A Model-Based Approach for the Integration of Formal Verification Techniques and Simulation in Cyber-Physical Systems

María Julia Blas[1,2], Florencia Hiembuchner[2] and Silvio Gonnet[1,2]

[1] INGAR, CONICET-UTN, Avellaneda 3657, Santa Fe, Argentina
[2] Facultad Regional Santa Fe, UTN, Lavaisse 610, Santa Fe, Argentina
{mariajuliablas@santafe-conicet.gov.ar, fhiembuchner@frsf.utn.edu.ar, sgonnet@santafe-conicet.gov.ar}

Abstract. The development of reliable systems, particularly in safety-critical domains, requires the specification and verification of requirements. Formal methods provide mathematical precision, but present limitations when addressing complex behaviors. In contrast, simulation-based analysis is a pragmatic alternative that enables the evaluation of system behavior, without guaranteeing absolute correctness. This work presents a high-level framework that integrates formal verification models with simulation. The proposal establishes a link between these strategies through artifacts derived from a UML-based system description. Applied to Cyber-Physical Systems, the approach includes a UML profile to model the main components. The proposed modeling constitutes the first step of the framework, aiming to link formal verification and simulation as complementary strategies.

Keywords: complex systems modeling, UML profile, structural modeling.

Un Enfoque Basado en Modelos para la Integración de Técnicas de Verificación Formal y Simulación en Sistemas Ciberfísicos

Resumen. El desarrollo de sistemas confiables, especialmente en dominios de seguridad crítica, requiere la especificación y verificación de requerimientos. Los métodos formales ofrecen precisión matemática, pero presentan limitaciones ante comportamientos complejos. En contraste, el análisis basado en simulación es una alternativa pragmática que permite evaluar el comportamiento del sistema, sin garantizar correctitud absoluta. Este trabajo presenta un framework de alto nivel que integra modelos de verificación formal con simulación. La propuesta establece un vínculo entre estas estrategias mediante artefactos derivados de una descripción del sistema basada en UML. Aplicado a sistemas ciberfísicos, el enfoque incluye un perfil UML para modelar los componentes principales. El modelado propuesto constituye el primer paso del framework, con el objetivo de vincular verificación formal con simulación como estrategias complementarias.

Palabras clave: modelado sistemas complejos, perfil UML, modelo estructural.

1 Introducción

El enfoque más riguroso para el desarrollo de sistemas implica redactar requerimientos matemáticamente precisos del sistema deseado, diseñar modelos de los componentes del sistema junto con el entorno en el que se espera que opere, y utilizar herramientas de análisis para verificar que dicho modelo cumpla con los requerimientos (Haxthausen et al., 2015). Este enfoque, basado en modelos formales, resulta especialmente atractivo en sistemas de seguridad crítica. Sin embargo, el análisis basado en simulación como alternativa dentro del análisis formal, corresponde a un enfoque pragmático, que prioriza el análisis de comportamientos complejos como un conjunto de elementos que interactúan entre sí sabiendo que no puede garantizar correctitud absoluta (Alur, 2015).

Esta práctica permite estimar el grado de cumplimiento de los requerimientos definidos en relación con el diseño propuesto. Sin embargo, como se describe en la Sección 2.2, la aplicación de Modelado y Simulación (*Modeling and Simulation*, M&S) requiere de una representación del sistema bajo estudio (descripción del modelo). Esta descripción antecede al modelo formal y, por interpretación y/o compilación, se relaciona con el algoritmo de simulación que permite estimar el estado sucesor.

Este trabajo presenta un framework de alto nivel que permite vincular los diferentes tipos de análisis de verificación (demostración de invariantes, verificación automática de invariantes, y análisis basado en simulación) por medio de un conjunto de artefactos que parten de una descripción del modelo del sistema definida en UML. La aplicación del framework se da en el contexto de sistemas ciberfísicos (*Cyber Physical Systems,* CPS). Los CPSs refieren a sistemas complejos, diseñados, que aprovechan la computación embebida, el sensado y la comunicación en red para monitorear, coordinar, controlar e integrar dispositivos o procesos físicos (Zhang et al., 2023). En este contexto, un CPS queda definido por la combinación de infraestructura de naturaleza cibernética (espacio cibernético) y hardware (mundo físico), integrados con elementos de control que permiten detectar, controlar y accionar sobre el comportamiento del mundo físico, así como la interacción con las decisiones humanas. El perfil UML que se presenta en este trabajo parte de los componentes de alto nivel que comúnmente componen este tipo de sistemas y, redefine los vínculos entre los mismos a partir de los constructores del lenguaje UML. Tiene como objetivo representar la estructura del sistema bajo estudio, siguiendo los tipos de componentes comúnmente evidenciados en dichos sistemas.

Luego, el modelado propuesto constituye el primer paso del framework, con el objetivo de vincular verificación formal con simulación como estrategias alternativas en la evaluación de requerimientos. En este sentido, las contribuciones de este trabajo son: *1)* la definición de un framework de alto nivel que vincula los artefactos de modelado formal con la construcción de un modelo de simulación consistente, y *2)* la definición e implementación de una estrategia de modelado basada en el lenguaje UML para la construcción de una descripción del CPS basada en requerimientos.

El resto del trabajo se estructura de la siguiente manera. La Sección 2 introduce los fundamentos de la propuesta, detallando como se vincula el modelado formal con el análisis basado en simulación y describiendo el framework de alto nivel que enmarca la propuesta. La Sección 3 presenta el perfil UML diseñado como estrategia de modelado de la parte estructural de CPSs. La Sección 4 discute los resultados

alcanzados, haciendo una breve comparación con otras propuestas existentes. Finalmente, la Sección 5 está destinada a las conclusiones y trabajos futuros.

