere. Recordar que las actividades A2.3 y siguientes escapan del alcance de este proyecto y serán realizadas dentro del contexto del proyecto en Agripay, cuando el líder del equipo de desarrollo de la Web App frontend lo disponga oportuno.

# Capítulo 4: Conclusiones y Trabajo Futuro

## 4.1 Resultados y Conclusiones

En este trabajo, se utilizó la estrategia SaST para el diseño de escenarios y casos de prueba para verificar la correctitud funcional de parte de una plataforma Web destinada a la gestión de contratos inteligentes y pagos digitales, en relación a su interacción con una API backend la cual brinda servicios que dicha plataforma consume. Además, para ello se tuvieron en cuenta distintas entidades que forman parte del contexto, y que por lo tanto influencian el funcionamiento de los objetos de prueba seleccionados.

Haciendo un análisis de los documentos producidos, se identificó que, además de dar por cumplidos los objetivos planteados en un principio, se obtuvo un resultado aún mayor al esperado, ya que, si se cambia el concepto “web” por “mobile” en todas las especificaciones y consideramos cada WC como un componente frontend mobile (por ejemplo, para Android, iOS, etc), los diseños de escenarios y sus respectivos casos de prueba son completamente válidos para una app mobile que consuma los mismos servicios de la misma API. Por lo tanto, se puede concluir que, para proyectos similares al de Agripay, donde se cuenta con una Web App y una aplicación mobile que interactúan con la misma API backend, la estrategia SaST provee un nivel de abstracción capaz de diseñar escenarios y casos de prueba de integración frontend-backend aplicables indistintamente a los distintos frentes.

## 4.2 Apreciaciones personales

Sobre la estrategia en sí y a modo de apreciación personal, puedo agregar que, la encontré de fácil interpretación y gran coherencia en su especificación de proceso por brindar actividades concisas y de rápida ejecución. Por su parte, utilizar el template provisto por la especificación del método de la misma me facilitó la identificación e interpretación de la información necesaria para la ejecución de cada actividad. Además, personalmente, encontré de utilidad la aplicación de dicho template como modelo para el relevamiento de información del objeto de estudio. La única observación que puedo hacer sobre el mismo es que, considero importante hacer explícita la información respectiva a precondiciones para los distintos escenarios, por lo que, encuentro de utilidad el agregar un nuevo campo en el template SaSTMe para dicho fin, ya que, resolví dicha ausencia agregando notas a estilo de comentario en cada uno de los diagramas de situación, pero no lo encuentro del todo práctico.

Además, debo mencionar que me fue de gran utilidad el sustento teórico brindado por las distintas ontologías a la hora de estudiar y entender la estrategia y, particularmente el modelo de TestTDO a la hora de documentar los Test Requirements.

## 4.3 Trabajo futuro

Finalmente, queda como trabajo futuro el desarrollo de las actividades restantes de la estrategia para la implementación de los casos de prueba diseñados, cuando los tiempos de desarrollo lo ameriten y el director del equipo de desarrollo Web de Agripay lo considere oportuno. Por su parte, se dejará a disposición del director del equipo de desarrollo Mobile la documentación de los diseños de prueba elaborados para que se evalúe su utilización para la implementación de las mismas pruebas. Además, en caso de ser consideradas efectivas las pruebas una vez implementadas, el paso siguiente implicaría la extensión de este trabajo para analizar y diseñar los escenarios y casos de prueba para contemplar todas las interacciones de la Web App (o del Frontend en general de contar con la aprobación del equipo Mobile) y la API backend para finalmente implementar y automatizar todas las pruebas diseñadas haciendo uso de algún framework de testing compatible con las tecnologías de desarrollo utilizadas por cada uno de los equipos.

# Referencias


[1] ISO, “ISO/IEC/IEEE 29119-1, Software and systems engineering - Software Testing – Part 1: Concepts and definitions.” 2013.

[2] ISO, “ISO/IEC/IEEE 29119-4: Software and systems engineering - Software Testing – Part 4: Test techniques.” 2015.

[3] ISO, “ISO/IEC/IEEE 29119-2: Software and systems engineering - Software Testing – Part 2: Test processes.” 2013.

[4] L. Olsina and P. Becker, “Family of strategies for different evaluation purposes,” in *XX CIbSE’ 17*, 2017, pp. 221–234.

[5] D. Amalfitano, S. Matalonga, A. C. S. Doreste, A. R. Fasolino, and G. H. Travassos, “A Rapid Review on Testing of Context-Aware Contemporary Software Systems. Technical Report, RT-ES-760, Systems Engineering and Computer Science Department,” 2019.

[6] ISTQB, “International Software Testing Qualifications Board, Standard Glossary of Terms used in Software Testing, Version 3.2.” 2019, [Online]. Available: https://www.istqb.org/.

[7] G. Tebes, D. Peppino, P. Becker, and L. Olsina, “A Situation-aware Scenario-based Testing Strategy,” in *SCCC 2021 - 40th International Conference of the Chilean Computer Science Society, to appear in IEEE Xplore, (held virtually, Nov. 2021)*, 2021, pp. 1–8.

