Parsing DEVS Specifications into Metamodels: A Model-Driven Engineering Approach

Clarisa Espertino[1], María Julia Blas[2] and Silvio Gonnet[2]

[1]Facultad Regional Santa Fe, Universidad Tecnológica Nacional, Santa Fe, Santa Fe, Argentina
[2]Instituto de Desarrollo y Diseño INGAR, CONICET-UTN, Santa Fe, Santa Fe, Argentina

Abstract. Despite its formal nature, the development of executable Discrete Event System Specification (DEVS) models often relies on manual implementations in general-purpose programming languages. This can compromise the consistency between the formal specification and its realization. This work proposes a model-driven engineering–based strategy as part of the development of a DEVS computational language, which transforms textual DEVS specifications into structured models conforming to a set of interconnected metamodels. The solution combines parsing techniques with a systematic mapping mechanism that converts syntax trees into instances of metamodels implemented in EMF. These metamodels are integrated to capture the concrete syntax of the language while preserving modularity and reusability. The resulting architecture enables the automatic generation of DEVS metamodel instances from textual specifications, assisting their validation and enabling their future integration with other models.

Keywords: context-free grammar, EMF, OCL, modular metamodel.

De Especificaciones Textuales DEVS a Metamodelos: Un Enfoque de Ingeniería Dirigida por Modelos

Resumen. A pesar de su naturaleza formal, el desarrollo de modelos Discrete Event System Specification (DEVS) ejecutables suele depender de implementaciones manuales en lenguajes de propósito general. Esto puede comprometer la consistencia entre la especificación formal y su realización. Este trabajo propone una estrategia basada en ingeniería dirigida por modelos como parte del desarrollo de un lenguaje de computación DEVS, la cual transforma especificaciones textuales DEVS en modelos estructurados conforme un conjunto de metamodelos interconectados. La solución combina técnicas de análisis sintáctico con un mecanismo sistemático de mapeo que convierte árboles de sintaxis en instancias de metamodelos implementados en EMF. Estos metamodelos se integran para capturar la sintaxis concreta del lenguaje, manteniendo modularidad y reutilización. La arquitectura resultante permite la generación automática de instancias de los metamodelos DEVS a partir de especificaciones textuales, facilitando su validación y dando lugar a su integración, a futuro, con otros modelos.

Palabras clave: gramática libre de contexto, EMF, OCL, metamodelo modular.

1 Introducción

Discrete Event System Specification (DEVS) (Zeigler et al. 2018) es un formalismo ampliamente utilizado para el modelado y la simulación de sistemas de eventos discretos, basado en teoría de sistemas y teoría de conjuntos. Este formalismo se basa en una definición matemática que permite describir sistemas de manera modular y jerárquica mediante dos tipos de modelos: *modelos atómicos*, que capturan el comportamiento a través de estados y funciones de transición, *y modelos acoplados*, que representan la estructura del sistema mediante la composición de múltiples modelos. Desde el punto de vista formal, un modelo DEVS se define como una especificación matemática independiente de su implementación. Sin embargo, en la práctica, la construcción de las contrapartes ejecutables suele requerir de una implementación en lenguajes de programación de propósito general (Java, C++, Python, entre otros) asociados a herramientas de simulación (Cristiá et al. 2019). Esta situación introduce diversas limitaciones, ya que dificulta la verificación formal, reduce la interoperabilidad entre herramientas y genera una fuerte dependencia de plataformas específicas. Además, la brecha entre la especificación formal y su implementación computacional dificulta la evaluación de consistencia entre ambas representaciones (Sarjoughian et al. 2015).

En este contexto, surge la necesidad de contar con mecanismos que permitan representar modelos DEVS de manera estructurada, independiente de herramientas y alineada con su definición formal. En particular, las técnicas de Ingeniería Dirigida por Modelos (MDE por sus siglas en inglés) pueden utilizarse para definir una representación computacional de la notación formal DEVS que contribuya a la generación de modelos concretos de manera sistemática. En este trabajo se propone, haciendo uso de los fundamentos de MDE, una estrategia de transformación basada en la instanciación de metamodelos modulares a partir de las especificaciones textuales de modelos DEVS.

La principal contribución es el conjunto de metamodelos interrelacionados que dan soporte al modelo estructural de la especificación formal, junto con el proceso de transformación que los conecta con la sintaxis concreta del lenguaje definida en una gramática libre de contexto. Cada metamodelo del conjunto aplica a un subdominio particular del lenguaje (teoría de conjuntos, expresiones matemáticas, expresiones booleanas y el metamodelo estructural DEVS). Estos metamodelos se integran para representar la estructura de la sintaxis concreta, manteniendo la modularidad y reutilización.

El resto del artículo se organiza de la siguiente manera. En la Sección 2 se presenta la motivación que da contexto a la propuesta, la cual se encuentra relacionada a la definición de un lenguaje computacional DEVS que respete la notación formal de estos modelos. La Sección 3 describe la propuesta, incluyendo la sintaxis concreta adoptada, el proceso de mapeo y la arquitectura de metamodelos definidos para representar la sintaxis concreta. En la Sección 4 se presenta una prueba de concepto asociada a un modelo atómico. Finalmente, la Sección 5 expone las conclusiones y trabajos futuros.