2 Modelado Formal y Simulación

2.1 Modelado Formal y CPS

El objetivo del modelado formal en la etapa de diseño es proveer una abstracción matemática que represente la complejidad de dicho diseño. En este sentido, se busca que el diseñador defina al sistema por medio de una semántica formal (matemática) que, posteriormente, permita dar respuesta preguntas como: *¿cuáles son los posibles comportamientos de un componente?* y *¿qué significa componer dos componentes?*

La actividad de modelado formal consta de dos tareas bien definidas: *especificación* y *análisis*. Durante la *especificación* el diseñador debe expresar los requerimientos asociados a la correctitud del sistema de forma matemática. Luego, debe construir un modelo formal T (matemático) que cumpla los requerimientos del sistema. Por su parte, durante el *análisis*, T es evaluado usando herramientas específicas a fin de estudiar si el sistema (mediante su modelo) satisface los requerimientos de seguridad (*safety requirements*). Estos requerimientos se definen en invariantes. Una invariante es una propiedad que permanece verdadera a lo largo de la ejecución de un sistema (Clarke et al., 2018) (es decir, se mantiene vigente a lo largo de todas las transiciones de estado).

Según (Alur, 2015), la tarea de *análisis* puede llevarse a cabo haciendo uso de una de las siguientes técnicas:

1) *Demostración de invariantes*: Se prueba por inducción matemática que las propiedades que han sido definidas como invariantes en un estado inicial de T se mantienen en todos los casos posibles.
2) *Verificación automática de invariantes*: Se construye una herramienta de verificación que, a partir del diseño propuesto en T y una propiedad específica, es capaz de determinar si la propiedad es una invariante del diseño.
3) *Análisis basado en simulación*: Se utiliza un algoritmo de simulación que permite explorar la evolución del sistema en una cantidad definida de pasos a partir de una representación de T. En el caso de situaciones no determinísticas, el algoritmo toma una decisión aleatoria respecto a cómo continuar.

Si bien *1)* es la técnica de propósito general para el análisis de invariantes, *3)* es la técnica más utilizada a nivel industria (Alur, 2015). Esto se debe a que, si bien los métodos formales tienen la ventaja de garantizar la correctitud, no son prácticos para ser aplicados a sistemas complejos de gran escala (Lee et al., 2015). Este es el caso de los CPS, los cuales se definen como una colección de dispositivos que se comunican entre sí e interactúan con el mundo real mediante sensores y actuadores en un lazo de retroalimentación. Mas aún, es normal que el análisis del diseño de un CPS (que, tal como se ha definido, presenta una combinación de comportamientos continuos y discretos complejos), se realice por medio de simulaciones numéricas (Bak et al., 2017).

En este punto, es importante destacar que la exploración mediante simulación no puede probar que el sistema satisface sus invariantes (ya que no puede explorar todos los posibles estados, sino sólo los que se encuentran al ejecutar una cantidad predefinida de pasos de simulación con elecciones aleatorias del simulador). Sin embargo, este tipo

de análisis permite ganar confianza en el diseño propuesto si se ejecutan múltiples corridas del algoritmo de simulación (con la misma parametrización) y no se detectan diferencias respecto al cumplimiento de las invariantes. Esto es, se realiza una prueba por falsificación. Adicionalmente, el uso de simulación facilita la realización de un análisis de performance del sistema bajo diseño en etapas tempranas del proyecto, permitiendo evaluar su funcionamiento en diferentes escenarios (por ejemplo, utilizando distintos componentes físicos y/o modificando su ubicación en el sistema) (Ait-Alla et al., 2019). Esto, a su vez, requiere que los diseñadores tengan conocimientos de simulación, siendo capaces de representar este tipo de situaciones (Hehenberger et al., 2016).

2.2 Análisis basado en Simulación: ¿Cómo Obtener el Modelo de Simulación?

Como se ha detallado con anterioridad, tanto *1)* como *2)* requieren de T para realizar el análisis del diseño. En estos casos, por ejemplo, T puede encontrarse definido siguiendo la descripción formal de una red de Petri sobre la cual puede realizarse un análisis (como, por ejemplo, la formulación del grafo de alcanzabilidad) o el uso de herramientas como análisis estructural (Reisig, 2013). Sin embargo, en el caso de *3)*, es necesaria una *representación de T*. Esto se debe a que la ejecución del algoritmo de simulación requiere de una representación del sistema que le permita encontrar el siguiente estado en un contexto dado.

Sobre dicha representación, ya sea por interpretación o por compilación, es posible realizar *3)* (Figura 1). En el primer caso, los elementos incluidos en la descripción son interpretados según lo requiera el algoritmo de simulación. En el segundo caso, la descripción es compilada en un modelo ejecutable, el cual puede ser ejecutado por el algoritmo de simulación. Luego, el principal beneficio de usar compilación sobre interpretación está dado en la performance (ya que interpretar continuamente la descripción del modelo es altamente costoso) (Alur, 2015).

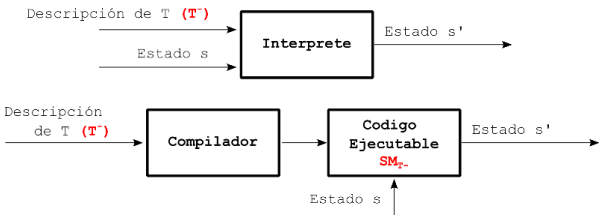

Fig. 1. Interpretación y compilación en análisis basado en simulación (adaptado de (Alur, 2015)).