[8] L. Olsina, “Applicability of a Foundational Ontology to Semantically Enrich the Core and Domain Ontologies,” in *13th International Conference on Knowledge Engineering and Ontology Development (KEOD), IC3K*, 2021, pp. 111–119.

[9] G. Tebes, L. Olsina, D. Peppino, and P. Becker, “Specifying and Analyzing a Software Testing Ontology at the Top-Domain Ontological Level,” *Journal of Computer Science & Technology (JCS&T)*, vol. 21, no. 2, pp. 126–145, 2021.

[10] L. Olsina, G. Tebes, and P. Becker, “SituationCO v1.2’s Terms, Properties and Relationships - A Core Ontology for Particular and Generic Situations. Preprint in Research Gate.” 2021, doi: 10.13140/RG.2.2.23646.56644.

[11] P. Becker, F. Papa, G. Tebes, and L. Olsina, “Analyzing a Process Core Ontology and its Usefulness for Different Domains,” in *Springer Nature Switzerland AG Int’l Conference on the Quality of Information and Communication Technology, A. C. R. Paiva et al. (Eds.): QUATIC 2021, CCIS 1439*, 2021, pp. 183–196.

[12] A. C. De Souza Doreste and G. H. Travassos, “Towards Supporting the Specification of Context-Aware Software System Test Cases,” in *Proceedings of the XXIII Iberoamerican Conference on Software Engineering, CIbSE 2020*, 2020, pp. 356–363.

[13] OMG, “Software & Systems Process Engineering Meta-Model Specification v. 2.0.” 2008.

[14] D. Nalic, H. Li, A. Eichberger, C. Wellershaus, A. Pandurevic, and B. Rogic, “Stress Testing Method for Scenario-Based Testing of Automated Driving Systems,” *IEEE*

*Access*, vol. 8, pp. 224974–224984, 2020, doi: 10.1109/ACCESS.2020.3044024.

[15] C. King, L. Ries, C. Kober, C. Wohlfahrt, and E. Sax, “Automated Function Assessment in Driving Scenarios,” in *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, 2019, pp. 414–419, doi: 10.1109/ICST.2019.00050.

[16] S. Zhang, J. Shi, K. Guo, and Y. Wang, “Virtual Validation Method for Automated Driving Vehicles Based on Traffic Accident,” in *CICTP 2020*, 2020, pp. 365–375.

[17] K. Baek, H. Moon, and I.-Y. Ko, “VR-Powered Scenario-Based Testing for Visual and Acoustic Web of Things Services,” in *Bakaev M., Frasincar F., Ko IY. (eds) Web Engineering. ICWE 2019. Lecture Notes in Computer Science*, 2019, pp. 514–518.

[18] G. Murad, A. Badarneh, A. Qusef, and F. Almasalha, “Software Testing Techniques in IoT,” in *2018 8th International Conference on Computer Science and Information Technology (CSIT)*, 2018, pp. 17–21, doi: 10.1109/CSIT.2018.8486149.

# Anexos

---

## ANEXO I: Test Basis

### Posibles respuestas de la API:

Para cada consulta realizada por parte de un componente de la Web App a un servicio de la API backend, éste último realiza una respuesta en formato JSON que cumple con el siguiente formato:

```
{
        code: number
        message: string
        errors: [{string: string}] (Solo presente en casos de error)
        data: JSON (Solo presente en casos 'success')
}
```

A continuación se da una lista completa de todas las posibles respuestas de la API y bajo qué condiciones se da cada una:

| *CODE* | *MESSAGE* | *ERROR* | *CONDICIÓN* | *COMENTARIO* |
|---|---|---|---|---|
| 500 | Server Error | null | Ocurrió un fallo en la API | Inviable de probar por no ser un escenario reproducible sin alterar la testable entity |
| 200 | Success | null | El AS procesó correctamente la consulta | El campo 'data' que acompaña esta respuesta variará según el AS considerado. |
| 402 | Authorization Error: Token missing or expired | sesion-expirada : token inválido o vencido | La sesión del usuario finalizó y su token está vencido. O bien, | Esta respuesta es indistinta para cualquier par de WC y AS |

| | | | | |
|---|---|---|---|---|
| | | | el token indicado por parámetro es inválido | que se tomen, ya que depende de la sesión del usuario y no de la acción que está intentado realizar. |
| 403 | Authorization Error: Permission not granted | <nombre de permiso>: ‘missing’ | El usuario no posee el permiso necesario para realizar ésta acción | |
| 404 | URL not found | error: La url no pudo ser procesada | El endpoint consultado no corresponde a un AS. | |
| | URL param error | Param Error: <nombre del parámetro> <descripción> | Alguno de los parámetros que conforman el endpoint no pudo ser procesado | El endpoint de algunos AS está formado por los ids de distintas entidades a referenciar. Si alguno de dichos ids es erróneo, la respuesta indica cuál o cuáles lo son. En caso de ser más de uno, errors pasa a ser un array con esa misma estructura |
| 405 | Method not allowed | HTTP: El método indicado no es el correcto | El método http utilizado para realizar la consulta no es el esperado por el AS | Se realizó un request GET a un endpoint etiquetado como POST en la API o viceversa. |
| 422 | Unprocessable Param | <nombre del parámetro> : No es un parámetro reconocible | Alguno de los parámetros http usados en la consulta no es reconocido por el AS. | De forma análoga al anterior y cumpliendo para los dos casos, si es |

