Especificación Formal y Validación para la Prevención de Colisiones en Tráfico Ferroviario

Infanzón Acosta R. E.[1], Aguilar Chirinos C. D.[1], and Molina Barriga M. - Mentor [2]

[1]Universidad La Salle, Arequipa, Perú

***Resumen*—Este artículo presenta una especificación formal y un proceso de validación orientados a la prevención de colisiones en el tráfico ferroviario. Se propone un modelo basado en técnicas formales, específicamente utilizando VDM++, para estructurar y analizar la ocupación de vías y las distancias entre trenes, asegurando la coherencia y validez en la toma de decisiones automáticas. Además, se emplean técnicas de model checking con UPPAAL para verificar propiedades de seguridad y tiempos de respuesta ante obstáculos o situaciones peligrosas. Se destacan los avances en la integración de sensores sofisticados—como sensores de comunicación, detección de objetos y comunicación entre trenes—que proporcionan datos precisos para anticipar potenciales colisiones. El enfoque contribuye a desarrollar sistemas de control más seguros, confiables y eficientes, fortaleciendo la seguridad en la operación ferroviaria mediante metodologías rigurosas de especificación y validación.**

***Index Terms*—Prevención de colisiones ferroviarias, Especificación formal , VDM++, Model checking, Sistemas de seguridad ferroviaria**

I. Introducción

La seguridad en el transporte ferroviario es un aspecto crítico para garantizar el bienestar de las personas que utilizan este medio de transporte a diario. Según el *Informe sobre la Seguridad y la Interoperabilidad Ferroviaria en la UE 2024*, el sistema ferroviario de la UE se considera uno de los más seguros del mundo. Sin embargo, a pesar de la disminución de accidentes significativos desde 2010, se ha registrado un aumento en 2021 y 2022, alineándose con los niveles prepandemia de COVID-19. Esta realidad se ve agravada por el estancamiento de las tasas de fatalidades de pasajeros desde 2017 [1]. Las repercusiones de estos incidentes se extienden más allá de los pasajeros y trabajadores, afectando profundamente a las comunidades y economías que dependen del ferrocarril [2].

Ante esta situación, es fundamental explorar nuevas estrategias para prevenir colisiones y reforzar un sistema más seguro. La implementación de tecnologías avanzadas en el ámbito de la seguridad ferroviaria se ha convertido en una prioridad. Investigaciones recientes indican que la evolución de las tecnologías de seguridad está proporcionando respuestas más efectivas ante situaciones peligrosas [3]. En este contexto, se busca desarrollar un mecanismo que minimice el riesgo de colisiones y reduzca los errores humanos y las fallas en la comunicación dentro del sistema ferroviario. La automatización de tareas críticas, como la detección de trenes en las vías y el control de su velocidad, es esencial para una respuesta adecuada ante situaciones potencialmente peligrosas [4].

Además, será necesario diseñar un modelo formal que estructure y analice la información relacionada con la seguridad ferroviaria, asegurando así la precisión y la integridad de los datos utilizados en la prevención de colisiones. Este enfoque permitirá mejorar la identificación de situaciones de riesgo y sentar las bases para futuras herramientas de mejora en la seguridad ferroviaria.

I-A. Alcance esperado

El alcance de este proyecto se centra en la aplicación de métodos formales para estructurar y analizar el sistema de prevención de colisiones ferroviarias. A través del uso de VDM++ y model-checking, se busca desarrollar un modelo formal que permita verificar la consistencia y validez de las interacciones entre los componentes del sistema, como los trenes y sensores. Este enfoque garantizará la integridad de las decisiones automatizadas, como los cambios de señales o ajustes de velocidad, y asegurará que el sistema funcione correctamente en todas las condiciones posibles. Mediante un análisis riguroso y sistemático, se contribuirá a mejorar la seguridad del tráfico ferroviario, proporcionando una base sólida para el desarrollo de herramientas futuras que optimicen la prevención de accidentes en el ámbito ferroviario.

II. Resumen Ejecutivo

II-A. Antecedentes

II-A1. Un Sistema de Seguridad y Prevención de Colisiones en Tiempo Real para Trenes:

Resumen: El artículo presenta un sistema integral de seguridad que combina diversas tecnologías avanzadas para la prevención de colisiones en el transporte ferroviario. Reconoce que, a pesar de las mejoras en la infraestructura y las técnicas de gestión, los peligros persisten, lo que justifica la implementación de un enfoque más proactivo. El sistema propuesto utiliza datos en tiempo real para detectar y responder a situaciones de riesgo, con el fin de prevenir accidentes antes de que ocurran. Además, se subraya la importancia de la comunicación entre trenes, estaciones y otros elementos del sistema para asegurar una operación sincronizada y segura.

Puntos de interés para la investigación:

- **Detección y Respuesta en Tiempo Real:** Según Wu (2017), el sistema se basa en tecnologías de detección que permiten identificar situaciones peligrosas en tiempo real, activando respuestas automáticas como la reducción de velocidad o la detención total antes de que se produzca una colisión [5].

- **Integración de Tecnologías Avanzadas:** Wu (2017) destaca que se integran tecnologías como el monitoreo por GPS, sensores de proximidad y sistemas de comunicación entre trenes, lo que mejora la conciencia situacional y la capacidad de respuesta ante emergencias [5].
- **Comunicaciones Eficientes:** Según Wu (2017), la comunicación efectiva entre los trenes y las estaciones es clave para coordinar acciones y minimizar riesgos. El autor resalta la necesidad de protocolos de comunicación robustos para asegurar que la información crítica se transmita sin demora [5].
- **Beneficios Socioeconómicos:** Wu (2017) señala que, al reducir la frecuencia y severidad de los accidentes, el sistema no solo mejora la seguridad de los pasajeros y el personal, sino que también tiene beneficios económicos para las comunidades dependientes del transporte ferroviario, al mitigar pérdidas asociadas a accidentes y paradas operativas [5].
- **Evaluación Continua y Mejora del Sistema:** Según Wu (2017), el sistema incluye mecanismos para la evaluación continua y la mejora de los procedimientos de seguridad, asegurando que se mantenga actualizado frente a nuevas amenazas y desarrollos tecnológicos en el ámbito ferroviario [5].

II-A2. SafeCap: Un Sistema de Seguridad para la Prevención de Colisiones en Tráfico Ferroviario:

Resumen: El documento presenta el sistema SafeCap, diseñado específicamente para abordar la seguridad en las operaciones ferroviarias y evitar colisiones entre trenes. A pesar de las mejoras en la infraestructura ferroviaria y las tecnologías actuales, los accidentes siguen siendo una preocupación significativa. SafeCap utiliza sensores y tecnología de comunicación para proporcionar un monitoreo continuo del estado de los trenes y su entorno. El sistema puede detectar situaciones potencialmente peligrosas y activar alertas o intervenciones automáticas para prevenir colisiones.

