Universidad Tecnológica Nacional
AJEA – Actas de Jornadas y Eventos Académicos de UTN
Trabajo Completo Publicado en AJEA – Full Text Published in AJEA
https://rtyc.utn.edu.ar/index.php/ajea/article/view/1c33
Libro de actas de resúmenes DOI: https://doi.org/10.33414/ajea.1640.2024
Colección JIT en AJEA https://rtyc.utn.edu.ar/index.php/ajea/JITs

Jóvenes Investigadores Tecnológicos
Facultad Regional Rafaela 26 y 27 de octubre de 2023

Área: Sistemas de Información e Informática / Categoría: Estudiante de grado / Regional: Facultad Regional Santa Fe

Generación de Modelos de Componentes Basados en Modelos DEVS Acoplados.

Component Model Generation Based on Coupled DEVS Models.

Presentación: 17/10/2023

Santiago Andrés Mercanti
Universidad Tecnológica Nacional - FRSF
santiagoamercanti@gmail.com

Resumen

Este artículo presenta una extensión de una investigación previa sobre sintaxis de modelos acoplados DEVS. Se describe el diseño de una gramática libre de contexto para definir con precisión la sintaxis de dichos modelos. Se explora la relación entre modelos acoplados DEVS y modelos de componentes UML, que representan interacciones en sistemas. Se propone un método para transformar modelos DEVS en modelos de componentes utilizando la herramienta "PlantText". Los pasos detallados incluyen la preparación del texto del modelo, la definición de nombres, componentes e interfaces, y la creación de vínculos. Este enfoque amplía la comprensión de los modelos de componentes y su utilidad en contextos diversos.

Palabras clave: modelo, componente, UML, interfaz.

Abstract

This article presents an extension of a previous research on syntax for coupled DEVS models. It describes the design of a context free grammar to precisely define the syntax of these models. The relationship between coupled DEVS models and UML component models, which depict interactions in systems, is explored. A method is proposed to transform DEVS models into component models using the "PlantText" tool. The detailed steps include preparing the model's text, defining names, components, and interfaces, and creating connections. This approach enhances the understanding of component models and their adaptability in various contexts.

Keywords: model, component, UML, interface

Introducción

En el presente artículo se desarrolla una continuación a la investigación llevada a cabo durante el pasado año 2022 presentada como “Sintaxis para modelos acoplados DEVS” en JIT 2022 Reconquista bajo mi autoría. Dicho artículo exhibe el diseño de una gramática libre de contexto que permita definir una sintaxis precisa para la especificación de modelos acoplados DEVS.

A modo de introducción es importante recordar que Discrete EVent System specification, o DEVS, (Zeigler, Muzy y Kofman, 2018) es el formalismo más general para la representación de sistemas de eventos discretos, orientado a problemas de modelización y simulación. Se dice es el formalismo para eventos discretos universal, ya que absorbe a otros más populares como las Redes de Petri, las Statecharts, Grafcet y Grafos de Eventos. Los modelos DEVS acoplados son modelos formados básicamente por tres elementos: puertos (de entrada, salida e internos), acoplamientos y otros modelos más internos.

AJEA – Actas de Jornadas y Eventos Académicos de UTN
Libro de actas de resúmenes DOI: https://doi.org/10.33414/ajea.1640.2024
Texto Completo Publicado en AJEA – Full Text Published in AJEA
https://rtyc.utn.edu.ar/index.php/ajea/article/view/1633

Una vez ya comprendida la naturaleza y la definición de los modelos acoplados DEVS, en este trabajo se aborda la vinculación de estos modelos con los modelos de componentes de UML. Los modelos de componentes son representaciones gráficas basadas en la especificación de UML que ilustran las relaciones entre los elementos individuales de un sistema. Estos elementos interactúan a través de interfaces, las cuales actúan como vínculos entre diferentes componentes. En esencia, un modelo de componentes se compone de dos elementos fundamentales:

- Componente: Estos son bloques que contienen unidades lógicas del sistema, presentando una abstracción ligeramente más elevada que las clases convencionales.

- Interfaz: Siempre asociada a un componente, se utiliza para representar la zona del módulo destinada a la comunicación con otros componentes.