| | | | | |
|---|---|---|---|---|
| | Param Missing | <nombre del parámetro> : No fue brindado | En la consulta http faltó alguno de los parámetros requeridos por el AS. | más de uno los parámetros, errors para a ser un array con esa estructura, o una combinación de los dos si pasan ambas cosas al mismo tiempo. |
| - | Connection Error | - | No se pudo establecer la conexión con el servidor de la API | Común y ajeno a todo par de WC y AS que se tomen. |

# ANEXO II: Especificaciones de escenarios y casos de prueba para el requisito ‘Firmar Contratos’

En este anexo se encuentran los escenarios restantes para el requisito de prueba ‘Firmar Contratos’, los cuales no se mostraron en el Capítulo 3. Además, se documentan los respectivos casos de prueba .

## Requerimiento de prueba (‘Test Requirements’): Firmar Contratos.

**Label**: TR-Integration-SignContract
**Statement**: Se requiere verificar la correctitud funcional del **WC firmar-contratos** en relación a su interacción con el **AS firmar-contratos**.
**Testable entity phase**: Desarrollo
**Test level**: Integración
**Completion Criteria**: Se debe desarrollar al menos un caso de prueba para cada una de las siguientes situaciones:

- El servicio de la API respondió correctamente y el WC recibe un código 200 acompañado de un mensaje ‘Success’ y el contrato seleccionado figura como correctamente firmado.
- Sucedió un error de parámetros en la URL, el WC recibe un código de error 404 acompañado de un mensaje ‘Param Error’.
- El usuario no tiene el permiso para firmar el contrato, el WC recibe un código de error 403 acompañado de un mensaje ‘Authorization Error: Permission not granted’.
- Falló la entidad de contexto que hace el envío de SMS, el WC recibe un error sin código con el mensaje ‘SMS sending failed’.
- Falló la entidad de contexto que hace el estampado blockchain para la trazabilidad de la firma, el WC recibe un error sin código con el mensaje ‘Stamping failed, sign cancelled’.

**Refers to Testable Entities**: WC firmar-contrato, AS firmar-contrato.
**Refers to Test Context Entities**: AS enviar-sms, Servicio externo de envío de SMS, Servicio externo de estampado blockchain, Test DB

Plantilla SaSTMe completa para los diferentes escenarios del requisito de prueba de Firmar Contratos:

**Test Particular Situation:** TS-Integration-SignContracts-S200
**Positive Statement:** Con el objetivo de firmar correctamente un contrato, en el WC firmar-contratos se genera un request HTTP con el método POST al endpoint /base-url/users/user_id/send-sms, instanciando la variable user_id con el id del usuario logueado.

El AS enviar-sms recibe la petición, la procesa, genera un código de validación de 6 dígitos e indica al servicio externo de sms que lo envíe al número de teléfono asociado al usuario recibido en el request.

El servicio externo de SMS responde correctamente y notifica al AS, el cual genera una respuesta para el WC con la siguiente estructura:
{code:200, message:'Success: SMS sended'}.