Puntos de interés para la investigación:

- **Detección Proactiva de Riesgos:** Hann y Couch (2014) afirman que SafeCap implementa tecnologías de detección avanzada que permiten identificar riesgos en tiempo real, lo que facilita una respuesta rápida ante situaciones peligrosas y garantiza la seguridad de los viajeros y el personal [6].
- **Integración de Sensores y Sistemas de Comunicación:** Los autores destacan que el sistema hace uso de diversos sensores que recopilan datos críticos y los transmiten a través de redes seguras, asegurando que la información relevante esté disponible de manera inmediata, lo que mejora la toma de decisiones en situaciones de emergencia [6].
- **Intervenciones Automáticas:** Hann y Couch (2014) subrayan que una característica clave de SafeCap es la capacidad de llevar a cabo intervenciones automáticas, como frenar o desviar trenes, a fin de evitar accidentes. Esto reduce la dependencia de la intervención humana y el margen de error asociado [6].
- **Mejora Continua del Sistema:** Según los autores, SafeCap incluye un enfoque de mejora continua que permite al sistema aprender de incidentes previos y ajustarse para optimizar su rendimiento, asegurando que se mantenga actualizado frente a nuevas amenazas [6].
- **Impacto Social y Económico:** Hann y Couch (2014) mencionan que la implementación de SafeCap no solo incrementa la seguridad en el transporte ferroviario, sino que también tiene un impacto positivo en las comunidades, al reducir los costos económicos relacionados con accidentes y mejorar la confianza del público en el sistema ferroviario [6].

II-A3. Intel: Sistemas de Prevención de Colisiones en Trenes:

Resumen: El informe detalla la importancia de los sistemas de prevención de colisiones en el contexto del transporte ferroviario. A pesar de que el medio ferroviario es considerado uno de los más seguros, los accidentes continúan representando una amenaza significativa. Los sistemas de prevención de colisiones utilizan tecnología de monitoreo y comunicación para identificar y mitigar riesgos en tiempo real, permitiendo la intervención automática ante situaciones de peligro. Esto no solo ayuda a evitar accidentes, sino que también optimiza las operaciones del servicio ferroviario.

Puntos de interés para la investigación:

- **Monitoreo en Tiempo Real:** Intel (2022) explica que los sistemas de prevención de colisiones se basan en tecnologías de monitoreo en tiempo real, lo que permite la evaluación constante de las condiciones operativas y la detección de situaciones de riesgo, asegurando respuestas rápidas ante emergencias [7].
- **Interconexión de Sistemas:** El autor enfatiza la importancia de la interconexión entre diferentes componentes del sistema ferroviario, como señales, trenes y centros de control, para una comunicación efectiva. Esta integración es esencial para coordinar acciones preventivas y garantizar una respuesta ágil ante cualquier riesgo [7].
- **Intervenciones Automáticas:** Según Intel (2022), los sistemas son capaces de implementar intervenciones automáticas, como el frenado o cambios de ruta, cuando se detectan condiciones peligrosas. Esto reduce la dependencia del operador humano y minimiza el margen de error [7].
- **Compatibilidad con Nuevas Tecnologías:** Intel (2022) también destaca cómo los sistemas de prevención de colisiones pueden integrarse con tecnologías emergentes, como inteligencia artificial y análisis de datos, para mejorar su eficacia y adaptabilidad ante nuevos desafíos [7].
- **Beneficios Económicos y Sociales:** El autor señala que la reducción de accidentes no solo tiene un impacto positivo en la seguridad de los usuarios, sino que también contribuye a la eficiencia económica del transporte ferroviario, disminuyendo costos asociados a paradas no planificadas y daños. Esto fomenta la confianza pública en los viajes en tren, beneficiando a las comunidades y economías locales [7].

III. Requerimientos funcionales

III-1. Detección de ocupación de vías: El sistema debe ser capaz de detectar con precisión si una vía está ocupada por un tren u objeto, y activar automáticamente los semáforos correspondientes para evitar que otro tren entre en una vía ocupada.

III-2. Cálculo de distancia entre trenes: El sistema debe medir la distancia entre trenes en circulación para garantizar que se mantenga una distancia de seguridad adecuada. Si la distancia se reduce a niveles peligrosos, el sistema debe alertar o activar mecanismos de frenado.

III-3. Alerta de ajuste de velocidad: El sistema debe ser capaz de enviar alertas de ajuste de velocidad a los operadores cuando las condiciones de la vía, como la proximidad a otros trenes, la ocupación de las vías o factores relevantes, lo requieran.

III-4. Cálculo de la distancia mínima entre dos trenes: El sistema debe calcular la distancia mínima entre dos trenes en función de la distancia de frenado de cada uno y la superposición de sus trayectorias. Este cálculo debe tener en cuenta las capacidades de frenado, la velocidad de los trenes y las condiciones de operación para garantizar la seguridad.

III-5. Verificación en tiempo real: El sistema debe ser capaz de verificar en tiempo real las decisiones tomadas por los sensores y los semáforos, asegurándose de que las acciones se basen en datos correctos y precisos. Debe proporcionar informes de error si se detectan inconsistencias o fallos.

IV. Objetivo general de la investigación

Diseñar un modelo formal con VDM++ para estructurar y analizar la información relacionada con los trenes y sus sistemas de control, mejorando la seguridad ferroviaria.

V. Objetivo general de la investigación

Diseñar un modelo formal con VDM++ para estructurar y analizar la información relacionada con los trenes y sus sistemas de control, mejorando la seguridad ferroviaria.

V-A. Objetivos Específicos

- Crear un modelo formal en VDM++ que organice y represente de manera precisa los datos de ocupación de vías y distancias entre trenes, asegurando la coherencia de la información en el análisis de la seguridad ferroviaria.
- Diseñar un procedimiento formal que evalúe la coherencia temporal de los eventos en el sistema de prevención de colisiones, verificando que las decisiones, como el ajuste de velocidad, se realicen en un orden lógico y consistente, y detectando posibles discrepancias en la comunicación del sistema.
- Desarrollar un método de validación que garantice que la información sobre la ocupación de vías y el estado del sistema no haya sido alterada ni corrompida, asegurando que los datos utilizados en el análisis de seguridad sean válidos y confiables en todas las etapas del procedimiento.
- Evaluar el impacto del modelo desarrollado en la precisión y eficiencia de la prevención de colisiones ferroviarias, estableciendo las bases para el desarrollo de futuras herramientas especializadas que optimicen el procesamiento y análisis de datos de seguridad ferroviaria.

VI. Sensores en el Transporte Ferroviario

La seguridad en el transporte ferroviario es esencial para proteger a los pasajeros y las tripulaciones, así como para mantener la confianza de las comunidades que dependen de este medio de transporte. La implementación de tecnologías de sensores juega un papel crucial en la mejora de la seguridad ferroviaria mediante la comunicación entre trenes y la detección de objetos en las vías.

VI-A. 1. Sensores de Comunicación entre Trenes

Los sistemas de comunicación entre trenes son vitales para mantener la seguridad y la eficiencia en el transporte ferroviario. La implementación de sistemas como el **European Train Control System (ETCS)** permite la transmisión de datos en tiempo real sobre la posición y velocidad de los trenes [7].