2 Lenguaje Computacional DEVS

Un Lenguaje de Modelado Específico de Dominio (DSML por sus siglas en inglés) permite definir soluciones utilizando conceptos propios de un dominio particular,

reduciendo la brecha entre expertos del dominio y desarrolladores. Al igual que los lenguajes de programación, estos lenguajes necesitan ser diseñados para ser prácticos y utilizables (Paige et al. 2000). Un DSML se compone de tres elementos fundamentales (Krahn et al. 2007), presentados en la Tabla 1.

Tabla 1. Componentes de un DSML.

Componente	Descripción
Sintaxis abstracta	Describe la estructura lógica y las reglas de composición de los modelos que se pueden crear con el lenguaje. Se centra en los elementos y relaciones fundamentales sin considerar cómo se representarán visual o textualmente.
Sintaxis concreta	Define cómo se visualizan o escriben los elementos del modelo. Es decir, establece las notaciones específicas que se utilizarán para representar las construcciones de la sintaxis abstracta, por ejemplo, mediante texto.
Semántica	Describe el significado de las construcciones y modelos creados con el lenguaje. Es fundamental para comprender cómo debe interpretarse el modelo.

De acuerdo con Krahn et al. (2007) y Pons et al. (2010), el metamodelado permite definir la sintaxis abstracta de los lenguajes. Un metamodelo especifica los elementos, atributos y relaciones que pueden formar parte de un modelo, permitiendo su validación y manipulación automática dentro de MDE. Por otro lado, la sintaxis concreta de un DSML textual puede definirse mediante gramáticas libres de contexto (CFG), las cuales establecen las reglas de formación de las expresiones válidas del lenguaje (Hopcroft et al. 2007). A partir una CFG, herramientas como ANTLR permiten generar analizadores sintácticos que producen árboles de sintaxis en base a especificaciones textuales.

Un desafío relevante en este contexto es la integración entre la sintaxis concreta definida mediante gramáticas libres de contexto y la sintaxis abstracta representada por metamodelos. Si bien herramientas como ANTLR permiten generar árboles de sintaxis a partir de especificaciones textuales, y EMF facilita la definición de metamodelos, no existe un soporte directo que permita mapear automáticamente estas representaciones. En este trabajo se aborda esta problemática mediante la utilización de metamodelos intermedios que estructuran la sintaxis concreta y facilitan su transformación hacia modelos conformes a la sintaxis abstracta.

En este punto, una pregunta clave es: ¿para qué pretendemos usar un lenguaje de modelado formal DEVS? Los lenguajes de modelado se construyen para ser utilizados por diseñadores (no programadores) (Paige et al. 2000). En este sentido, el grupo de trabajo ha centrado sus esfuerzos en la construcción de un lenguaje de modelado de base computacional que permita, tanto a modeladores como a programadores DEVS, construir modelos (tanto atómicos como acoplados) siguiendo sus especificaciones formales de manera sencilla y obtener sus contrapartes computacionales de forma automática. Los fundamentos de los trabajos previos pueden consultarse en Blas et al. (2021, 2023a, 2023b, 2023c). La propuesta de este trabajo aborda uno de los problemas que surgen al momento de diseñar DSML, que es el mapeo de las tecnologías usadas para construir la CFG con los entornos de metamodelado utilizados para la definición de sintaxis abstractas. El objetivo es proporcionar una representación intermedia,

estructurada y validable de la especificación formal DEVS, que permita preservar la trazabilidad entre la notación matemática y los artefactos computacionales derivados.

Estos antecedentes abordan dos aspectos complementarios: por un lado, la definición de una representación computacional DEVS basada en metamodelado y, por otro, la especificación de una gramática libre de contexto para describir modelos DEVS clásicos. El presente trabajo se ubica entre ambos aportes, ya que se concentra en modelar la sintaxis concreta del lenguaje mediante un conjunto de metamodelos intermedios implementados en EMF. Dichos metamodelos permiten estructurar el resultado del análisis sintáctico y prepararlo para una etapa posterior de transformación hacia la sintaxis abstracta DEVS. De esta manera, la contribución no reemplaza los desarrollos previos, sino que los articula mediante un mecanismo de mapeo que permite obtener instancias estructuradas, validables y reutilizables a partir de especificaciones textuales.

3 Estrategia de Transformación

La Figura 1 presenta un esquema general de la transformación propuesta, observándose el flujo completo que parte de la especificación textual de un modelo DEVS creada por un modelador (artefacto #1) y finaliza con la construcción de una instancia del metamodelo CFG_DEVS (artefacto #2). La transformación que establece un puente entre la sintaxis concreta del lenguaje (reglas que rigen la creación del artefacto #1) y su representación basada en modelos (metamodelos que rigen la definición del artefacto #2) queda definida según un conjunto de *mappers* implementados como *listeners*.