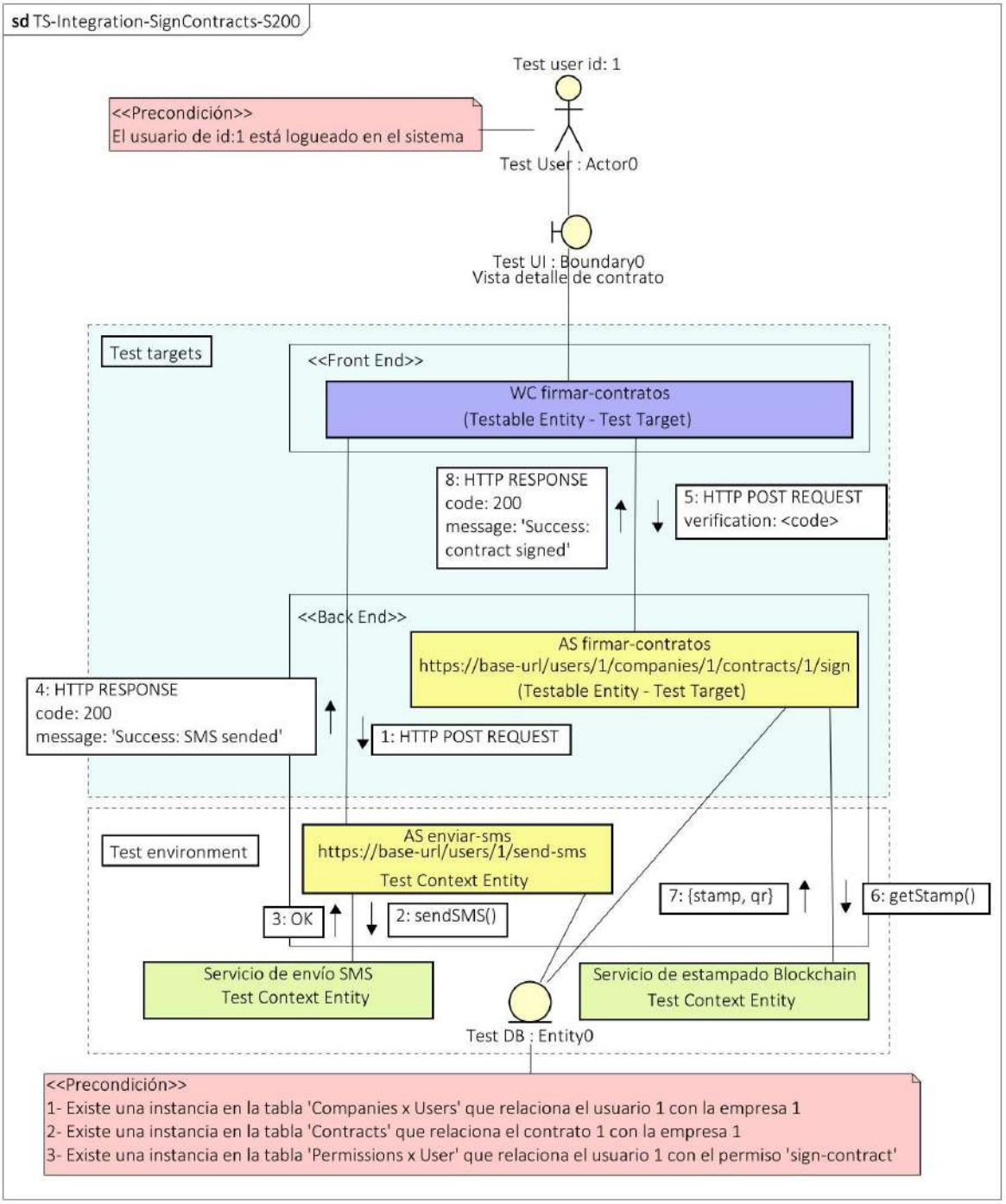
El WC firmar-contratos procesa la respuesta de la API y en función al success hace visible al usuario un formulario para ingresar el código de verificación.

El usuario ingresa correctamente el código recibido y presiona el botón "firmar"
(Se supone que el usuario va a ingresar correctamente el código recibido ya que el objeto principal de prueba está en la comunicación front-end/back-end por lo que los errores humanos quedan fuera de consideración).

El WC firmar-contratos genera un request HTTP con método POST al endpoint base-url/users/user_id/companies/company_id/contracts/contract_id/sign, instanciando las variables user_id, company_id y contract_id con los ids del usuario logueado, una empresa vinculada al usuario y el id del contrato correspondiente a la empresa que va a ser firmado, respectivamente.

El AS firmar-contratos recibe la petición, la procesa e indica al servicio externo de estampado blockchain que genere una estampa y su correspondiente código QR.

El servicio externo de estampado responde correctamente y el AS genera una respuesta para el WC con la siguiente estructura:
response:{code:200, message:'Success: contract signed'}.
**(Particular) Situation Model specification:**




**Testable Entities:** WC firmar-contratos (test target) y AS firmar-contratos (test target)
**Test Context Entities:** Test DB, AS enviar-sms, Servicio externo de envío de SMS, Servicio externo de estampado blockchain.
**Test Requirement:** TR-Integration-SignContracts.
**Test Basis:** 'Descripción de las posibles respuestas de la API' (Tabla 1) , 'Test Data Pool' (Fig. 5), 'Descripción de los componentes de la Web App frontend' (Tabla 2) y 'Descripción de los servicios de la API backend' (Tabla 3).
**Test Case:** TC-Integration-SignContracts-S200-1
**Preconditions:** El usuario de id = 1 debe estar correctamente logueado, poseer el permiso para firmar contratos, estar vinculado a la empresa de id = 1 y estar posicionado

en la vista de firma de contrato para el contrato de id 1 de la aplicación Web. La instancia de la tabla Contracts correspondiente al contrato de id 1 debe tener su atributo signed en 0.

| Contracts | | | | |
|---|---|---|---|---|
| ***contract_id*** | ***company_id*** | ***contract_type*** | ***date*** | ***signed*** |
| 1 | 1 | 1 | 06/03/2021 | 0 |
| 2 | 1 | 3 | 04/04/2021 | 1 |
| 3 | 2 | 3 | 11/10/2021 | 0 |
| 4 | 2 | 2 | 20/02/2021 | 0 |
| 5 | 1 | 4 | 14/07/2021 | 1 |

**Postconditions:** Ver un cambio en la tabla de contratos en la base de datos reflejando la firma de la siguiente forma:

| Contracts | | | | |
|---|---|---|---|---|
| ***contract_id*** | ***company_id*** | ***contract_type*** | ***date*** | ***signed*** |
| 1 | 1 | 1 | 06/03/2021 | 1 |
| 2 | 1 | 3 | 04/04/2021 | 1 |
| 3 | 2 | 3 | 11/10/2021 | 0 |
| 4 | 2 | 2 | 20/02/2021 | 0 |
| 5 | 1 | 4 | 14/07/2021 | 1 |

**Input:** { usuario id: 1, company id: 1, contract id: 1 }
**Expected Result:** code: 200, message:message: 'Success: contract signed'.