VI-A1. Modelos y precios:

- **Siemens ETCS Onboard Unit**
  - **Precio:** Aproximadamente $40,000 - $60,000 USD por unidad, puede consultarse en la guía de costes del sistema ERTMS [8].
  - **Ubicación:** Instalado en la parte superior del tren, normalmente en la cabina de conducción.
- **Alstom CBTC System**
  - **Precio:** Estimaciones de mercado sugieren que el costo puede variar entre $500,000 y $1,000,000 USD, dependiendo de la escala del proyecto y las funcionalidades requeridas. Para detalles específicos, consultar cotizaciones oficiales [**?**], [9].
  - **Ubicación:** Instalado en los vagones y en las estaciones de control.

VI-B. 2. Sensores de Detección de Objetos

La detección de objetos es fundamental para evitar accidentes en las vías. Los sistemas de **LIDAR**, cámaras de visión artificial y otros sensores se utilizan para identificar obstáculos en tiempo real [10].

VI-B1. Modelos y precios:

- **Velodyne HDL-64E LIDAR Sensor**
  - **Precio:** Aproximadamente $75,000 USD por unidad, según información de mercado y distribuidores especializados [11].
  - **Ubicación:** Montado en la parte superior del tren para proporcionar un campo de visión amplio.
- **SmartCam AI**
  - **Precio:** Desde $3,500 hasta $6,000 USD por unidad, dependiendo del proveedor y especificaciones [12].
  - **Ubicación:** Instaladas en los laterales y en la parte frontal del tren para captar imágenes y video en

tiempo real, mejorando la vigilancia y detección de obstáculos o eventos anómalos. [13].

- **Tenaxx Ultrasonic Object Detection Sensor**
  - **Precio:** Aproximadamente $1,200 USD por unidad, consultar en el sitio oficial [14].
  - **Ubicación:** Integrado en la parte frontal y trasera del tren para detectar objetos a corta distancia.

Con estos avances en tecnología de sensores, los sistemas ferroviarios pueden operar de manera más segura y eficiente, contribuyendo al bienestar de millones de usuarios en todo el mundo.

VII. Distancia de Frenado

La distancia de frenado es un parámetro esencial en la operación ferroviaria, ya que define el espacio necesario para que un tren pueda detenerse de forma segura desde una determinada velocidad. Según la metodología descrita por la Agencia Estatal de Seguridad Ferroviaria, el cálculo de esta distancia se basa en una deceleración constante durante el proceso de frenado y toma en cuenta varios factores, incluyendo la velocidad inicial del tren, la velocidad final deseada, la declividad de la vía, la respuesta del sistema de frenos y la inercia rotacional de las masas [15].

La fórmula general utilizada para determinar la distancia de frenado es la siguiente:

$$s_{grad} = v_0 \cdot t_e - \frac{1}{2} \cdot \frac{m_{st}}{m_{dyn}} \cdot g_n \cdot i \cdot t_e^2 + \left(v_0 - \frac{m_{st}}{m_{dyn}} \cdot g_n \cdot i \cdot t_e \right) \cdot t_e \tag{1}$$

donde v_0 es la velocidad inicial, t_e el tiempo de respuesta equivalente del freno, a_e la deceleración efectiva, i la pendiente longitudinal de la vía, y $\frac{m_{st}}{m_{dyn}}$ el inverso del coeficiente de inercia de las masas rotativas.

Este modelo, conocido como modelo ETCS, permite representar de forma precisa el comportamiento del tren durante el frenado y está alineado con los estándares europeos actuales [15]. Su aplicación permite calcular de manera estandarizada las distancias de frenado necesarias para distintas velocidades y condiciones operativas.

VIII. Metodología

La presente investigación se enmarca en el contexto de la creciente necesidad de mejorar la seguridad ferroviaria, con énfasis en la prevención de colisiones. La elección de datos sobre ocupación de vías y distancias entre trenes como fuente primaria responde a su relevancia en la identificación de situaciones de riesgo y en la mejora de la gestión del tráfico ferroviario.

Se emplean técnicas avanzadas de formalización de software, específicamente el uso de VDM++ y model checking, que garantizan la consistencia y precisión en el análisis. Estas herramientas permiten modelar de manera adecuada las relaciones temporales y los eventos registrados, asegurando que las evidencias sean confiables y verificables en el ámbito de la seguridad ferroviaria, así como establecer un precedente para una construcción adecuada de estrategias de prevención.

El diseño cualitativo y el enfoque formal elegido se justifica por su capacidad de estructurar información compleja, ordinal y fragmentada, maximizando su validez para establecer conclusiones en los resultados obtenidos y contribuir al desarrollo de procedimientos estandarizados en el análisis de datos de seguridad ferroviaria.

Método investigativo: El método empleado en esta investigación se centra en la interpretación y análisis de datos de ocupación de vías y distancias entre trenes. Estos datos se abstraen para facilitar su comprensión por sistemas computacionales y establecer relaciones entre ellos, permitiendo la formulación de un caso basado en las características específicas de cada tipo de registro, ahora definidos como clases. El objetivo principal es identificar patrones significativos y evaluar situaciones de riesgo asociadas a la operación ferroviaria.

Los datos que se tendrán en cuenta para ser analizados en el modelo provienen de fuentes pertinentes, como registros ferroviarios y documentos oficiales de investigaciones previas. En el modelo, estos datos son procesados mediante técnicas formales, utilizando modelos en VDM++ y herramientas de model checking, que están ligadas a estados dentro de las operaciones de cada una de las clases (registros), lo que permite evaluar la coherencia temporal y la validez de los eventos registrados.

Finalmente, se concluye que este método de enfoque cualitativo sobresale por su capacidad de abordar información incompleta y garantizar precisión en la reconstrucción de eventos, contribuyendo significativamente al desarrollo de procedimientos de seguridad más eficientes y adaptables, lo que representa una adecuada base para proceder con esta investigación.

Procedimiento de Investigación

En el repositorio de GitHub se tiene el código completo de la investigación, que está basado en el modelado formal de datos relacionados con la seguridad ferroviaria utilizando el lenguaje VDM++. [16]
A continuación, se presenta una imagen básica del modelado de clases que muestra las principales clases involucradas en la estructura del modelo. Esta imagen ilustra cómo se organizan las clases de datos en el modelo propuesto.

Figura 1. Modelo de datos generado por VDM

Clase `Alerta`: La clase `Alerta` representa una alerta del sistema con información relevante sobre su tipo, mensaje, fecha y hora de ocurrencia. Esta clase incluye los siguientes atributos privados:

- `idAlert`: Identificador numérico único de la alerta (tipo `nat1`).
- `typeAlert`: Tipo de la alerta como una secuencia de caracteres (`seq of char`).
- `messageAlert`: Mensaje descriptivo de la alerta (`seq of char`).