3.1 Sintaxis Concreta CFG_DEVS

Tal como se indicó con anterioridad, se adoptó la gramática CFG_DEVS propuesta en Blas et al. 2023c, la cual permite describir modelos DEVS clásicos (atómicos y acoplados) a través de una notación textual basada en expresiones matemáticas. Esta gramática se estructura en múltiples módulos, cada uno enfocado en distintos aspectos del lenguaje, tales como la definición de modelos, teoría de conjuntos, expresiones matemáticas, expresiones booleanas y tokens léxicos. De esta manera, la especificación de CFG_DEVS organiza las reglas de producción según su dominio.

La elección de esta sintaxis concreta responde al propósito de mantener una correspondencia cercana con la forma en que los modelos DEVS son definidos en su especificación formal. En este sentido, la gramática no introduce una notación completamente ajena al dominio, sino que estructura expresiones ya presentes en el formalismo —tales como definiciones de conjuntos, funciones de transición, condiciones y expresiones matemáticas— para que puedan ser procesadas computacionalmente. Por ello, para modeladores familiarizados con DEVS, el esfuerzo de aprendizaje se orienta principalmente a adoptar una escritura más estructurada y validable, antes que a incorporar conceptos propios de un lenguaje de programación de propósito general. Así, el enfoque no busca reemplazar a dichos lenguajes, sino proporcionar una representación intermedia alineada con la definición formal de DEVS y adecuada para su validación y transformación automática.

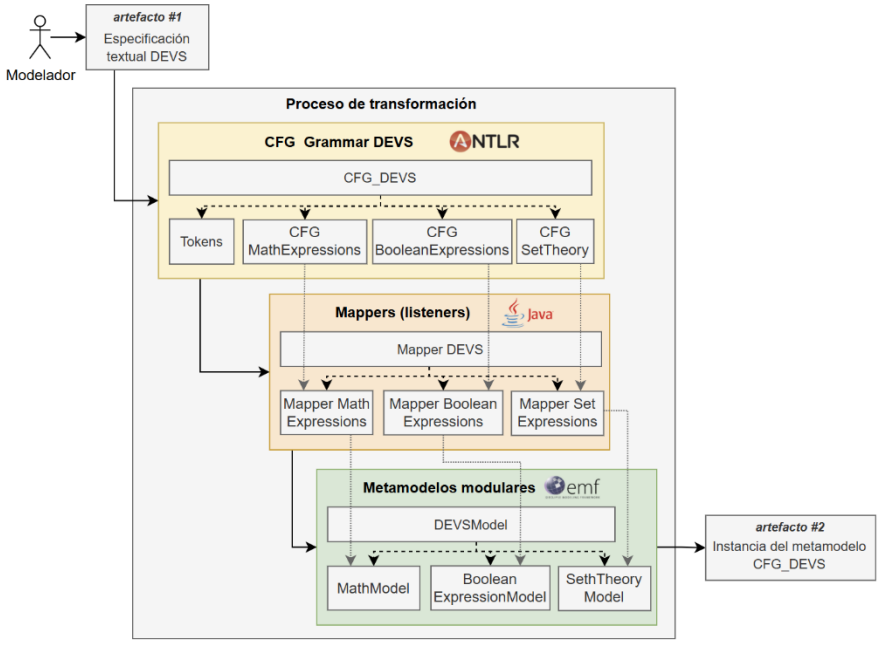

Fig. 1. Solución propuesta para la transformación de especificaciones textuales DEVS en instancias del metamodelo CFG_DEVS.

La gramática fue implementada utilizando ANTLR (Parr, 2026), lo que permite generar automáticamente un analizador léxico y sintáctico capaz de procesar especificaciones textuales y construir árboles de sintaxis. En este trabajo, la gramática original fue reutilizada sin modificaciones en su estructura ni en sus reglas de producción, realizándose únicamente adaptaciones menores consistentes en la incorporación de atributos y etiquetas en un subconjunto limitado de reglas. Estas extensiones tienen como objetivo facilitar el procesamiento durante la fase de mapeo, particularmente en la implementación de los *listeners*.

3.2 Metamodelos Modulares

Con el objetivo de representar la sintaxis concreta del lenguaje en un modelo, se definió una arquitectura basada en múltiples metamodelos interrelacionados (metamodelos modulares). Así, en lugar de definir un único metamodelo monolítico, se optó por separar los distintos aspectos del lenguaje en cuatro metamodelos (tal como se resume en la Tabla 2). Esta decisión permite descomponer el lenguaje en componentes especializados, facilitando la modularidad y mantenibilidad de la solución. Esto también favorece la reutilización de los componentes definidos. Por ejemplo, los metamodelos asociados a teoría de conjuntos, expresiones matemáticas y expresiones booleanas pueden utilizarse de manera independiente o integrarse en futuras extensiones del lenguaje, incluyendo variantes del formalismo DEVS u otros lenguajes de modelado que requieran

representar estructuras similares. Asimismo, la separación entre los metamodelos auxiliares y el metamodelo DevsModel permite modificar o extender partes específicas del lenguaje sin afectar necesariamente la arquitectura completa.

Tabla 2. Metamodelos diseñados.