---

**Test Particular Situation:** TS-Integration-SignContracts-S403
**Positive Statement:** Con el objetivo de comprobar que se está bloqueando correctamente el proceso de firma para usuarios sin permiso, en el WC firmar-contratos se genera un request HTTP con el método POST al endpoint /base-url/users/user_id/send-sms, instanciando la variable user_id con el id del usuario logueado.

El AS enviar-sms recibe la petición, la procesa, genera un código de validación de 6 dígitos e indica al servicio externo de sms que lo envíe al número de teléfono asociado al usuario recibido en el request.

El servicio externo de SMS responde correctamente y notifica al AS, el cual genera una respuesta para el WC con la siguiente estructura:
{code:200, message:'Success: SMS sended'}.

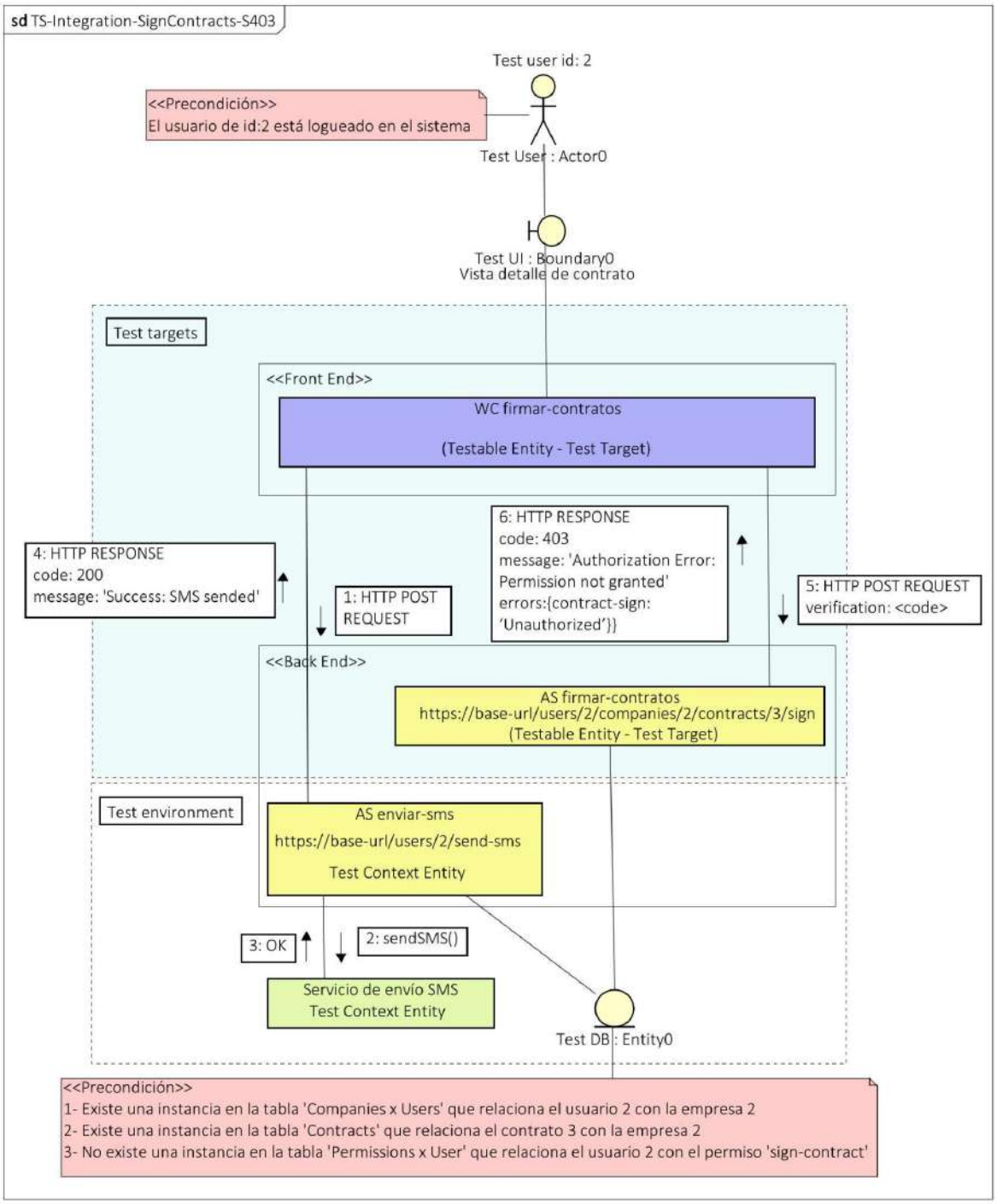
El WC firmar-contratos procesa la respuesta de la API y en función al success hace visible al usuario un formulario para ingresar el código de verificación.