- `dateAlert`: Fecha de la alerta, representada como una tupla `(día, mes, año)` de tipo `nat1 * nat1 * nat1`.
- `hourAlert`: Hora de la alerta, representada como una tupla `(hora, minuto, segundo)` de tipo `nat * nat * nat`.

La clase proporciona un único constructor público:

```
Alerta: nat1 * seq of char * seq of
char * (nat1 * nat1 * nat1) * (nat
* nat * nat) ==> Alerta
```

El constructor inicializa todos los campos privados a partir de los valores proporcionados como argumentos:

- `identificadorAlerta` → `idAlert`
- `tipoAlerta` → `typeAlert`
- `mensageAlerta` → `messageAlert`
- `diaAlerta` → `dateAlert`
- `horaAlerta` → `hourAlert`

```
class Alerta
    instance variables
    private idAlert: nat1;
    private typeAlert: seq of char;
    private messageAlert: seq of char;
    private dateAlert: nat1 * nat1 * nat1;
    private hourAlert: nat * nat * nat;
    operations
    public Alerta: nat1 * seq of char * seq of char *
         (nat1 * nat1 * nat1) * (nat * nat * nat) ==>
        Alerta
    Alerta(identificadorAlerta, tipoAlerta,
         mensageAlerta, diaAlerta, horaAlerta) ==
     (idAlert := identificadorAlerta;
      typeAlert := tipoAlerta;
      messageAlert := mensageAlerta;
      dateAlert := diaAlerta;
      hourAlert := horaAlerta);

    -- Operaci n para obtener el ID de la alerta
    public getID: () ==> nat1
    getID() ==
        return idAlert;
end Alerta
```

Listing 1. Alerta.vpp

Clase `Sensor`: La clase `Sensor` representa un sensor genérico dentro del sistema, encapsulando información básica sobre su identidad y estado operativo. Esta clase incluye los siguientes atributos protegidos:

- `idSensor`: Identificador numérico único del sensor (tipo `nat1`).
- `assetSensor`: Indicador booleano del estado del sensor: `true` si está activo, `false` en caso contrario.

La clase proporciona un constructor público:

```
Sensor: nat1 * bool ==> Sensor
```

El constructor inicializa los campos protegidos con los valores proporcionados como argumentos:

- `identificadorSensor` → `idSensor`
- `activoSensor` → `assetSensor`

```
class Sensor
    instance variables
    protected idSensor: nat1;
    protected assetSensor: bool;
    protected alertas: map nat1 to Alerta := {|->}; --
        Map vac o inicialmente

    operations
    public Sensor: nat1 * bool ==> Sensor
    Sensor(identificadorSensor, activoSensor) ==
     (idSensor := identificadorSensor;
      assetSensor := activoSensor;
      alertas := {|->});

    -- Operaci n para a adir una alerta
    public addAlerta: Alerta ==> ()
    addAlerta(alerta) ==
    (
        alertas := alertas ++ {alerta.getID() |->
            alerta}
    )
    pre alerta.getID() not in set dom alertas;

    -- Operaci n para obtener el ID del sensor
    public getID: () ==> nat1
    getID() ==
        return idSensor;
end Sensor
```

Listing 2. Sensor.vpp

Clase `SensorDeProximidad`: La clase `SensorDeProximidad` es una subclase de `Sensor` y extiende sus funcionalidades al incluir información sobre la detección y el objeto detectado. Esta clase introduce dos nuevas variables de instancia privadas:

- `detection`: Estado de la detección, indicando si el sensor ha detectado un objeto (tipo `bool`).
- `thatIsDetected`: Descripción del objeto detectado, representada como una secuencia de caracteres (`seq of char`).

La clase incluye un constructor público:

```
SensorDeProximidad: nat1 *
bool * bool * seq of char ==>
SensorDeProximidad
```

El constructor recibe los siguientes parámetros:

- `identificadorSensor` → `idSensor`
- `activoSensor` → `assetSensor`
- `deteccion` → `detection`
- `queSeDetecta` → `thatIsDetected`

```
class SensorDeProximidad is subclass of Sensor
instance variables
    private detection: bool;
    private thatIsDetected: seq of char;

operations
    public SensorDeProximidad: nat1 * bool * bool * seq
         of char ==> SensorDeProximidad
    SensorDeProximidad(identificadorSensor,
         activoSensor, deteccion, queSeDetecta) ==
    (
        idSensor := identificadorSensor;
        assetSensor := activoSensor;
        detection := deteccion;
        thatIsDetected := queSeDetecta
    );
end SensorDeProximidad
```

Listing 3. SensorDeProximidad.vpp

Clase `SensorEntreTrenes`: La clase `SensorEntreTrenes` es una subclase de `Sensor` que extiende su funcionalidad al incluir información adicional sobre la distancia con el tren delantero y la existencia de otro tren en la vía. Esta clase contiene las siguientes variables de instancia privadas:

- `distanceWithFrontTrain`: Distancia con el tren delantero, medida en unidades reales (tipo `real`).

- existsTrain: Estado que indica si existe un tren en la vía (tipo bool).

El constructor público de la clase SensorEntreTrenes se define como:

```
SensorEntreTrenes: nat1 * bool *
real * bool ==> SensorEntreTrenes
```

El constructor inicializa las variables de instancia a partir de los parámetros proporcionados:

- identificadorSensor → idSensor
- activoSensor → assetSensor
- distanciaConTrenDelantero → distanceWithFrontTrain
- existeTren → existsTrain

```
class SensorEntreTrenes is subclass of Sensor
instance variables
    private distanceWithFrontTrain: real;
    private existsTrain: bool;

operations
    public SensorEntreTrenes: nat1 * bool * real * bool
         ==> SensorEntreTrenes
    SensorEntreTrenes(identificadorSensor,
         activoSensor, distanciaConTrenDelantero,
         existeTren) ==
    (
        idSensor := identificadorSensor;
        assetSensor := activoSensor;
        distanceWithFrontTrain :=
             distanciaConTrenDelantero;
        existsTrain := existeTren
    );
end SensorEntreTrenes
```

Listing 4. SensorEntreTrenes.vpp

Clase SistemaControl: La clase SistemaControl representa el sistema de control que gestiona el funcionamiento del conjunto de sensores y alertas. Su única variable de instancia es privada:

- name: Nombre del sistema de control, representado como una secuencia de caracteres (seq of char).