En este contexto, surge naturalmente la pregunta *¿de qué manera los elementos incluidos en la representación de T pueden ser interpretados o compilados?* Según (Alur, 2015), *"la descripción original de T en el lenguaje de modelado original es interpretada/compilada"*. Esto es, debido a la imposibilidad de utilizar *T* en *3)*, se toma la *descripción del modelo T* (descripta en el lenguaje de modelado utilizado para diseñar el sistema, no necesariamente de manera formal), la cual llamaremos T^{-}, a los efectos de evaluar el estado sucesor en una simulación discreta.

Respecto a la Figura 1, es importante destacar que desde el punto de vista de la disciplina de M&S, sólo en la alternativa de compilación se materializa la existencia

del *modelo de simulación* asociado (el cual queda definido por el modelo ejecutable, al cual llamaremos SM_{T^-}). Aún con la posibilidad de interpretación y/o compilación, en la práctica es más sencillo (y menos tedioso) construir el modelo de simulación dinámica SM_{T^-} en base a T^-. Esta estrategia amplia el rango de lenguajes de modelado que pueden utilizarse para expresar T^-, ya que independiza la descripción de T de la propiedad de compilación y permite describir el CPS en términos de sus requerimientos funcionales y de calidad (Pohl, 2010). Mientras que los primeros son las acciones y tareas que el sistema debe ser capaz de realizar para satisfacer las necesidades de los distintos grupos de interés, los segundos definen las propiedades de calidad de todo el sistema (o de un componente, servicio o función) incluyendo atributos como rendimiento, seguridad, disponibilidad, usabilidad. En este segundo subconjunto de requerimientos (es decir, los requerimientos de calidad), pueden incluirse los requerimientos asociados a la correctitud del sistema (definidos en la Sección 2.1).

La Figura 2 presenta nuestra propuesta de estudio de CPSs como un framework de alto nivel, donde se observa que se vinculan múltiples artefactos por medio de relaciones bien establecidas. Cada uno de estos elementos se describe en la Tabla 1. Como punto de partida de este enfoque, en la Sección 3 se describe la estrategia de modelado basada en un perfil UML para la definición de T^- (esto es, el primer artefacto).

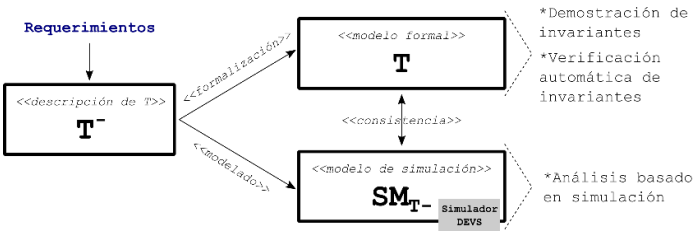

Fig. 2. Framework de alto nivel para la especificación y el análisis de CPSs basado en M&S.

2.3 DEVS como Formalismo para Especificar SM_{T^-}

Según lo indicado en (Clarke et al., 2018), un CPS es un sistema reactivo. Este tipo de sistemas puede ser estudiado como un sistema basado en transiciones, debido a que presenta una naturaleza síncrona de interacción entre componentes reactivos (Alur, 2015). Esta abstracción facilita además el proceso de pruebas, ya que al probar sistemas reactivos las entradas suelen consistir en trazas de entrada (es decir, secuencias finitas de vectores de entrada temporizados) que estimulan al sistema en un determinado momento durante la ejecución de la prueba (Haxthausen et al., 2015). Según (Alur, 2015), en sistemas basados en transiciones que modelan componentes reactivos, T^- se define como una composición estructural basada en componentes. Cada componente se encarga de resolver una o más tareas. A su vez, cada tarea detalla la actualización que produce, pudiendo indicar precondiciones de ejecución.

Por su parte, en una especificación Discrete Event System Specification (DEVS) (Zeigler et al., 2018), una secuencia temporal de eventos ingresa a un sistema provocando cambios instantáneos en su estado (Van Tendeloo et al., 2017). Estos eventos son generados: *a)* de forma externa (por otro modelo), o *b)* de forma interna (por el mismo modelo debido a que se ha cumplido el tiempo destinado a la actividad

asociada). De esta manera, los eventos externos ocurren cuando el estado actual del modelo es interrumpido por una acción externa, mientras que los eventos internos ocurren cuando expira el tiempo asignado al estado actual. En ambos casos, el siguiente estado se define según el estado anterior del sistema y el evento que ha tenido lugar. Entre eventos, el estado del sistema no cambia (es decir, no ocurren transiciones de estado entre eventos). Luego, DEVS facilita el modelado de sistemas basados en transiciones (Zeigler et al., 2018).

En este contexto, se propone utilizar DEVS como especificación de SM_{T^-}, a los efectos de representar el sistema de transición asociado al CPS con el objetivo de construir un modelo de simulación asociado. La dualidad de los modelos DEVS en términos de sus contrapartes formales y computacionales, brinda la posibilidad de definir la relación de *consistencia* entre artefactos, lo que representa un claro beneficio del framework propuesto. Tomando en consideración que DEVS permite representar tanto la estructura como el comportamiento (por medio de los modelos acoplados y atómicos, respectivamente), se establece la necesidad de capturar en T^- tanto las relaciones estructurales, como así también los esquemas de comportamiento de los componentes del CPS.