A través de esta exploración, el artículo busca ampliar nuestra comprensión de los modelos de componentes y su importancia en la representación precisa de las interacciones en sistemas complejos. Los modelos de componentes ayudan a esclarecer de manera práctica las relaciones entre los distintos elementos de un sistema, por lo tanto, su uso mejora el entendimiento del modelo en el cual se aplique. Además, en este trabajo se aborda la interesante tarea de transformar un modelo DEVS en un modelo de componentes, aportando una perspectiva valiosa sobre la adaptabilidad y utilidad de estos enfoques en diferentes contextos.

Metodología

Para facilitar la elaboración de los diagramas de componentes, optaremos por la utilización de la herramienta "PlantText". Dicha plataforma corre con la ventaja de permitir la construcción de diagramas de modelos a partir de código, lo cual brinda una serie de ventajas sustanciales a la hora de convertir la definición de un modelo a otro.

Dentro de esta herramienta nos encontramos con distintos estilos de samples, de los cuales nos quedaremos con el nombrado “Component 04 – Labels And Notes”, idóneo para la elaboración de diagramas de componentes.

Un script de un modelo de componentes tiene la siguiente forma:

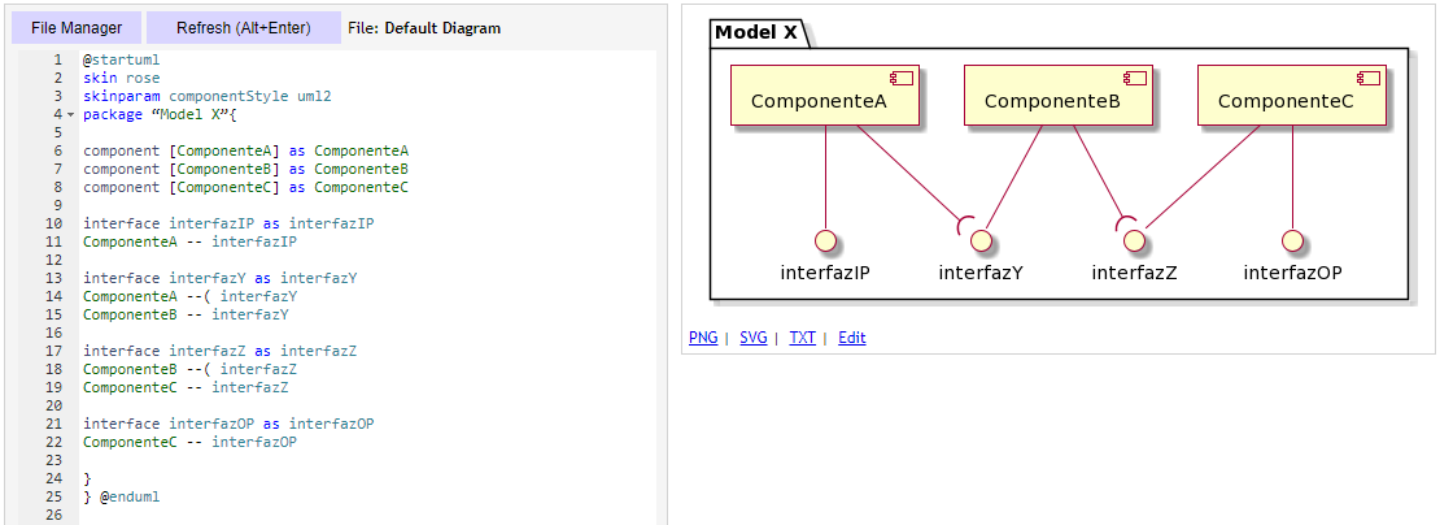

Figura 1 – Modelo de componentes y su representación en PlantText

El término "package" establece el nombre del modelo (en este caso, denominado "Modelo X"), el cual contiene tres componentes fundamentales (ComponenteA, ComponenteB y ComponenteC), acompañados por cuatro interfaces específicas (interfazIP, interfazY, interfazZ y interfazOP). Estas conexiones entre los componentes e interfaces se establecen mediante " -- " cuando un componente provee la interfaz a otros, y a través de " --(" cuando un componente busca establecer una conexión con la interfaz suministrada por otro.