La clase incluye un constructor público:

```
SistemaControl: seq of char ==>
SistemaControl
```

El constructor inicializa la variable de instancia name con el valor proporcionado:

- nombre → name

```
class SistemaControl
instance variables
    private name: seq of char;
    private trenes: map nat1 to Tren := {|->}; -- Map
         vac o inicialmente

operations
    public SistemaControl: seq of char ==>
         SistemaControl
    SistemaControl(nombre) ==
    (
        name := nombre;
        trenes := {|->}
    );

    -- Operaci n para a adir un tren al sistema
    public addTren: Tren ==> ()
    addTren(tren) ==
    (
        trenes := trenes ++ {tren.getID() |-> tren}
    )
    pre tren.getID() not in set dom trenes;

    -- Operaci n para obtener un tren espec fico
    public getTren: nat1 ==> [Tren]
    getTren(id) ==
        if id in set dom trenes
        then return trenes(id)
        else return nil;
end SistemaControl
```

Listing 5. SistemaControl.vpp

Clase Tren: La clase Tren representa un tren en el sistema. Contiene información sobre su identificador, tipo, longitud y distancia de seguridad. Las variables de instancia de la clase son privadas:

- idTrain: Identificador numérico único del tren (tipo nat1).
- typeTrain: Tipo de tren, representado como una secuencia de caracteres (seq of char).
- lengthTrain: Longitud del tren en unidades reales (tipo real).
- distanceSafetyTrain: Distancia de seguridad del tren, también representada en unidades reales (tipo real).

El constructor público de la clase Tren es:

```
Tren: nat1 * seq of char * real *
real ==> Tren
```

Este constructor inicializa las variables de instancia con los valores proporcionados:

- identificadorTren → idTrain
- tipoTren → typeTrain
- longitudTren → lengthTrain
- distanciaSeguridadTren → distanceSafetyTrain

```
class Tren
  instance variables
    private idTrain: nat1;
    private typeTrain: seq of char;
    private lengthTrain: real;
    private distanceSafetyTrain: real;
    private sensors: map nat1 to Sensor := {|->}; --
         Map para almacenar exactamente 2 sensores

  operations
    public Tren: nat1 * seq of char * real * real ==>
         Tren
    Tren(identificadorTren, tipoTren, longitudTren,
         distanciaSeguridadTren) ==
     (idTrain := identificadorTren;
      typeTrain := tipoTren;
      lengthTrain := longitudTren;
      distanceSafetyTrain := distanciaSeguridadTren;
      sensors := {|->});

    -- Operaci n para asignar un sensor al tren
    public addSensor: Sensor ==> ()
    addSensor(sensor) ==
    (
        sensors := sensors ++ {sensor.getID() |->
             sensor}
    )
    pre card dom sensors < 2 and sensor.getID() not in
         set dom sensors;

    -- Operaci n para obtener el ID del tren
    public getID: () ==> nat1
    getID() ==
        return idTrain;

    -- Operaci n para verificar que el tren tiene
         exactamente 2 sensores
    public hasCorrectNumberOfSensors: () ==> bool
    hasCorrectNumberOfSensors() ==
        return card dom sensors = 2;
```

```
end Tren
```

Listing 6. Tren.vpp

VIII-A. Análisis de cobertura

El análisis de cobertura en VDM++ nos permitirá evaluar qué tan exhaustivamente se han probado las especificaciones formales desarrolladas. Este análisis es fundamental para garantizar que el modelo formal del sistema de prevención de colisiones ferroviarias cubra todos los escenarios críticos de seguridad.

VIII-A1. Descripción del Modelo: El presente modelo es de naturaleza estática y está diseñado con el propósito de verificar el correcto funcionamiento de las clases que conforman el sistema de control ferroviario. Para ello, se implementa una clase llamada `TestSuite`, que agrupa un conjunto de operaciones de prueba unitarias e integradas orientadas a validar la creación de objetos, el cumplimiento de precondiciones y la interacción entre componentes.

La clase `TestSuite` contiene las siguientes funciones:

- **testAlerta**: Verifica la correcta creación de instancias de la clase `Alerta` y la obtención de sus identificadores.
- **testSensor**: Evalúa el funcionamiento básico de la clase `Sensor`, incluyendo la adición de alertas y la obtención del identificador del sensor.
- **testSensorDeProximidad**: Comprueba la correcta construcción de sensores de proximidad con distintos parámetros y la asociación con alertas.
- **testSensorEntreTrenes**: Realiza pruebas sobre sensores entre trenes, asegurando su correcta inicialización y el manejo de alertas.
- **testSistemaControl**: Verifica la funcionalidad del sistema de control para agregar y recuperar trenes, incluyendo casos donde se busca un tren no existente.
- **testTren**: Evalúa la creación de objetos `Tren`, así como la asociación de sensores de distintos tipos a estos trenes.
- **testIntegracion**: Ejecuta una prueba de integración donde se instancian múltiples objetos del sistema, se conectan entre sí y se verifica su estado global.
- **testCasosLimite**: Revisa el comportamiento ante valores límite, como fechas extremas o combinaciones mínimas de sensores y alertas.
- **runAllTests**: Método principal que invoca todas las pruebas anteriores y retorna `true` si todas se ejecutan correctamente.

Esta clase de prueba permite asegurar la consistencia del modelo estático antes de integrarlo a un entorno dinámico o a una simulación más compleja.

```
class TestSuite

operations

-- Test para la clase Alerta
public testAlerta: () ==> bool
testAlerta() ==
(
    dcl alerta1: Alerta := new Alerta(1, "Emergencia",
        "Colision detectada", mk_(15, 5, 2024),
        mk_(14, 30, 0));
    dcl alerta2: Alerta := new Alerta(2, "Advertencia",
        "Velocidad excesiva", mk_(16, 5, 2024),
        mk_(10, 15, 30));
    dcl alerta3: Alerta := new Alerta(3, "Info",
        "Mantenimiento programado", mk_(17, 5, 2024),
        mk_(8, 0, 0));

    -- Verificar que se pueden crear alertas
        correctamente
    return alerta1.getID() = 1 and alerta2.getID() = 2
        and alerta3.getID() = 3
);

-- Test para la clase Sensor base
public testSensor: () ==> bool
testSensor() ==
(
    dcl sensor1: Sensor := new Sensor(101, true);
    dcl sensor2: Sensor := new Sensor(102, false);
    dcl alerta1: Alerta := new Alerta(1, "Emergencia",
        "Sensor activado", mk_(15, 5, 2024), mk_(14,
        30, 0));
    dcl alerta2: Alerta := new Alerta(2, "Advertencia",
        "Sensor desactivado", mk_(16, 5, 2024),
        mk_(10, 15, 30));

    -- A adir alertas a sensor1
    sensor1.addAlerta(alerta1);
    sensor1.addAlerta(alerta2);

    -- Verificar IDs de sensores
    return sensor1.getID() = 101 and sensor2.getID() =
        102
);

-- Test para la clase SensorDeProximidad
public testSensorDeProximidad: () ==> bool
testSensorDeProximidad() ==
(
    -- Corregir constructores usando los par metros
        correctos seg n la clase
    dcl sensorProx1: SensorDeProximidad := new
        SensorDeProximidad(201, true, true,
        "Ultrasonico");
    dcl sensorProx2: SensorDeProximidad := new
        SensorDeProximidad(202, false, false,
        "Infrarrojo");
    dcl sensorProx3: SensorDeProximidad := new
        SensorDeProximidad(203, true, true, "Laser");
    dcl alerta1: Alerta := new Alerta(10, "Proximidad",
        "Objeto detectado", mk_(15, 5, 2024), mk_(14,
        30, 0));

    -- A adir alerta
    sensorProx1.addAlerta(alerta1);

    -- Verificar IDs (no podemos verificar getType ya
        que no existe ese m todo)
    return sensorProx1.getID() = 201 and
        sensorProx2.getID() = 202 and
        sensorProx3.getID() = 203
);

-- Test para la clase SensorEntreTrenes
public testSensorEntreTrenes: () ==> bool
testSensorEntreTrenes() ==
(
    -- Corregir constructores usando los par metros
        correctos seg n la clase
    dcl sensorTren1: SensorEntreTrenes := new
        SensorEntreTrenes(301, true, 10.5, true);
    dcl sensorTren2: SensorEntreTrenes := new
        SensorEntreTrenes(302, false, 0.0, false);
    dcl sensorTren3: SensorEntreTrenes := new
        SensorEntreTrenes(303, true, 25.3, true);
    dcl alerta1: Alerta := new Alerta(20, "Distancia",
        "Tren muy cerca", mk_(15, 5, 2024), mk_(14,
        30, 0));

    -- A adir alerta
    sensorTren1.addAlerta(alerta1);

    -- Verificar IDs (no podemos verificar getType ya
        que no existe ese m todo)
    return sensorTren1.getID() = 301 and
        sensorTren2.getID() = 302 and
        sensorTren3.getID() = 303
);