Tabla 1. Definición de los elementos que componen el framework propuesto en la Figura 1.

Tipo	Nombre	Descripción
Artefacto	T^-	Representación del CPS, expresada en un lenguaje de modelado que corresponde a una versión inicial del sistema, derivada directamente de los requerimientos.
Artefacto	T	Representación matemática precisa del sistema, utilizada para aplicar técnicas de verificación formal. Permite demostrar propiedades como invariantes.
Artefacto	SM_{T^-}	Representación ejecutable del sistema que permite analizar su comportamiento a través de simulaciones. No garantiza correctitud, pero permite explorar posibles escenarios.
Relación	Formalización	Proceso mediante el cual la descripción del sistema se traduce a un lenguaje matemático formal, eliminando ambigüedades.
Relación	Modelado	Proceso de construcción de una representación del sistema orientada a su análisis mediante simulación. Puede implicar abstracciones y decisiones de diseño.
Relación Derivada	Consistencia	Relación entre el modelo formal y el modelo de simulación que asegura que ambos representan el mismo sistema de manera coherente. Implica la no existencia de contradicciones.

3 Modelado UML de CPS como Descripción de T

3.1 Componentes de CPS

Cardin (2019) indica que la forma de delimitar la separación entre el mundo físico y el cibernético en un CPS es por medio de la identificación de tres tipos de componentes: *control*, *computación*, y *comunicación*. El *control* indica cuales son los medios por los cuales el mundo cibernético interactúa con el mundo real, tomando datos y actuando sobre los componentes físicos. La *computación* indica cómo el mundo cibernético

realiza los cómputos requeridos para transformar los datos del mundo físico en información de valor. Finalmente, la *comunicación* describe cómo se lleva a cabo el intercambio de datos e información entre el componente de control y el componente de computación. De acuerdo con esta separación, cada tipo de componente existe como un grupo de elementos que, en su conjunto, describen un aspecto básico del CPS. Asimismo, cada componente corresponde a una parte abstracta del ciberespacio.

De acuerdo con el comportamiento que exhiben, los CPS corresponden sistemas híbridos. Esto se debe a que combinan dinámicas discretas con continuas entre las que se da un control constante por retroalimentación del mundo físico (Alur, 2015). La *computación*, definida por software discreto, incluye componentes concurrentes que operan en múltiples modos de funcionamiento. Éstos interactúan con un entorno físico que evoluciona continuamente. Entre ambos, el *control* y la *comunicación* establecen los vínculos entre la continuidad del mundo real y la discretización del cibernético.

Adicionalmente, es usual que ambos mundos interactúen por medio de una copia del entorno físico dentro del entorno cibernético. Esta copia normalmente corresponde a algún tipo de modelo ubicado en el ámbito *computación*, el cual de forma genérica se denomina depósito de inteligencia (Cardin, 2019). Esto quiere decir que la inteligencia no está implementada físicamente, sino que surge de la conexión con un Gemelo Digital (Digital Twin, DT) del mismo. Luego, referiremos con el término DT a la parte de software encargada de llevar a cabo el control de un CPS por medio de una copia del mundo físico. El rol básico de este elemento es simular en el mundo virtual y predecir los posibles resultados de una acción en el mundo real (Negri et al., 2017); por lo que todo elemento del ámbito *computación* que no tenga como objetivo la predicción por medio de una copia no será considerado DT. Como es evidente, pueden existir otros componentes del ámbito de *computación* que no sean DTs. Tales componentes tendrán como responsabilidad el monitoreo y/o la detección de situaciones preestablecidas, controlando el accionar de los dispositivos físicos mediante comunicación en tiempo real a fin de procesar los datos que dichos dispositivos recolectan.

3.2 Modelado de CPS

Tal como se ha mencionado con anterioridad, en el contexto de CPSs el modelado resulta relevante debido a la naturaleza heterogénea de los componentes que integran este tipo de sistemas. La identificación clara de estos elementos y de sus interacciones contribuye a representar la estructura del CPS y, al mismo tiempo, comprender cómo se produce la integración entre el mundo físico y el digital. Con el objetivo de representar la estructura de un CPS en términos de los elementos descriptos en la Sección 3.1, se definió el perfil UML que se visualiza en la Figura 3.

Como complemento a la notación gráfica, se especificó un conjunto de invariantes expresadas en Object Constraint Language (OCL). Estas restricciones contribuyen a garantizar la validez de los modelos que usan el perfil, restringiendo el uso de estereotipos, reduciendo ambigüedades y habilitando la validación automática de modelos al implementarlos en herramientas de modelado. La Tabla 2 presenta el listado de restricciones OCL definidas.

Luego, como puede observarse en la Figura 3, a partir de un *Model* se redefine un *CPSModel*. Estos modelos se componen únicamente de tipos específicos redefinidos desde *Component*, *Dependency*, *Port*, *Interface* y *Class* (ver la restricción OCL#1):

CPSComponent, *CPSExternalComponent,CPSDependency*, *CPSPort*, *CPSInterface* y *CPSRequirement*.