AJEA – Actas de Jornadas y Eventos Académicos de UTN
Libro de actas de resúmenes DOI: https://doi.org/10.33414/ajea.1640.2024
Texto Completo Publicado en AJEA – Full Text Published in AJEA
https://rtyc.utn.edu.ar/index.php/ajea/article/view/1633

El objetivo de esta investigación radica en determinar el procedimiento óptimo para obtener un modelo de componentes, partiendo de la definición formal de del diagrama DEVS de un modelo.

Definimos entonces los pasos que deberá realizar el algoritmo:

1. Solicitar el ingreso de la formalización del modelo DEVS:

- Solicitar al usuario que ingrese el texto con la definición del modelo DEVS.

2. Preparar el texto para el modelo de componentes:

- Crear un archivo de texto y escribir las siguientes líneas en él:

```
"@startuml
skin rose
skinparam componentStyle uml2"
```

3. Definir el nombre del modelo:

- Buscar la línea que contiene "defineCouple" en el texto ingresado.
- Extraer el "ElementName" antes del "=" para obtener el nombre del modelo.
- Agregar la siguiente línea al archivo de texto creado en el paso 2:

```
"package [ElementName] {"
```

4. Establecer los componentes del modelo:

- Seleccionar el tercer elemento de la definición del modelo (componentes).
- Buscar el "SetDefinitions" con el mismo nombre.
- Por cada elemento dentro de "SetDefinitions", agregar la siguiente línea al archivo de texto:

```
"component [NombreComponente] as NombreComponente"
```

5. Establecer las interfaces internas del modelo:

- Detectar el séptimo elemento (IC) en la definición del modelo.
- Buscar el "couplingDefinition" correspondiente.
- Por cada coupling, tomar el segundo elemento de los primeros y segundos "couplingPair" y crear la interfaz correspondiente:

```
"interface IC_primerCP_segundoCP as IC_primerCP_segundoCP"
```

- Tomar el primer elemento del primer "couplingPair" y definir la relación:

```
"nombreComponente1 --( IC_primerCP_segundoCP"
```

- Tomar el primer elemento del segundo "couplingPair" y definir la relación:

AJEA – Actas de Jornadas y Eventos Académicos de UTN
Libro de actas de resúmenes DOI: https://doi.org/10.33414/ajea.1640.2024
Texto Completo Publicado en AJEA – Full Text Published in AJEA
https://rtyc.utn.edu.ar/index.php/ajea/article/view/1633

"*nombreComponente2* -- *IC_primerCP_segundoCP*"

6. Establecer las interfaces de entrada al modelo:

- Detectar el quinto elemento (EIC) en la definición del modelo.

- Buscar el "couplingDefinitions" con el mismo nombre.

- Por cada coupling, tomar el segundo elemento de los primeros y segundos "couplingPair" y crear la interfaz correspondiente:

"interface *EIC_primerCP_segundoCP* as *EIC_primerCP_segundoCP*"

- Tomar el primer elemento del segundo "couplingPair" y definir la relación:

"[*nombreComponente*] -- *EIC_primerCP_segundoCP*"

7. Cerrar el modelo:

- Agregar la siguiente línea al archivo de texto creado en el paso 2:

"}} @enduml"

8. Guardar y cerrar el archivo de texto.

9. Mostrar un mensaje al usuario indicando que el proceso se ha completado con éxito.

Resultados y discusión

Realizado todos los pasos anteriores, una plantilla del modelo de componentes queda establecida como:

```
@startuml
skin rose
skinparam componentStyle uml2
package “ElementName”{

component [NombreComponente_1] as NombreComponente_1
...
component [NombreComponente_N] as NombreComponente_N

interface IC_primerCP_segundoCP_1 as IC_primerCP_segundoCP_1
nombreComponente1_1 --( IC_primerCP_segundoCP_1
nombreComponente2_1 -- IC_primerCP_segundoCP_1
...
interface IC_primerCP_segundoCP_N as IC_primerCP_segundoCP_N
nombreComponente1_N --( IC_primerCP_segundoCP_N

nombreComponente2_N -- IC_primerCP_segundoCP_N

interface EIC_primerCP_segundoCP_1 as EIC_primerCP_segundoCP_1
nombreComponente_1 -- EIC_primerCP_segundoCP_1
...
```