Metamodelo	Descripción
SetTheoryModel	Representa definiciones de conjuntos y operaciones asociadas.
MathExpressionModel	Modela expresiones matemáticas utilizadas en funciones.
BooleanExpressionModel	Define condiciones y lógica de control.
DevsModel	Representa la estructura principal de los modelos, incluyendo modelos atómicos, acoplados y funciones.

Es importante destacar que la separación propuesta en la Tabla 2 refleja la estructura conceptual del dominio DEVS, donde las definiciones formales combinan teoría de conjuntos, expresiones matemáticas, booleanas y condiciones lógicas. De esta manera, cada metamodelo encapsula un conjunto específico de conceptos, integrándose con los demás por vínculos bien definidos (Figura 2). El metamodelo *DevsModel* actúa como eje de la arquitectura, referenciando elementos definidos en los otros metamodelos para representar funciones, condiciones y estructuras propias de los modelos DEVS.

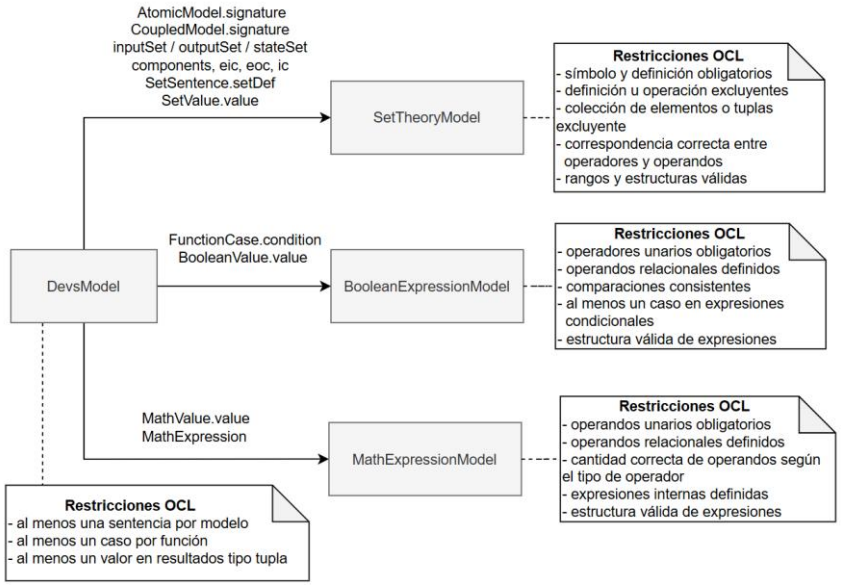

Fig. 2. Esquema de la arquitectura de metamodelos y resumen de restricciones OCL incluidas. El metamodelo *DevsModel* actúa como núcleo y establece vínculos con *SetTheoryModel*, *BooleanExpressionModel* y *MathExpressionModel*, los cuales encapsulan conceptos relacionados a conjuntos, condiciones lógicas y expresiones matemáticas respectivamente.

Asimismo, para garantizar la consistencia de los modelos generados, se incorporaron restricciones Object Constraint Language (OCL). Estas restricciones permiten validar distintas propiedades, tales como la correcta definición de relaciones, la presencia de elementos obligatorios y la coherencia entre los distintos componentes del modelo. En la Figura 2 se presentan de manera resumida las principales restricciones OCL asociadas a cada metamodelo.

Metamodelo *BooleanExpressionModel*. El metamodelo de expresiones booleanas permite representar condiciones lógicas utilizadas en la definición de funciones y reglas dentro de los modelos DEVS. Estas expresiones son fundamentales para modelar comportamientos condicionales, tales como las funciones de transición y salida.

Se organiza a partir de una jerarquía de clases abstractas que estructuran los distintos tipos de expresiones. La clase base *Expression* es extendida por *BooleanExpression*, que a su vez se especializa en *BinaryCondition* (operaciones binarias entre expresiones booleanas, por ejemplo, AND, OR), *UnaryCondition* (operaciones unarias como la negación), y *RelationalCondition* (permite comparar términos mediante operadores relacionales, por ejemplo, igualdad, desigualdad, mayor o menor). Para representar los operandos de estas expresiones, se incluyen clases auxiliares como *BooleanTerm*, *Element*, *Constant* y *ElementDefinition*, permitiendo modelar tanto valores simples como referencias a elementos definidos en otros metamodelos. Asimismo, se incorporan estructuras como *ConditionalExpression*, que permite representar expresiones condicionales compuestas por múltiples casos (como ser, estructuras tipo *if-then-else*), y *ActionExpression*, utilizada para modelar resultados asociados a dichas condiciones.

Metamodelo *MathExpressionModel*. El metamodelo de expresiones matemáticas representa expresiones aritméticas usadas en la definición de funciones dentro de los modelos DEVS, tales como cálculos en funciones de transición, salida y avance de tiempo.