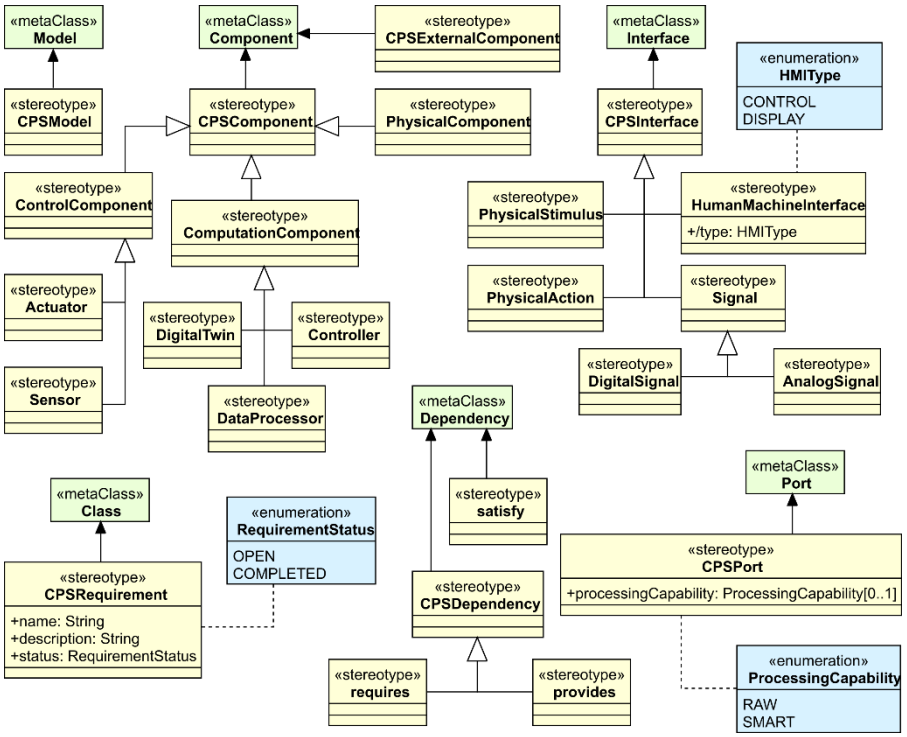

Fig. 3. Diseño del perfil UML para el modelado estructural de CPSs. Las metaclases resaltadas en verde corresponden a elementos de UML, mientras que los estereotipos resaltados en amarillo corresponden a las nuevas definiciones.

Como es evidente, se optó por tomar como base los elementos de UML utilizados en los diagramas de componentes, ya que se busca definir el CPS como el conjunto de componentes de distinta naturaleza que interactúan por medio de interfaces bien definidas. Así, sobre un *CPSComponent* se definen los principales tipos de componentes (*ControlComponent*, *ComputationComponent* y *PhysicalComponent*), refinando la generalización en elementos más específicos según el caso (como ser *Sensor* y *Actuator* para *ControlComponent*; y *DigitalTwin*, *DataProcessor*, y *Controller* para *ComputationComponent*).

En relación con *CPSInterface*, se especializa *HumanMachineInterface* (interacción entre CPS y usuarios humanos, como ser, visualización de información y aplicación de comandos), *Signal* (en sus versiones *AnalogSignal* y *DigitalSignal* para la transmisión de señales entre componentes), *PhysicalStimulus* (a través de la cual un componente recibe estímulos del entorno real), y *PhysicalAction* (mediante la cual un componente produce acciones sobre el entorno físico).

Para vincular los componentes y las interfaces, se propone el uso explícito de *CPSPort* y *CPSDependency*. El sentido de las dependencias indica el tipo de asociación definida, estableciendo para cada caso una vinculación en particular entre el componente involucrado y la interfaz asociada, a saber: *provides* indica que un *CPSComponent* expone una interfaz o servicio a otros componentes del sistema a través de un *CPSPort*; y *requires* indica que un *CPSComponent* consume una interfaz o servicio por medio de un *CPSPort*.

Finalmente, la definición de *CPSRequirement* se da como extensión de *Class* con el objetivo de indicar en el modelo los requerimientos que dan lugar al diseño propuesto. La vinculación entre *CPSRequirement* y *CPSComponent* se modela con *satisfy*.

Tabla 2. Restricciones OCL que se aplican sobre el perfil de la Figura 3.

#	**Especificación**
1	Un *CPSModel* solo puede contener elementos que han sido definidos en el perfil (todo elemento incluido en un *CPSModel* es un *CPSComponent, CPSExternalComponent*, *CPSInterface*, *CPSDependency*, *CPSPort*, o *CPSRequirement*).
	context CPSModel inv i1: self.ownedElement->forAll(e| e.oclIsKindOf(CPSComponent) or e.oclIsKindOf(CPSExternalComponent) or e.oclIsKindOf(CPSInterface) or e.oclIsKindOf(CPSDependency) or e.oclIsKindOf(CPSPort) or e.oclIsKindOf(CPSRequirement))
2	Un *CPSModel* debe tener al menos un elemento de cada tipo: *ComputationComponent*, *ControlComponent*, y *PhysicalComponent*.
	context CPSModel inv i2: self.ownedElement->exists(e | e.oclIsKindOf(ComputationComponent)) and self.ownedElement->exists(e | e.oclIsKindOf(ControlComponent)) and self.ownedElement->exists(e | e.oclIsKindOf(PhysicalComponent))
3	Un *CPSComponent* no puede estar aislado.
	context CPSComponent inv i3: self.ownedPort.supplierDependency.client->size() + self.ownedPort.clientDependency.supplier->size() > 0
4	Un *CPSExternalComponent* no puede estar aislado.
	context CPSExternalComponent inv i4: self.ownedPort.supplierDependency.client->size() + self.ownedPort.clientDependency.supplier->size() > 0
5	Un *CPSInterface* no puede estar aislado.
	context CPSInterface inv i5: self.clientDependency.supplier->size() + self.supplierDependency.client->size() > 0
6	Un *CPSRequirement* no puede estar aislado.
	context CPSRequirement inv i6: self.supplierDependency.client->notEmpty()
7	Un *CPSComponent* siempre debe estar asociado al menos a un *CPSRequirement*.
	context CPSComponent inv i7: self.clientDependency->select(s | s.oclIsTypeOf(satisfy)).supplier->notEmpty()
8	Un *CPSComponent* debe tener al menos un *CPSPort* (ya que para estar incluido en el modelo debe poder interactuar con otros componentes).
	context CPSComponent inv i8: self.ownedPort->notEmpty()