AJEA – Actas de Jornadas y Eventos Académicos de UTN
Libro de actas de resúmenes DOI: https://doi.org/10.33414/ajea.1640.2024
Texto Completo Publicado en AJEA – Full Text Published in AJEA
https://rtyc.utn.edu.ar/index.php/ajea/article/view/1633

```
interface EIC_primerCP_segundoCP_N as EIC_primerCP_segundoCP_N
nombreComponente_N -- EIC_primerCP_segundoCP_N

} } @enduml
```

De esta manera, dada una definición formal DEVS obtenemos el código final a cargar en la herramienta "PlantText" que nos brinda el diagrama de componentes del modelo cargado.

Es de destacar que las salidas del sistema del modelo acoplado DEVS señaladas en la formalización como EOC quedan representadas en el modelo de componentes como las interfaces a las cuales se pueden conectar de los componentes del modelo.

A manera de ejemplo se presenta el siguiente modelo denominado "Environment" y su respectivo diagrama acoplado DEVS, obtenido de (Goldstein, Wainer y Khan, 2014). Dicho modelo posee tres componentes (Detector, Fixture y Worker), los cuales interactúan entre si mediante sus puertos de entradas y salidas (como phaseIn, actionOut, actionIn, signalOut), que vendrían a representar las interfaces de cada componente.

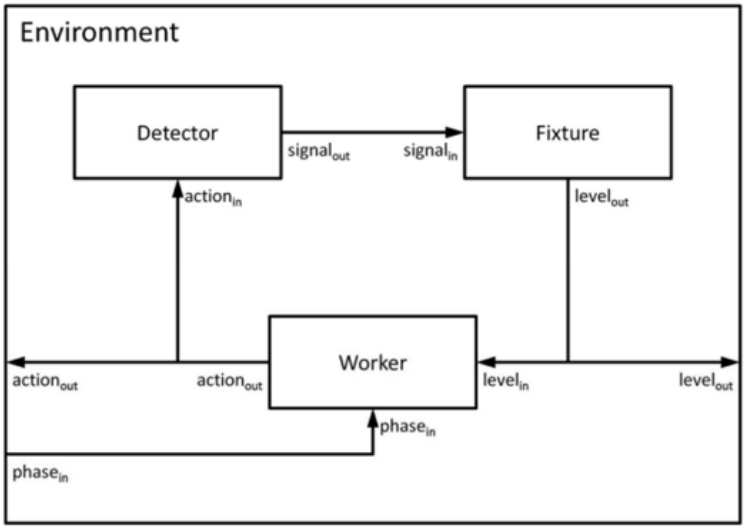

Figura 1 – Modelo acoplado DEVS "Environment" (tomado de Goldstein, Welner y Khan, 2014).

La formalización del modelo DEVS Environment en términos de la gramática propuesta en *"Sintaxis para modelos acoplados DEVS"* (Santiago Mercanti, 2022) se establece como:

```
include ./modelo_Envi.devs as specification of MOD_Ambiente
import modelo-123 from ./modelo-B.devs

ModelEnviroment_1 = (x,y,componentes,definicionDeComponentes,eic,eoc,ic)

InPort = { phaseIn }
OutPort = { actionOut, levelOut }
x = { (phaseIn, phase) / phase belongs to Phases }
y = y_action union y_level where y_action = { (actionOut, v_actionOut) / v_actionOut belongs to Actions } , y_level = { (levelOut, v_levelOut) / v_levelOut belongs to Levels }
componentes = { Detector, Worker, Fixture }
definicionDeComponentes = { Detector is ModelDetector, Worker IS ModelWorker, Fixture is ModelFixture }
eic = { ( (ModelEnviroment; phaseIn), (Worker; phaseIn) ) }
eoc = { ( (Fixture; levelOut), ( ModelEnviroment; levelOut) ), ( (Worker; actionOut), (ModelEnviroment; actionOut) ) }
ic = { ( (Worker; actionOut), (Detector; actionIn) ), ( (Detector; signalOut), (Fixture; signalIn) ), ( (Fixture; levelOut), (Worker; levelIn) ) }
```