El usuario ingresa correctamente el código recibido y presiona el botón “firmar”
(Se supone que el usuario va a ingresar correctamente el código recibido ya que el objeto principal de prueba está en la comunicación front-end/back-end por lo que los errores humanos quedan fuera de consideración).

El WC firmar-contratos genera un request HTTP con método POST al endpoint base-url/users/user_id/companies/company_id/contracts/contract_id/sign, instanciando las variables user_id, company_id y contract_id con los ids del usuario logueado, una empresa vinculada al usuario y el id del contrato correspondiente a la empresa que va a ser firmado, respectivamente.

El AS firmar-contratos recibe la petición, la procesa y genera una respuesta para el WC con la siguiente estructura:
response:{code:403, message:’Authorization Error: Permission not granted’, errors:{contract-sign: ‘Unauthorized’}}.

**(Particular) Situation Model specification:**




**Testable Entities:** WC firmar-contratos (test target) y AS firmar-contratos (test target)
**Test Context Entities:** Test DB, AS enviar-sms, Servicio externo de envío de SMS.
**Test Requirement:** TR-Integration-SignContracts.
**Test Basis:** 'Descripción de las posibles respuestas de la API' (Tabla 1) , 'Test Data Pool' (Fig. 5), 'Descripción de los componentes de la Web App frontend' (Tabla 2) y 'Descripción de los servicios de la API backend' (Tabla 3).
**Test Case:** TC-Integration-SignContracts-S403-1
**Preconditions:** El usuario de id = 2 debe estar correctamente logueado, no poseer el permiso para firmar contratos, estar vinculado a la empresa de id = 2 y estar posicionado en la vista de firma de contrato para el contrato de id 3 de la aplicación Web.

**Input:** { usuario id: 2, company id: 2, contract id: 3 }
**Expected Result:** code: 403, message: 'Authorization Error: Permission not granted', errors:{contract-sign: ‘Unauthorized’}

---

**Test Particular Situation:** TS-Integration-SignContracts-S404
**Positive Statement:** Con el objetivo de comprobar que se está bloqueando correctamente las acciones del usuario sobre contratos de otras empresas, en el WC firmar-contratos se genera un request HTTP con el método POST al endpoint /base-url/users/user_id/send-sms, instanciando la variable user_id con el id del usuario logueado.

El AS enviar-sms recibe la petición, la procesa, genera un código de validación de 6 dígitos e indica al servicio externo de sms que lo envíe al número de teléfono asociado al usuario recibido en el request.

El servicio externo de SMS responde correctamente y notifica al AS, el cual genera una respuesta para el WC con la siguiente estructura:
{code:200, message:’Success: SMS sended’}.

El WC firmar-contratos procesa la respuesta de la API y en función al success hace visible al usuario un formulario para ingresar el código de verificación.

El usuario ingresa correctamente el código recibido y presiona el botón “firmar”
(Se supone que el usuario va a ingresar correctamente el código recibido ya que el objeto principal de prueba está en la comunicación front-end/back-end por lo que los errores humanos quedan fuera de consideración).