```

```
-- Test para la clase SistemaControl
public testSistemaControl: () ==> bool
testSistemaControl() ==
(
    -- Corregir constructor que usa seq of char, no nat1
    dcl sistema1: SistemaControl := new SistemaControl("Sistema1");
    dcl sistema2: SistemaControl := new SistemaControl("Sistema2");
    dcl tren1: Tren := new Tren(1001, "Electrico", 100.0, 50.0);
    dcl tren2: Tren := new Tren(1002, "Diesel", 120.0, 60.0);

    -- A adir trenes a los sistemas para usar las variables
    sistema1.addTren(tren1);
    sistema2.addTren(tren2);

    -- Verificar que los trenes se a adieron correctamente Y que getTren devuelve nil para IDs inexistentes
    return sistema1.getTren(1001).getID() = 1001 and
        sistema2.getTren(1002).getID() = 1002 and
          sistema1.getTren(9999) = nil and
              sistema2.getTren(8888) = nil
);

-- Test para la clase Tren
public testTren: () ==> bool
testTren() ==
(
    -- Corregir constructores usando todos los par metros requeridos
    dcl tren1: Tren := new Tren(5001, "Electrico", 100.5, 50.0);
    dcl tren2: Tren := new Tren(5002, "Diesel", 120.0, 60.0);
    dcl tren3: Tren := new Tren(5003, "Hibrido", 110.0, 55.0);
    dcl sensor1: Sensor := new Sensor(501, true);
    dcl sensor2: Sensor := new Sensor(502, false);
    dcl sensorProx1: SensorDeProximidad := new SensorDeProximidad(503, true, true, "Ultrasonico");
    dcl sensorTren1: SensorEntreTrenes := new SensorEntreTrenes(504, true, 15.0, true);

    -- A adir sensores al tren1
    tren1.addSensor(sensor1);
    tren1.addSensor(sensor2);

    -- A adir sensores al tren2
    tren2.addSensor(sensorProx1);
    tren2.addSensor(sensorTren1);

    -- Verificar IDs de trenes
    return tren1.getID() = 5001 and tren2.getID() = 5002 and tren3.getID() = 5003
);

-- Test integrado que combina m ltiples clases
public testIntegracion: () ==> bool
testIntegracion() ==
(
    -- Crear sistema de control
    dcl sistemaControl: SistemaControl := new SistemaControl("SistemaIntegrado");

    -- Crear trenes con todos los par metros
    dcl trenElectrico: Tren := new Tren(1001, "Electrico", 150.0, 75.0);
    dcl trenDiesel: Tren := new Tren(1002, "Diesel", 180.0, 90.0);

    -- Crear sensores de diferentes tipos con par metros correctos
    dcl sensorBase1: Sensor := new Sensor(2001, true);
    dcl sensorBase2: Sensor := new Sensor(2002, false);
    dcl sensorProx1: SensorDeProximidad := new SensorDeProximidad(3001, true, true, "Ultrasonico");
    dcl sensorProx2: SensorDeProximidad := new SensorDeProximidad(3002, true, false, "Infrarrojo");
    dcl sensorTren1: SensorEntreTrenes := new SensorEntreTrenes(4001, true, 20.5, true);
    dcl sensorTren2: SensorEntreTrenes := new SensorEntreTrenes(4002, false, 0.0, false);

    -- Crear alertas
    dcl alerta1: Alerta := new Alerta(100, "Emergencia", "Colision inminente", mk_(20, 5, 2024), mk_(15, 45, 30));
    dcl alerta2: Alerta := new Alerta(101, "Advertencia", "Velocidad alta", mk_(20, 5, 2024), mk_(15, 46, 0));
    dcl alerta3: Alerta := new Alerta(102, "Info", "Sensor activado", mk_(20, 5, 2024), mk_(15, 46, 30));
    dcl alerta4: Alerta := new Alerta(103, "Proximidad", "Objeto detectado", mk_(20, 5, 2024), mk_(15, 47, 0));
    dcl alerta5: Alerta := new Alerta(104, "Distancia", "Distancia critica", mk_(20, 5, 2024), mk_(15, 47, 30));

    -- A adir alertas a sensores
    sensorBase1.addAlerta(alerta1);
    sensorBase2.addAlerta(alerta2);
    sensorProx1.addAlerta(alerta3);
    sensorProx2.addAlerta(alerta4);
    sensorTren1.addAlerta(alerta5);

    -- A adir sensores a los trenes
    trenElectrico.addSensor(sensorProx1);
    trenElectrico.addSensor(sensorTren1);
    trenDiesel.addSensor(sensorProx2);
    trenDiesel.addSensor(sensorTren2);

    -- A adir trenes al sistema
    sistemaControl.addTren(trenElectrico);
    sistemaControl.addTren(trenDiesel);

    -- Verificar que todo se c re  correctamente
    return trenElectrico.getID() = 1001 and
        trenDiesel.getID() = 1002 and
          sensorProx1.getID() = 3001 and
              sensorTren1.getID() = 4001
);

-- Test de casos l mite y precondiciones
public testCasosLimite: () ==> bool
testCasosLimite() ==
(
    dcl sensor1: Sensor := new Sensor(1, true);
    dcl sensor2: SensorDeProximidad := new SensorDeProximidad(2, false, false, "Test");
    dcl tren: Tren := new Tren(1, "Test", 50.0, 25.0);

    -- Crear alertas con IDs  nicos
    dcl alerta1: Alerta := new Alerta(1, "Test1", "Mensaje1", mk_(1, 1, 2024), mk_(0, 0, 0));
    dcl alerta2: Alerta := new Alerta(2, "Test2", "Mensaje2", mk_(31, 12, 2024), mk_(23, 59, 59));

    -- A adir alertas (debe funcionar)
    sensor1.addAlerta(alerta1);
    sensor2.addAlerta(alerta2);

    -- A adir sensores al tren
    tren.addSensor(sensor1);
    tren.addSensor(sensor2);

    -- Verificar que el tren tiene exactamente 2 sensores
    return tren.hasCorrectNumberOfSensors()
);

-- M todo principal que ejecuta todos los tests
public runAllTests: () ==> bool
runAllTests() ==
(
    dcl resultado1: bool := testAlerta();
    dcl resultado2: bool := testSensor();
    dcl resultado3: bool := testSensorDeProximidad();
    dcl resultado4: bool := testSensorEntreTrenes();
    dcl resultado5: bool := testSistemaControl();
    dcl resultado6: bool := testTren();
    dcl resultado7: bool := testIntegracion();
    dcl resultado8: bool := testCasosLimite();
```

```
196
197     return resultado1 and resultado2 and resultado3 and
            resultado4 and
198           resultado5 and resultado6 and resultado7 and
                resultado8
199 );
200
201 end TestSuite
```