Figura 3 – Definición del modelo DEVS "Environment"

Una vez aplicado el algoritmo sobre dicha formalización, el script a obtener y su implementación en PlantText para visualizar el modelo de componentes queda:

AJEA – Actas de Jornadas y Eventos Académicos de UTN
Libro de actas de resúmenes DOI: https://doi.org/10.33414/ajea.1640.2024
Texto Completo Publicado en AJEA – Full Text Published in AJEA
https://rtyc.utn.edu.ar/index.php/ajea/article/view/1633

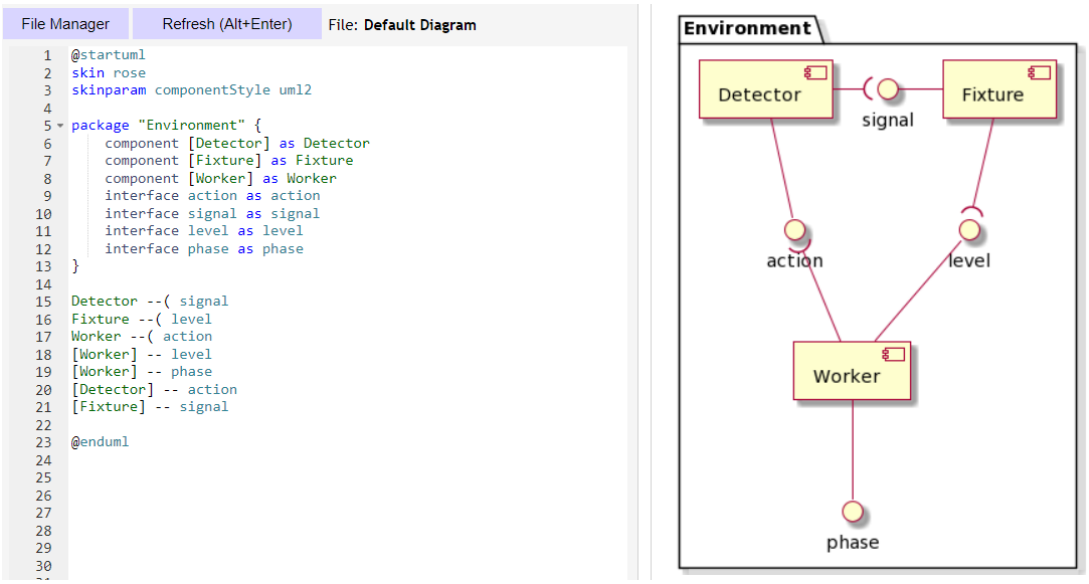

Figura 4 – Modelo de componentes del modelo "Environment"

En donde podemos ver como se crearon correctamente los componentes, sus interfaces y las respectivas conexiones que conforman al modelo de componentes.

Conclusiones

A lo largo del artículo, definimos el concepto de modelo de componentes y establecimos los fundamentos para la creación de estos modelos a partir de la formulación de un modelo acoplado DEVS. Hemos ilustrado como ejemplo su funcionamiento mediante la traducción del modelo "Environment" y su implementación en la herramienta PlantText.

Destacamos especialmente en este informe la elaboración del pseudo-código que describe los pasos necesarios para la transformación de DEVS a Componentes, ya que este será fundamental en la fase siguiente: la implementación real del algoritmo en lenguaje JAVA como parte integral del proyecto.

Además, prevemos que será esencial revisar y corregir este algoritmo una vez que haya sido sometido a pruebas con diversos modelos de uso cotidiano. Nuestra intención es continuar perfeccionando este trabajo con la aspiración de que pueda ser utilizado en un futuro próximo en un contexto más amplio.

Referencias bibliografías

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

Mercanti, S. (2022). "Sintaxis para modelos acoplados DEVS". JIT 2022 Reconquista.

Zeigler, B., Muzy, A. y Kofman, E. (2018). "Theory of Modeling and Simulation: Discrete Event & Iterative System Computational Foundations – Third Edition". Londres: Academic Press.

PlantUML, Graphviz, Ace Editor, Johan Sundström, and Steven Nichols (2013). PlantText. Disponible en *planttext.com*