Su estructura se basa en una jerarquía de clases que permite modelar expresiones simples y compuestas. La clase *MathExpression* representa cualquier expresión matemática, y es especializada en cuatro subclases: *i) BinaryExpression* que modela operaciones binarias entre expresiones (por ejemplo, suma, resta, multiplicación, división y potencia); *ii) UnaryExpression* que representa operaciones unarias (por ejemplo, raíz cuadrada u otras funciones matemáticas); *iii) ValueExpression* que permite representar valores constantes o literales numéricos; y *iv) IdentifierExpression* utilizada para representar variables o referencias a elementos definidos en otros contextos.

Las expresiones se construyen de manera recursiva, permitiendo la composición de estructuras complejas mediante la combinación de subexpresiones. Esta característica es clave para reflejar la naturaleza jerárquica de las expresiones matemáticas definidas en la gramática. Además, el metamodelo contempla la definición de operadores matemáticos, asegurando su correcta asociación con los operandos y respetando la precedencia de operaciones definida a nivel sintáctico en la gramática.

Metamodelo *SetTheoryModel*. El metamodelo de teoría de conjuntos permite representar definiciones de conjuntos y operaciones asociadas, las cuales son requeridas para la definición de conjuntos de entrada, salida y estados.

Se organiza a partir de la clase raíz *SetExpression*, que es especializada en distintas formas de definición, a saber: *i) SetExplicitDefinition* y *SetImplicitDefinition* que permiten definir conjuntos de manera explícita (enumerando elementos) o mediante condiciones; *ii) SetDefinition*, que agrupa colecciones de elementos o tuplas; y *iii) SetOperationResult* que modela operaciones entre conjuntos (como unión, intersección, diferencia y producto cartesiano). Adicionalmente, se incluyen estructuras auxiliares como *ElementCollection*, *TupleCollection*, *ElementDefinition* y *Range* que permiten representar distintos tipos de definiciones y construcciones sobre conjuntos.

Metamodelo *DevsModel*. El metamodelo del modelo DEVS constituye la base de la representación de la sintaxis concreta del lenguaje, modelando tanto la estructura como el comportamiento de los modelos definidos bajo este formalismo. Este metamodelo integra los distintos componentes del lenguaje mediante referencias a los metamodelos definidos previamente (Figura 3).

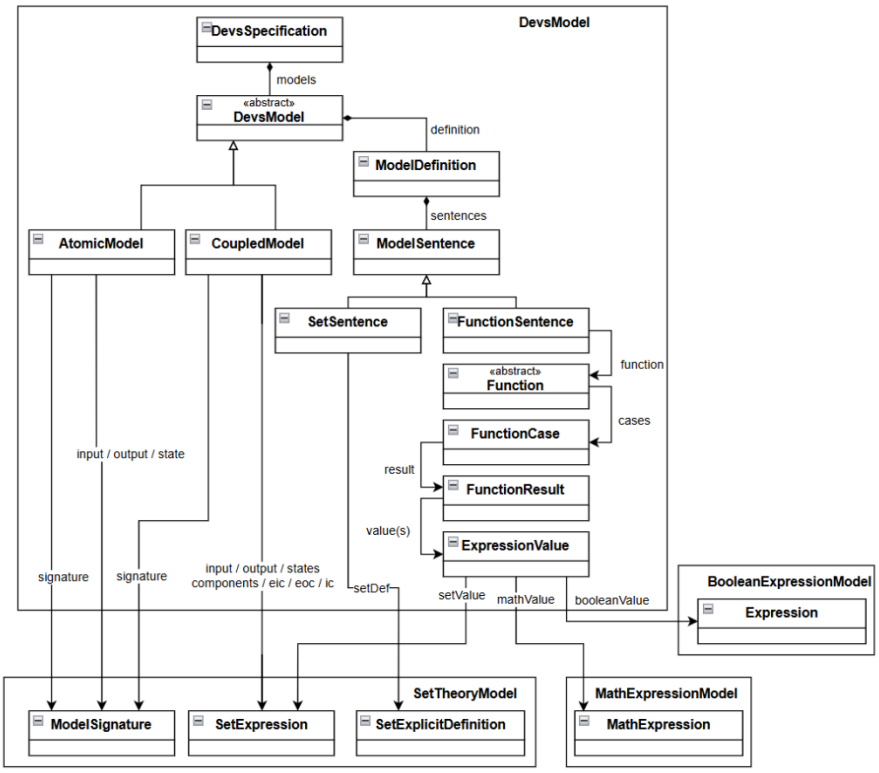

Fig. 3. Vista simplificada de la arquitectura de metamodelos diseñada, donde el metamodelo DEVS actúa como núcleo e integra los metamodelos restantes mediante relaciones específicas.

La clase principal *DevsSpecification* representa la raíz del modelo y contiene un conjunto de definiciones de modelos DEVS. Estos modelos son instancias de la clase abstracta *DevsModel*, la cual se especializa en *AtomicModel* y *CoupledModel.*

AtomicModel representa modelos atómicos e incluye el nombre del modelo, los conjuntos de entrada, salida y estado, y las funciones características del formalismo: transición externa e interna, salida y avance de tiempo. Por su parte, *CoupledModel* representa modelos acoplados, permitiendo describir sistemas compuestos a partir de submodelos. Incluye el conjunto de componentes, las relaciones de acoplamiento (entrada externa, salida externa e internas), junto con la función de selección.