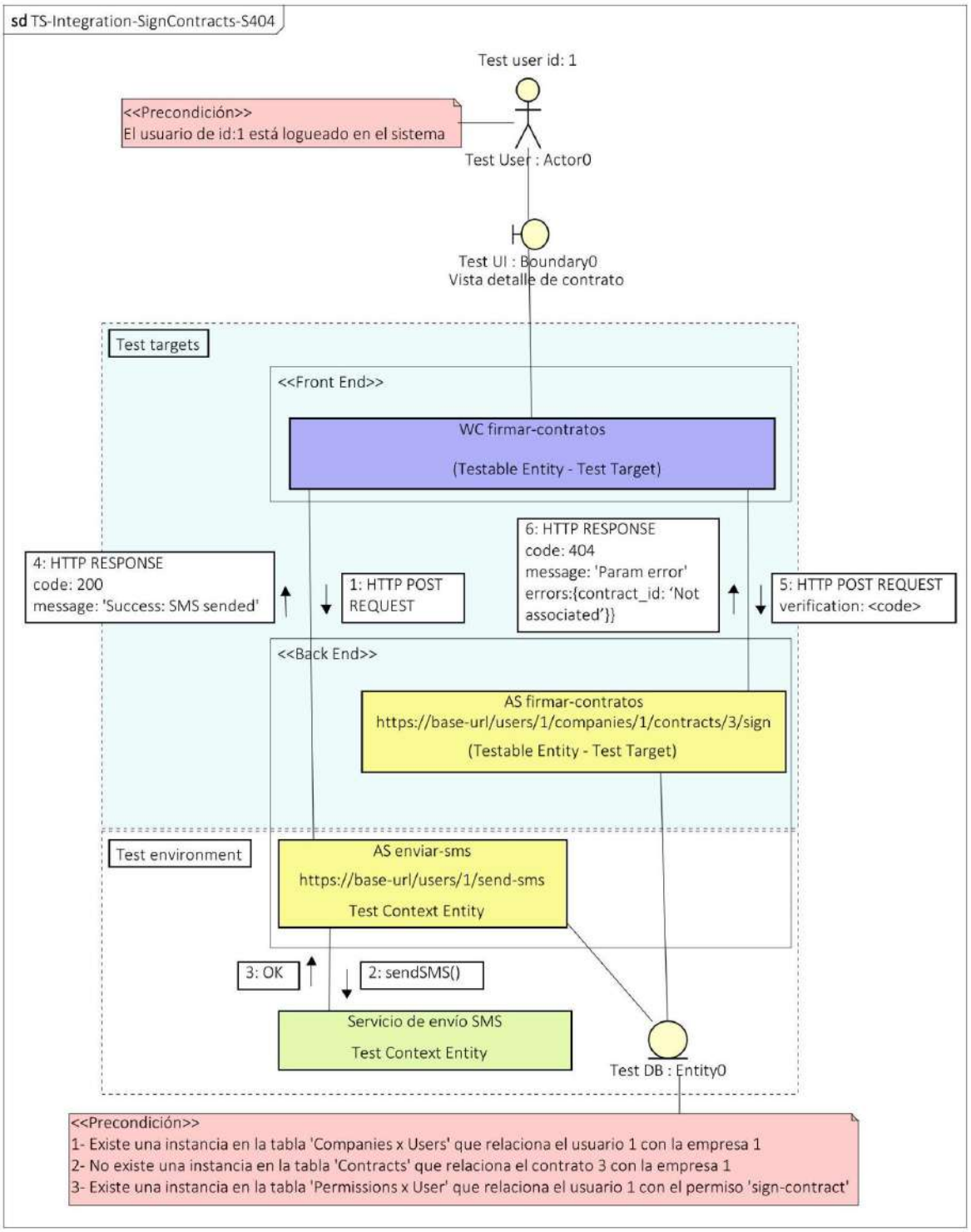
El WC firmar-contratos genera un request HTTP con método POST al endpoint base-url/users/user_id/companies/company_id/contracts/contract_id/sign, instanciando las variables user_id, company_id y contract_id con algún conjunto de datos erróneos.

El AS firmar-contratos recibe la petición, la procesa y genera una respuesta para el WC con la siguiente estructura:
response:{code:404, message:’Param error’, errors:{<errors>}}.

Donde <errors> puede ser cualquier combinación de los siguientes valores según la situación dada:

- contract_id: ‘not associated’ (si el id de contrato no corresponde al id de empresa)
- company_id: ’not associated’ (si el usuario no está relacionado a la empresa)

**(Particular) Situation Model specification:**




**Testable Entities:** WC firmar-contratos (test target) y AS firmar-contratos (test target)
**Test Context Entities:** Test DB, AS enviar-sms, Servicio externo de envío de SMS.
**Test Requirement:** TR-Integration-SignContracts.
**Test Basis:** 'Descripción de las posibles respuestas de la API' (Tabla 1) , 'Test Data Pool' (Fig. 5), 'Descripción de los componentes de la Web App frontend' (Tabla 2) y 'Descripción de los servicios de la API backend' (Tabla 3).
**Test Case:** TC-Integration-SignContracts-S404-1

**Preconditions:** El usuario de id = 1 debe estar correctamente logueado, poseer el permiso para firmar contratos, estar vinculado a la empresa de id = 1 y estar posicionado en la vista de firma de contrato para el contrato de id 3 de la aplicación Web.
**Input:** { usuario id: 1, company id: 1, contract id: 3 }
**Expected Result:** code: 404, message: 'Param error', errors:{contract_id: 'Not associated'}

---

**Test Case:** TC-Integration-SignContracts-S404-2
**Preconditions:** El usuario de id = 1 debe estar correctamente logueado, poseer el permiso para firmar contratos, no estar vinculado a la empresa de id = 2 y estar posicionado en la vista de firma de contrato para el contrato de id 3 de la aplicación Web.
**Input:** { usuario id: 1, company id: 2, contract id: 3 }
**Expected Result:** code: 404, message: 'Param error', errors:{company_id: 'Not associated'}