Listing 7. SistemaControl.vpp

VIII-A2. Ejecución del análisis de cobertura: Para realizar el análisis de cobertura del modelo estático, se utilizó la herramienta `vdmde` junto con el comando `tcov`. A continuación, se detallan los comandos ejecutados en el entorno VDM++:

```
>> tcov reset
>> create test := new TestSuite()
>> print test.runAllTests()
>> tcov write test.tc
>> rtinfo test.tc
```

Las imágenes de cobertura generadas se presentan a continuación. Cada una corresponde a la clase TestSuite:

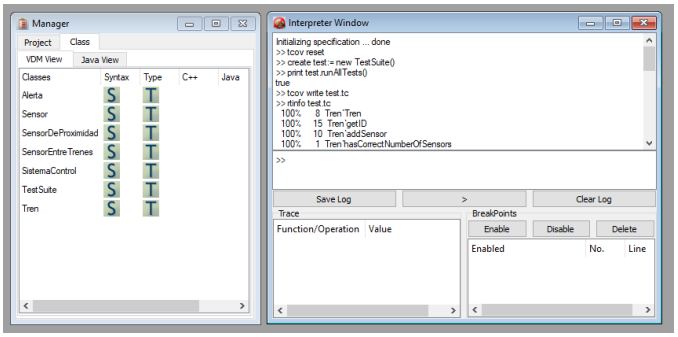

Figura 2. Esquema Análisis de cobertura

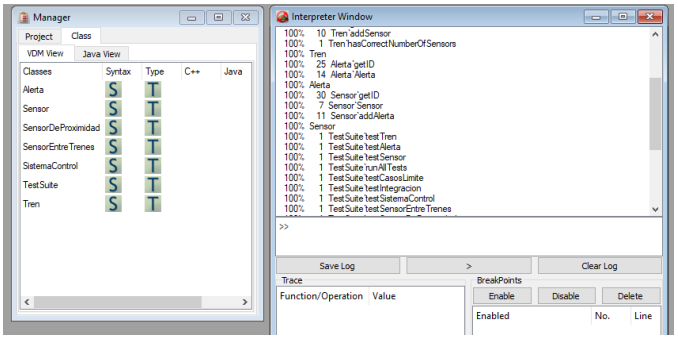

Figura 3. Esquema Análisis de cobertura

Se puede observar que todas las clases del modelo han sido completamente cubiertas por las pruebas. Esto indica un adecuado nivel de validación del comportamiento del sistema modelado.

VIII-B. Model Checking con UPPAAL

Para complementar el modelo formal desarrollado en VDM++, se implementó un modelo de verificación de tiempo real utilizando la herramienta UPPAAL. Este modelo permite analizar el comportamiento del sistema de control de tráfico

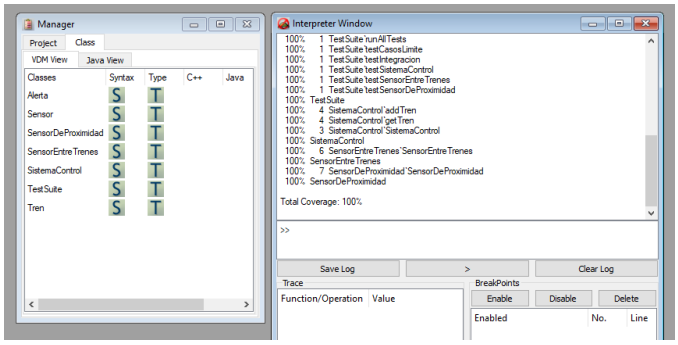

Figura 4. Esquema Análisis de cobertura

ferroviario considerando aspectos temporales y de concurrencia que son difíciles de capturar completamente en un modelo estático como VDM++.

VIII-B1. Descripción del Modelo: El modelo en UPPAAL consta de dos plantillas principales que interactúan mediante canales de comunicación síncronos:

- **Tren**: Representa el comportamiento del tren con tres estados principales:
  - `Velocidad_Normal`: Estado donde el tren mantiene una velocidad constante.
  - `Frenado`: Estado de frenado activo cuando se detecta un obstáculo o peligro.
  - `Acelerando`: Estado de aceleración cuando las condiciones son seguras.

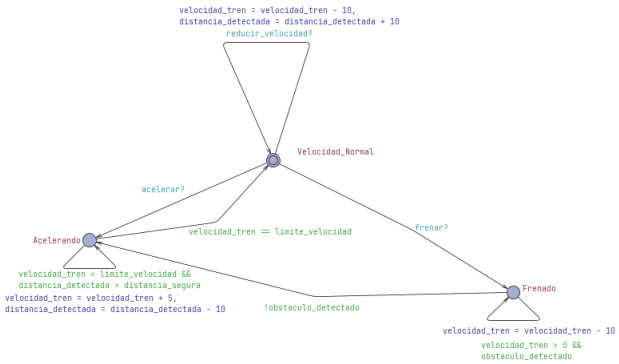

Figura 5. Esquema tren

- **SistemaControl**: Monitorea continuamente las condiciones del sistema cada 2 unidades de tiempo y envía señales al tren según sea necesario. Sus estados incluyen:
  - `Monitoreando`: Estado de observación continua con invariante temporal.
  - `Evaluando`: Estado comprometido donde se toman decisiones basadas en las condiciones actuales.

Las variables globales del sistema incluyen velocidad del tren, distancia detectada, límites de seguridad y señales de obstáculos, permitiendo un control integral del comportamiento ferroviario.