El comportamiento de los modelos se define mediante una jerarquía de clases que incluye *Function*, *FunctionBody* y *FunctionCase*, lo que permite representar funciones como conjuntos de reglas condicionales. Cada caso combina una condición —expresada mediante el metamodelo de expresiones booleanas— con un resultado. Los resultados de las funciones se modelan a través de distintas variantes de *FunctionResult*, tales como *TupleResult*, *SingleValueResult* e *IdentifierResult*, permitiendo representar tanto valores simples como estructuras más complejas, en concordancia con la especificación formal de DEVS.

Como puede observarse, este metamodelo se integra estrechamente con los restantes. En particular, las expresiones booleanas se utilizan para definir condiciones en las funciones, las expresiones matemáticas permiten modelar cálculos dentro de las mismas, y la teoría de conjuntos se emplea en la definición de los conjuntos de entrada, salida, estado y en diversas estructuras internas del modelo. De esta manera, el metamodelo DEVS no solo estructura los modelos, sino que también articula las relaciones entre los distintos componentes de la sintaxis concreta, dando lugar a una representación completa, consistente y alineada con la especificación formal CFG_DEVS.

3.3 Transformación de la CFG al metamodelo

La transformación desde la sintaxis concreta hacia su representación basada en modelos se realiza mediante un mecanismo de mapeo basado en listeners generados por ANTLR. Estos listeners se activan durante el recorrido del árbol de sintaxis (producido por el parser de CFG_DEVS) y permiten asociar cada construcción sintáctica reconocida como nodo del árbol con su correspondiente elemento en los metamodelos.

El proceso es incremental y dirigido por eventos (*enter/exit* de reglas), lo que permite construir progresivamente instancias de los metamodelos a medida que se recorre el árbol de derivación. Para gestionar el contexto y las dependencias entre elementos, se emplean estructuras auxiliares -principalmente pilas- que permiten almacenar resultados parciales y resolver correctamente la composición de construcciones complejas, como expresiones anidadas o definiciones de funciones.

La modularidad de la gramática se refleja en la implementación de *listeners* especializados, cada uno responsable de mapear un subconjunto del lenguaje (expresiones matemáticas, booleanas o de teoría de conjuntos). Estos componentes se integran progresivamente para construir una instancia completa del metamodelo CFG_DEVS.

Mapeo de reglas de la gramática a elementos del metamodelo. Sobre el mecanismo de transformación descrito, cada regla relevante de la gramática se asocia a uno o más elementos de la sintaxis concreta. Este mapeo se implementa en los métodos del *listener* mediante instanciación, utilizando las funciones generadas por EMF. Por ejemplo:

- Las reglas de expresiones matemáticas se mapean a instancias del metamodelo *MathExpressionModel*, tales como *BinaryExpression* y *UnaryExpression*.
- Las expresiones booleanas se transforman en instancias del metamodelo *BooleanExpressionModel*, incluyendo *BinaryCondition*, *UnaryCondition* y *RelationalCondition*.
- Las definiciones de conjuntos se representan mediante instancias del metamodelo *SetTheoryModel*, tales como *SetExplicitDefinition* y *SetOperationResult*.
- Las construcciones principales del lenguaje DEVS se mapean a instancias del metamodelo *DevsModel*, incluyendo *AtomicModel*, *CoupledModel* y sus funciones asociadas.

Durante la transformación, el *listener* no solo construye instancias de un único metamodelo, sino que integra elementos provenientes de los distintos metamodelos definidos. Esta integración se logra mediante la creación de referencias entre instancias de distintos metamodelos, respetando las relaciones definidas en la sintaxis concreta.

Una vez completado el recorrido del árbol de sintaxis, el resultado es una instancia completa del metamodelo DEVS, que incluye referencias a elementos de los metamodelos auxiliares. Posteriormente se aplican las restricciones definidas mediante OCL para verificar la consistencia de la instanciación resultante.

4 Caso de Estudio

Con el fin de ilustrar la estrategia de transformación propuesta, se presenta un ejemplo basado en la especificación textual de un modelo atómico DEVS denominado *Fixture* (Goldstein et al. 2014). Este caso permite evidenciar la correspondencia entre la sintaxis concreta del lenguaje, el procesamiento realizado por el mecanismo de mapeo y la estructura del modelo resultante conforme al metamodelo.

La Figura 4 presenta un fragmento representativo de la especificación textual. En este ejemplo, la definición del modelo incluye su nombre y los conjuntos de entrada, salida y estado, así como la especificación de las funciones características del formalismo DEVS. A partir de esta entrada, el *listener* asociado a la regla *AtomicModel* crea una instancia de *AtomicModel*, asigna el nombre y transforma los conjuntos definidos en la especificación textual mediante el metamodelo de teoría de conjuntos. Asimismo, se instancia un *ModelDefinition*, que actúa como contenedor de las sentencias que describen el comportamiento del modelo.

```
1   Fixture = (X, Y, S, \delExt, \delInt, \lambda, \ta)
2
3   \where
4
5   X = {(signalin,signal) | signal \in SIGNALS};
6   Y = {(levelout,level) | level \in LEVELS};
7   SIGNAL ={"Still", "Moving"};
8   LEVELS = {"Dark", "Light"};
9   S = LEVELS \product \R;
10  \delExt((level, deltatint),e,(signalin, signal)) =  (level, 0) \if signal \equalTo "Moving" \and level \equalTo "Dark",
11                                                      (level, \infinity) \inOtherCase;
```

Fig. 4. Fragmento de la especificación textual *Fixture* definida haciendo uso de CFG_DEVS.