---

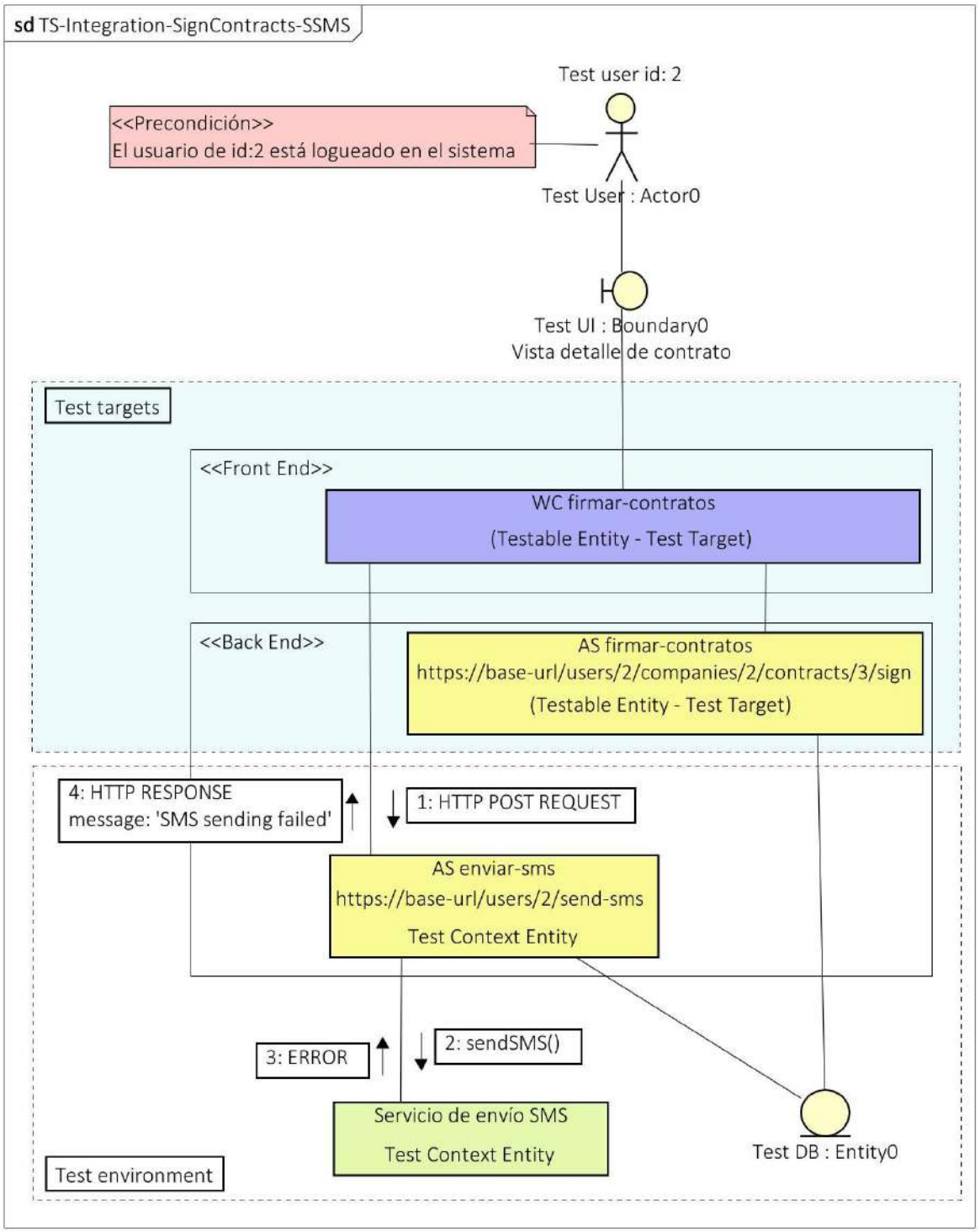
**Test Particular Situation:** TS-Integration-SignContracts-SSMS
**Positive Statement:** Con el objetivo de comprobar que se está procesando correctamente una falla externa relacionada a los SMS, en el WC firmar-contratos se genera un request HTTP con el método POST al endpoint /base-url/users/user_id/send-sms, instanciando la variable user_id con el id del usuario logueado.

El AS enviar-sms recibe la petición, la procesa, genera un código de validación de 6 dígitos e indica al servicio externo de sms que lo envíe al número de teléfono asociado al usuario recibido en el request.

El servicio externo de SMS responde con un error o no responde.

El AS genera una respuesta para el WC con la siguiente estructura:
{message: 'SMS sending failed'}.

**(Particular) Situation Model specification:**




**Testable Entities:** WC firmar-contratos (test target) y AS firmar-contratos (test target)
**Test Context Entities:** Test DB, AS enviar-sms, Servicio externo de envío de SMS.
**Test Requirement:** TR-Integration-SignContracts.
**Test Basis:** 'Descripción de las posibles respuestas de la API' (Tabla 1) , 'Test Data Pool' (Fig. 5), 'Descripción de los componentes de la Web App frontend' (Tabla 2) y 'Descripción de los servicios de la API backend' (Tabla 3).
**Test Case:** TC-Integration-SignContracts-SMS

**Preconditions:** El usuario de id = 2 debe estar correctamente logueado estar posicionado en la vista de firma de contrato para el contrato de id 3 de la aplicación Web.
**Input:** { usuario id: 2, company id: 2, contract id: 3 }
**Expected Result:** message: SMS sending failed'