9	Un *CPSPort* debe participar en una *CPSDependency*. context CPSPort inv i9: self.supplierDependency->union(self.clientDependency)->select(d| d.oclIsKindOf(CPSDependency))->notEmpty()
10	Una *CPSInterface* tiene un único vínculo *provides* con un *CPSPort*. context CPSInterface inv i10: self.supplierDependency->select(d | d.oclIsTypeOf(provides))->size() = 1
11	Los vínculos de una *CPSInterface* que no son el vínculo *provides*, son *CPSDependency* del tipo *requires*. context CPSInterface inv i11: self.supplierDependency->union(self.clientDependency)->forAll(d | not d.oclIsTypeOf(provides) implies d.oclIsTypeOf(requires))
12	Una *CPSDependency* debe tener como origen un *CPSPort* y terminar en un *CPSInterface*. context CPSDependency inv i12: self.client->forAll(c | c.oclIsTypeOf(CPSPort)) and self.supplier->forAll(s | s.oclIsKindOf(CPSInterface))
13	Un *CPSComponent* no puede tener un *requires* sobre un *CPSInterface* que él mismo provee en un *provides*. context CPSComponent inv i13: self.ownedPort.clientDependency->select(d |d.oclIsTypeOf(provides)).supplier->intersection(self.ownedPort.clientDependency->select(d | d.oclIsTypeOf(requires)).supplier)->isEmpty()
14	Un *PhysicalStimulus* solo puede relacionar componentes desde un *PhysicalComponent* hacia un *ControlComponent*. context PhysicalStimulus inv i14: self.supplierDependency.client -> forAll(p | Component.allInstances()->forAll(c | c.ownedPort->includes(p) implies c.oclIsKindOf(ControlComponent)))
15	Un *PhysicalAction* solo puede relacionar componentes desde un *ControlComponent* hacia un *PhysicalComponent*. context PhysicalAction inv 15: self.supplierDependency.client->forAll(p | Component.allInstances()->forAll(c | c.ownedPort->includes(p) implies c.oclIsKindOf(ControlComponent)))
16	Un *HumanMachineInterface* solo puede relacionar componentes desde un *CPSExternalComponent* hacia un *CPSComponent*, y viceversa. context HumanMachineInterface inv i16: self.supplierDependency->select(d | d.oclIsTypeOf(provides)).client-> forAll(p | Component.allInstances()->forAll(c | c.ownedPort->includes(p) implies ((c.oclIsKindOf(CPSExternalComponent) and self.supplierDependency->select(d | d.oclIsTypeOf(requires)).client->forAll(pr | Component.allInstances()->forAll(cr | cr.ownedPort->includes(pr) implies cr.oclIsKindOf(CPSComponent)))) or (c.oclIsKindOf(CPSComponent) and self.supplierDependency->select(d | d.oclIsTypeOf(requires)).client->forAll(pr | Component.allInstances()->forAll(cr | cr.ownedPort->includes(pr) implies cr.oclIsKindOf(CPSExternalComponent)))))))
17	Si *HumanMachineInterface* relaciona componentes desde un *CPSExternalComponent* hacia un *CPSComponent*, entonces *type* debe ser *CONTROL*. context HumanMachineInterface inv i17: self.supplierDependency->select(d | d.oclIsTypeOf(provides)).client-> forAll(p | Component.allInstances()->forAll(c | (c.ownedPort->includes(p) and c.oclIsKindOf(CPSExternalComponent)) implies self.type = HMIType::CONTROL))

	Si *HumanMachineInterface* relaciona componentes desde un *CPSComponent* hacia un *CPSExternalComponent*, entonces *type* debe ser *DISPLAY*.
18	context HumanMachineInterface inv i18: self.supplierDependency->select(d | d.oclIsTypeOf(provides)).client->forAll(p | Component.allInstances()->forAll(c | (c.ownedPort->includes(p) and c.oclIsKindOf(CPSComponent)) implies self.type = HMIType::DISPLAY))
	El atributo *processingCapability* sólo lleva valor si el *CPSPort* provee un *Signal*.
19	context CPSPort inv i19: self.processingCapability <> null implies self.clientDependency.supplier->forAll(s|s.oclIsKindOf(Signal))
	La asociación *satisfy* relaciona un *CPSRequirement* (origen) con un *CPSComponent* (destino).
20	context satisfy inv i20: self.client->forAll(c | c.oclIsKindOf(CPSComponent)) and self.supplier->forAll(s | s.oclIsTypeOf(CPSRequirement))

Perfil UML vs. Metamodelado. Un perfil UML es un mecanismo de extensión del lenguaje que permite especializar el metamodelo de UML para dominios específicos mediante el uso de estereotipos, valores etiquetados y restricciones (OMG, 2017). Su definición permite extender el lenguaje de manera "ligera" y estandarizada, favoreciendo el reúso de herramientas y reduciendo la complejidad en comparación con los enfoques de metamodelado. Si bien un perfil UML no puede cambiar la semántica fundamental del lenguaje original, permite adaptar UML rápidamente a un dominio específico. Esto hace que los modeladores ya familiarizados con UML pueden usar un perfil sin necesidad de aprender un nuevo lenguaje.