La Figura 5 muestra un fragmento de la implementación del método *enterAtomicModel*, el cual ilustra cómo las reglas de la gramática se vinculan directamente con la creación de instancias del metamodelo. Este mecanismo evidencia el carácter sistemático del proceso de transformación, en el que cada construcción sintáctica reconocida es mapeada a su correspondiente representación en la sintaxis concreta.

```
@Override
public void enterAtomicModel(CFG_DEVS_MODELParser.AtomicModelContext ctx) {
    AtomicModel model = factory.createAtomicModel();
    currentAtomic = model;

    ModelSignature sig = buildSignature(ctx.sig);
    model.setSignature(sig);
    model.setName(ctx.sig.modelSymbol().getText());

    model.setInputSet(SetTheoryMapperHelper.map(ctx.modelTuple.input));
    model.setOutputSet(SetTheoryMapperHelper.map(ctx.modelTuple.output));
    model.setStateSet(SetTheoryMapperHelper.map(ctx.modelTuple.state));

    ModelDefinition def = factory.createModelDefinition();
    model.setDefinition(def);

    stack.push(model);
}
```

Fig. 5. Implementación de la función *enterAtomicModel* presente en el mapeador DEVS.

La Figura 6 muestra el fragmento del modelo XMI generado, correspondiente a uno de los casos de la función de transición externa (δext). Puede observarse cómo la regla textual se representa mediante una instancia de *ExternalTransitionFunction*, cuyo cuerpo (*FunctionBody*) contiene una colección de casos (*FunctionCase*). Cada caso está compuesto por una condición y un resultado. La condición se modela como una expresión booleana estructurada, en este caso una *BinaryCondition* (líneas 147-156) que combina dos condiciones relacionales (*signal = Moving y level = Dark*). Por su parte, el resultado se representa mediante un *TupleResult* (líneas 157-172), cuyos valores corresponden a los elementos definidos en la especificación textual. En este punto, se evidencia la integración de múltiples metamodelos: los valores del resultado incluyen tanto expresiones de teoría de conjuntos como expresiones matemáticas, demostrando la capacidad del enfoque para combinar distintos dominios en una única representación coherente.

Este resultado no solo demuestra la transformación sintáctica, sino también la capacidad del enfoque para integrar de manera coherente múltiples metamodelos especializados dentro de una única representación estructurada. Luego, la estrategia propuesta establece una correspondencia directa entre las reglas de la gramática y los elementos de la sintaxis concreta, permitiendo una transformación sistemática y extensible. Este caso también permite evidenciar la aplicabilidad inicial de la propuesta como etapa dentro de un proceso de desarrollo de lenguaje DEVS. En particular, muestra que una especificación textual puede ser procesada automáticamente y convertida en una representación estructurada, validable y manipulable mediante herramientas EMF. Además, al persistirse en formato XMI, el modelo generado puede ser almacenado, inspeccionado y reutilizado como entrada para nuevas transformaciones o procesos posteriores de generación de artefactos ejecutables. De esta manera, el caso de estudio no solo

ilustra la transformación desde la sintaxis textual hacia instancias de metamodelos, sino también el potencial de reutilización de los modelos obtenidos dentro de una cadena de desarrollo dirigida por modelos.

```
144  <deltaExt xsi:type="devsModel:ExternalTransitionFunction">
145    <body xsi:type="devsModel:FunctionBody">
146      <cases xsi:type="devsModel:FunctionCase">
147        <condition xsi:type="booleanExpressionModel:BinaryCondition">
148          <left xsi:type="booleanExpressionModel:RelationalCondition">
149            <left xsi:type="booleanExpressionModel:Element" name="signal"/>
150            <right xsi:type="booleanExpressionModel:Constant" value="Moving"/>
151          </left>
152          <right xsi:type="booleanExpressionModel:RelationalCondition">
153            <left xsi:type="booleanExpressionModel:Element" name="level"/>
154            <right xsi:type="booleanExpressionModel:Constant" value="Dark"/>
155          </right>
156        </condition>
157        <result xsi:type="devsModel:TupleResult">
158          <values xsi:type="devsModel:SetValue">
159            <value xsi:type="setTheoryModel:SetImplicitDefinition">
160              <operation xsi:type="setTheoryModel:SetOperationResult">
161                <terms xsi:type="setTheoryModel:SetOperationTerm">
162                  <operationSet xsi:type="setTheoryModel:OperationSet">
163                    <symbol xsi:type="setTheoryModel:SetSymbol" name="level"/>
164                  </operationSet>
165                </terms>
166              </operation>
167            </value>
168          </values>
169          <values xsi:type="devsModel:MathValue">
170            <value xsi:type="mathExpressionModel:Element" name="0" value="0.0"/>
171          </values>
172        </result>
173      </cases>
174      <cases xsi:type="devsModel:FunctionCase">
196    </body>
197  </deltaExt>
```

Fig. 6. Fragmento del modelo instanciado por la transformación propuesta, el cual corresponde a un caso de la función de transición externa (δext).