VIII-B2. Propiedades Verificadas: Se verificaron las siguientes propiedades de seguridad y vivacidad utilizando el verificador de modelos de UPPAAL:

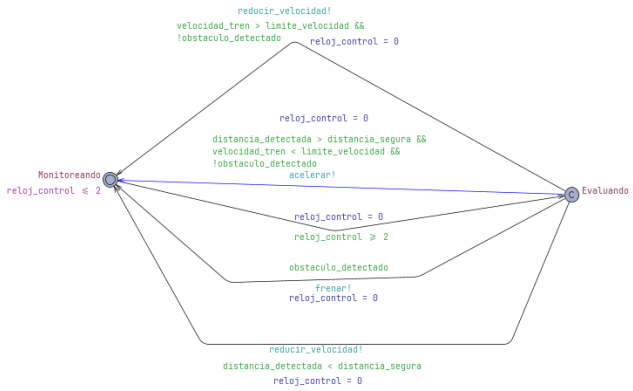

Figura 6. Esquema Monitor

VIII-B2a. Propiedades de Seguridad (Exitosas)::

1. **Seguridad de Velocidad No Negativa:**

```
A[] (velocidad_tren >= 0)
```

Resultado: ÉXITO - Garantiza que la velocidad del tren nunca sea negativa.

2. **Respuesta a Obstáculos Críticos:**

```
A[] (obstaculo_detectado and
     distancia_detectada < 50
     imply Tren.Frenado)
```

Resultado: ÉXITO - Cuando se detecta un obstáculo muy cerca, el tren entra en frenado.

3. **Control de Exceso de Velocidad:**

```
A[] (velocidad_tren >
     limite_velocidad
     imply SistemaControl.Evaluando)
```

Resultado: ÉXITO - Si se excede el límite, el sistema evalúa inmediatamente.

4. **Frenado Controlado:**

```
A[] (Tren.Frenado and
     obstaculo_detectado
     imply velocidad_tren <= 30)
```

Resultado: ÉXITO - Durante el frenado con obstáculo, la velocidad se mantiene controlada.

VIII-B2b. Propiedades de Vivacidad (Exitosas):: resume

1. **Detención Eventual ante Obstáculos:**

```
A<> (obstaculo_detectado
     imply velocidad_tren == 0)
```

Resultado: ÉXITO - Si se detecta obstáculo, eventualmente el tren se detiene.

2. **Capacidad de Recuperación:**

```
A<> (Tren.Frenado imply
     (Tren.Acelerando or
      Tren.Velocidad_Normal))
```

Resultado: ÉXITO - Después de frenar, el sistema puede recuperar operación normal.

VIII-B3. Análisis de Resultados: El modelo demostró robustez en las propiedades críticas de seguridad:

- Mantenimiento de invariantes de seguridad (velocidad no negativa)
- Respuesta correcta a situaciones de emergencia
- Control efectivo de límites de velocidad
- Capacidad de recuperación del sistema

VIII-B3a. Validación Complementaria:: Este análisis con UPPAAL complementa efectivamente el modelo VDM++ al:

- Verificar propiedades temporales y de concurrencia
- Identificar comportamientos emergentes del sistema
- Validar la lógica de control bajo diferentes escenarios
- Revelar limitaciones que requieren refinamiento del modelo

La combinación de ambos enfoques (VDM++ para especificación formal y UPPAAL para verificación temporal) proporciona una validación integral del sistema de control ferroviario, asegurando tanto la corrección lógica como el comportamiento temporal seguro.

Referencias

[1] European Union Agency for Railways, "Report on railway safety and interoperability in the eu 2024," 2024, acceso 2024. [Online]. Available: https://www.era.europa.eu/system/files/2024-06/Report-on-Railway-Safety-and-Interoperability-in-the-EU-2024.pdf

[2] D. Carrington, "The economic impact of rail accidents," *Rail Safety and Standards Board*, 2019, acceso 2019. [Online]. Available: https://rssb.co.uk

[3] Decel, "Revolutionizing railway safety: The impact of advanced deceleration systems," dec 2021, accessed 2024. [Online]. Available: https://www.decel.se/en/rail-insights/revolutionizing-railway-safety-the-impact-of-advanced-deceleration-systems

[4] C. S. C. Lum, J. S. J. Lim, and M. C. S. Teoh, "Automation in rail transportation: Evolution and challenges," *Transportation Research Part C: Emerging Technologies*, 2020, acceso 2020. [Online]. Available: https://goo.su/qgli

[5] C. Wu, "A real-time train safety and collision prevention system," *Proceedings of the International Railway Safety Council*, 2017, fall 2017. [Online]. Available: https://international-railway-safety-council.com/wp-content/uploads/2017/09/cai-wu-a-real-time-train-safety-and-collision-prevention-system.pdf

[6] D. Hann and P. Couch, "Safecap: A safety system for train collision prevention," in *Proceedings of the 2014 AdaEurope Conference*, 2014, fall 2014. [Online]. Available: https://www.ada-europe.org/conference2014/presentations/SafeCap.pdf

[7] Intel, "Train collision avoidance systems," jun 2022, fall 2022. [Online]. Available: https://www.intel.com/content/dam/www/central-libraries/us/en/documents/2022-06/train-collision-avoidance-sytems-brief.pdf

[8] European Commission, "Unit cost decision for cef funding: Ertms/etcs onboard unit," European Commission, Tech. Rep., 2021. [Online]. Available: https://ec.europa.eu/info/funding-tenders/opportunities/docs/2021-2027/cef/guidance/unit-cost-decision-cef-ertms-afif-evri-rfn_en.pdf

[9] Railway Technology, "How much does a cbtc system cost?" 2019. [Online]. Available: https://www.railway-technology.com/features/how-much-does-a-cbtc-system-cost/

[10] L. Zhang, "The utility of lidar for rail safety enhancements," *International Journal of Rail Safety*, 2022, accessed 2024. [Online]. Available: https://ijrail.com/lidar-utility

[11] LIDAR Market Insights, "Market report on automotive and railway lidar sensors," 2023, precio estimado para Velodyne HDL-64E. [Online]. Available: https://lidarmarketinsights.com/reports/2023/

[12] Avigilon. (2023) High-definition video surveillance cameras for rail and transportation. Acceso en 2024. [Online]. Available: https://www.avigilon.com/

[13] Moxa. (2024) Rail cctv systems: The road ahead. Acceso en 2024. [Online]. Available: https://www.moxa.com/en/articles/rail-cctv-systems-the-road-ahead
[14] Tenaxx, “Ultrasonic object detection sensors by tenaxx,” 2024, acceso en 2024. [Online]. Available: https://www.bxuansensor.com/ultrasonic-sensor
[15] Agencia Estatal de Seguridad Ferroviaria, “Especificación técnica de circulación: Cálculo de distancias de frenado,” Ministerio de Transportes, Movilidad y Agenda Urbana, Tech. Rep., 2023, versión 3.0, publicada el 20 de julio de 2023. [Online]. Available: https://www.seguridadferroviaria.es/recursos_aesf/etc_frenado_v.3.0.pdf
[16] R. Stranger, “mf-trafico-de-trenes,” https://github.com/RodrigoStranger/mf-trafico-de-trenes.git, 2025, fall 2025.
[17] MarketsandMarkets, “Railway signaling systems market by component, technology, application and region - global forecast to 2026,” 2021. [Online]. Available: https://www.marketsandmarkets.com/PressReleases/railway-signaling.asp