En virtud de la amplia adopción de UML y de su capacidad para modelar sistemas basados en componentes de diferente naturaleza, se optó por definir un perfil UML que permita especializar el lenguaje al dominio de CPSs. Este enfoque ya ha sido aplicado en (Mallet, 2015).

3.3 Caso de Estudio

A los efectos de elaborar una prueba de concepto, la Figura 4 modela un sistema de control ferroviario como un CPS encargado de gestionar el acceso de trenes a un puente compartido (caso básico al trabajar con modelado formal). El sistema cuenta con sensores ubicados en los extremos oeste y este de las vías que le permiten saber cuándo existe un tren esperando en un extremo dado. Al mismo tiempo, existen semáforos en cada extremo los cuales habilitan/inhabilitan el paso por el puente. A partir de los datos sensados, el controlador toma decisiones en tiempo real y actúa sobre las luces del semáforo mediante actuadores, permitiendo que un tren ingrese al puente cuando es seguro hacerlo y bloqueando el acceso a otros trenes una vez que el puente está ocupado. De este modo, se busca garantizar la seguridad de la operación, evitando colisiones o situaciones de riesgo.

El perfil diseñado se está implementando en Papyrus, que es un entorno de modelado orientado al Desarrollo Dirigido por Modelos. Papyrus proporciona soporte para la definición y aplicación de perfiles UML, permitiendo extender el metamodelo del lenguaje original en base a la especificación de la Figura 3. Esta característica resulta

especialmente adecuada para el desarrollo del perfil propuesto, ya que posibilita incorporar conceptos específicos del dominio de los CPSs dentro de un entorno de modelado estandarizado.

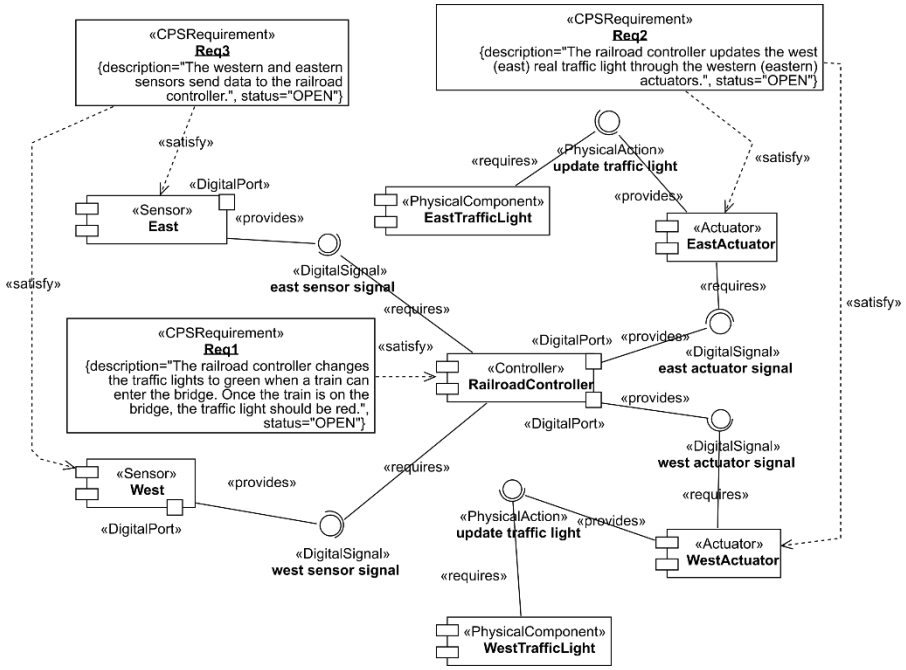

Fig. 4. Aplicación del perfil a una prueba de concepto clásica.

4 Discusión

Nuestra propuesta ha logrado definir estructuralmente múltiples CPSs en términos de sus componentes y las relaciones entre los mismos. Si bien el enfoque propuesto por (Mallet, 2015) se centra en el uso de un perfil UML (denominado MARTE) junto con CCSL para la especificación formal de restricciones temporales en CPS, el presente trabajo adopta una perspectiva diferente. En lugar de enfocarse en un formalismo específico para el modelado del tiempo, se propone un framework general que integra múltiples estrategias de verificación a partir de un diseño establecido. Mientras que MARTE/CCSL se orienta a la especificación detallada de aspectos temporales, nuestra propuesta se enfoca en el vínculo sistemático entre una descripción del sistema basada en UML y la posterior construcción de los modelos de simulación DEVS asociados (lo cual ya ha sido abordado en otros trabajos en dominios basados en lenguajes, como por ejemplo en Blas et al. (2020) y Dalmaso et al. (2023)). De este modo, el enfoque presentado prioriza la integración de técnicas de análisis más que la especialización en un único formalismo.