5 Conclusiones

En este trabajo se presentó un enfoque basado en ingeniería dirigida por modelos para transformar especificaciones textuales de DEVS, definidas mediante una gramática libre de contexto, en modelos estructurados conforme un conjunto de metamodelos interrelacionados. A diferencia de enfoques tradicionales que separan la especificación formal de su implementación, la propuesta establece un vínculo explícito entre la sintaxis concreta del lenguaje y su representación modular, mediante un proceso de transformación sistemático basado en el recorrido de árboles de derivación y la instanciación de metamodelos en EMF.

Un aporte central del trabajo es la definición de una arquitectura modular que descompone el lenguaje DEVS en múltiples metamodelos especializados —teoría de conjuntos, expresiones matemáticas, expresiones booleanas y estructura DEVS— los

cuales se integran para representar de manera unificada la sintaxis concreta. Esta organización permite reflejar con mayor fidelidad la estructura formal del lenguaje y, al mismo tiempo, favorece la extensibilidad de la propuesta, ya que los metamodelos definidos pueden reutilizarse o adaptarse en futuras extensiones y variantes del formalismo DEVS. Asimismo, la incorporación de restricciones mediante OCL permite validar propiedades de los modelos generados, garantizando su consistencia y reforzando la confiabilidad del proceso de transformación. Por razones de espacio, no se presenta un detalle exhaustivo de dichas restricciones, aunque se incluyen de forma resumida en la Figura 2.

El caso de estudio desarrollado demuestra la viabilidad del enfoque como estrategia de creación de un modelo estructurado DEVS a partir de especificaciones textuales. No obstante, el alcance actual del trabajo se concentra en el proceso de definición del lenguaje y en la transformación hacia instancias de metamodelos EMF, por lo que la ejecución directa de simulaciones a partir de los modelos generados no forma parte de esta etapa. Como trabajo futuro, se plantea el mapeo de la representación resultante hacia el metamodelo DEVS de la sintaxis abstracta (Blas & Gonnet, 2023a), así como la posterior generación de artefactos ejecutables o la integración con motores de simulación DEVS existentes. Esto permitirá completar el ciclo desde la especificación textual del modelo hasta su simulación.

Referencias

Blas, M. J., & Gonnet, S. (2023a). Modeling and simulation through the metamodeling perspective: The case of the discrete event system specification. *Handbook of Model-Based Systems Engineering*.

Blas, M.J. & Gonnet, S. (2023b). Metamodel-based formalization of DEVS atomic models. *Simulation 99(5)*.

Blas, M. J., Gonnet, S., Kim, D., & Zeigler, B. P. (2023c). *A context-free grammar for generating full classic DEVS models*. Proceedings of the Winter Simulation Conference.

Blas, M.J., Gonnet, S., & Zeigler, B. (2021). *Towards a Universal Representation of DEVS: A Metamodel-Based Definition of DEVS Formal Specification*. Proceedings of the 2021 Annual Modeling and Simulation Conference.

Cristiá, M., D. A. Hollmann, & C. Frydman. (2019). A Multi-target Compiler for CML-DEVS. *Simulation 95(1)*:11-29.

Goldstein, R., Wainer, G. A., & Khan, A. (2014). The DEVS formalism. *Formal Languages for Computer Simulation: Transdisciplinary Models and Applications* (pp. 62-102). IGI Global.

Hopcroft J. E., Motwani R. & Ullman J.D. (2007). *Introduction to Automata Theory, Languages and Computation*. Madrid: Pearson.

Krahn, H., B. Rumpe, & Völkel, S. (2007). *Integrated Definition of Abstract and Concrete Syntax for Textual Languages*. Proceedings of the 2007 Model Driven Engineering Languages and Systems Conference (pp. 286-300). Berlin: Springer Berlin Heidelberg.

Paige, R. F., Ostroff, J. S., & Brooke, P. J. (2000). Principles for Modeling Language Design. *Information and Software Technology 42(10)*:665-675.

Parr, T. (2026). ANTLR. Disponible en: https://www.antlr.org/

Pons, C., Giandini, R. S., & Pérez, G. (2010). *Desarrollo de software dirigido por modelos*. EDULP / McGraw-Hill.

Sarjoughian, H. S., A. Alshareef, and Y. Lei. 2015. *Behavioral DEVS Metamodelin*g. In Proceedings of the 2015 Winter Simulation Conference, edited by L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, 2788 2799. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.
Zeigler, B. P., Muzy, A., & Kofman, E. (2018). *Theory of modeling and simulation: Discrete event and iterative system computational foundations*. Academic Press.