Desde otra perspectiva, el enfoque propuesto por (Lee et al., 2015) adopta una perspectiva de sistema de sistemas para el modelado formal de CPS, donde cada subsistema es representado mediante formalismos heterogéneos e integrado a través de una infraestructura de simulación distribuida. Si bien esta aproximación permite analizar la interacción entre subsistemas complejos, su foco principal está en la interoperabilidad y ejecución de modelos (un subconjunto de ellos basados en DEVS). En contraste, el presente trabajo se centra en la integración de distintas estrategias de verificación a partir de una descripción unificada del sistema en UML, orientada por requerimientos. De este modo, nuestra propuesta prioriza la trazabilidad y consistencia entre modelos y técnicas de análisis, en lugar de la composición heterogénea de subsistemas para simulación.

5 Conclusiones y Trabajos Futuros

Los enfoques centrados en la verificación de modelos formales son ideales para el análisis temprano del diseño de un CPS. Sin embargo, cuando el análisis por demostración de invariantes y verificación automática de invariantes no puede llevarse a cabo debido a la complejidad de los sistemas de gran escala, el análisis basado en simulación es la única alternativa viable. Si bien este tipo de análisis no puede garantizar correctitud, permite ganar confianza en el diseño propuesto al observarse el cumplimiento de las invariantes en múltiples corridas del algoritmo de simulación.

En este contexto, en este trabajo se han sentado las bases de un framework de alto nivel que vincula el modelado formal con el área de M&S, estableciendo los artefactos básicos y las relaciones de garantía que permiten aplicar todas las técnicas de análisis ante el diseño de un CPS específico. Luego, el perfil diseñado constituye una base estructurada que permite generar modelos de CPS bien definidos según una especificación conocida a nivel estructural capturando los requerimientos asociados a los componentes. Los constructores UML sobre los cuales se ha extendido el lenguaje corresponden a elementos estáticos (modelan qué es el sistema, no cómo funciona).

De esta manera, el perfil UML captura la complejidad del diseño estructural de un CPS como parte de la descripción de T (T^-). El diseño del comportamiento de T^- forma parte de los trabajos futuros. Se espera que este pueda definirse en términos del diseño basado en requerimientos a fin de, posteriormente, mapear la T^- a modelos DEVS (tanto en lo estructural como en el comportamiento). Luego, el modelado propuesto constituye el primer paso del framework, con el objetivo de vincular verificación formal con simulación como estrategias alternativas en la evaluación de requerimientos.

Referencias

Ait-Alla, A., Kreutz, M., Rippel, D., Lütjen, M., & Freitag, M. (2019). Simulation-based analysis of the interaction of a physical and a digital twin in a cyber-physical production system. *IFAC-PapersOnLine, 52*(13), 1331–1336.

Alur, R. (2015). *Principles of cyber-physical systems*. MIT Press.

Bak, S., & Duggirala, P. S. (2017). Rigorous simulation-based analysis of linear hybrid systems. En *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (pp. 555–572). Springer.

Blas, M. J., Leone, H., & Gonnet, S. (2020). Modeling and simulation framework for quality estimation of web applications through architecture evaluation. *SN Applied Sciences, 2*.

Cardin, O. (2019). Classification of cyber-physical production systems applications: Proposition of an analysis framework. *Computers in Industry, 104*, 11–21.

Clarke, E. M., Henzinger, T. A., Veith, H., & Bloem, R. (Eds.). (2018). *Handbook of model checking*. Springer.

Dalmasso, F., Blas, M. J., & Gonnet, S. (2023). Enriching UML statecharts through a metamodel: A model-driven approach for the graphical definition of DEVS atomic models. *IEEE Latin America Transactions, 21*, 27–34.

Haxthausen, A. E., & Peleska, J. (2015). Model checking and model-based testing in the railway domain. En *Formal modeling and verification of cyber-physical systems: 1st International Summer School on Methods and Tools for the Design of Digital Systems* (pp. 82–121). Springer Fachmedien Wiesbaden.

Hehenberger, P., Vogel-Heuser, B., Bradley, D., Eynard, B., Tomiyama, T., & Achiche, S. (2016). Design, modelling, simulation and integration of cyber-physical systems: Methods and applications. *Computers in Industry, 82*, 273–289.

Lee, K. H., Hong, J. H., & Kim, T. G. (2015). System of systems approach to formal modeling of CPS for simulation-based analysis. *ETRI Journal, 37*(1), 175–185.

Mallet, F. (2015). MARTE/CCSL for modeling cyber-physical systems. En *Formal modeling and verification of cyber-physical systems: 1st International Summer School on Methods and Tools for the Design of Digital Systems* (pp. 26–49). Springer Fachmedien Wiesbaden.

Negri, E., Fumagalli, L., & Macchi, M. (2017). A review of the roles of digital twin in CPS-based production systems. *Procedia Manufacturing, 11*, 939–948.

Object Management Group. (2017). *OMG Unified Modeling Language, Version 2.5.1*. OMG.

Pohl, K. (2010). Requirements Engineering: Fundamentals, Principles, and Techniques. Springer Berlin, Heidelberg.

Reisig, W. (2013). Understanding petri nets-modeling techniques. *Analysis Methods, Case Studies. Springer*.

Van Tendeloo, Y., & Vangheluwe, H. (2017). Classic DEVS modelling and simulation. En *Proceedings of the 2017 Winter Simulation Conference* (pp. 644–658).

Zeigler, B. P., Muzy, A., & Kofman, E. (2018). *Theory of modeling and simulation: Discrete event and iterative system computational foundations* (3rd ed.). Academic Press.

Zhang, K., Shi, Y., Karnouskos, S., Sauter, T., Fang, H., & Colombo, A. W. (2023). Advancements in industrial cyber-physical systems: An overview and perspectives. *IEEE Transactions on Industrial Informatics, 19*(1), 716–729.