Asset Administration Shell para representación de procedimientos de procesos batch

Johnny Alvarado Domínguez[1[0000-0002-2099-2321]], Silvio Gonnet[1[0000-0003-3024-4754]] y Marcela Vegetti[1[0000-0003-4016-1717]]

[1] INGAR (UTN-CONICET), 3000 Santa Fe, Argentina
(jaalvarado, sgonnet, mvegetti)@santafe-conicet.gov.ar

Abstract. Un Asset es un activo considerado de importancia para una empresa, puede ser tangible como máquinas, productos, sensores planos, o intangibles como un software. Un Asset Administration Shell (AAS) provee una representación digital de un asset por medio de submodelos que pueden o no ser estandarizados. Los submodelos estandarizados permiten la interoperabilidad con otros asset. Un digital Twin puede ser implementado mediante un AAS. Un digital Twin que represente con gran precisión un proceso de producción (tareas de procesamiento, recursos necesarios) puede mejorar la tarea de planeación y programación de la producción. El estándar ISA 88 provee un conjunto de estándares y terminología para el control de procesos batch definiendo el modelo físico, procedimientos y recetas. El objetivo del presente trabajo es el desarrollo de un submodelo que capture la información relativa al proceso de producción batch basado en el estándar ISA 88. Mas específicamente, se presenta un submodelo para la representación del procedimiento de producción basado en el lenguaje Sequential Function Chart.

Keywords: Asset Administration Shell, Digital Twin, Submodel, ISA 88, Sequential Function Chart

1 Introducción

Las empresas enfrentan un nuevo desafío relacionado a la personalización de la producción. Los clientes han aumentado su participación en la toma de decisiones con respecto al diseño de los productos que consumen, dando origen a la llamada personalización de la producción o producción de lote de tamaño 1. Esto se puede reflejar en la industria automovilística, ropa y calzado, pero también en empresas farmacéuticas, donde se está empezando a emplear el concepto de personalización de medicamentos para pequeños grupos de personas (pacientes con llamadas enfermedades raras) o en la industria de productos químicos, donde las regulaciones de distintos países obliga a las empresas a fabricar el mismo producto con diferentes especificaciones en cuanto al contenido de sus ingredientes [1], impactando el número de fórmulas a producir y el tamaño de cada una de ellas. Poder suplir la demanda de lotes pequeños y al mismo tiempo ser rentables, permitiría a las pequeñas y medianas empresas poder acceder a nuevos mercados, fidelizar dichos clientes y disminuir su inventario, sin embargo, esto requiere de una gran flexibilidad en la planeación y el control de la producción. La llamada Industria 4.0 es una iniciativa alemana que integra máquinas, sensores,

módulos de producción y productos para que compartan información en los llamados sistemas ciber físicos de producción (CPPS). En un CPPS, un producto puede consultar los recursos requeridos y orquesta su propia producción, mediante la autoconfiguración de los procesos en un sistema de producción modular. Lo anterior permite que lotes pequeños puedan ser fabricados bajo las condiciones de producción en masa [2].

Para lograr la interoperabilidad de los componentes de un CPPS, la organización Plattform Industrie 4.0 ha propuesto el concepto de Asset Administration Shell (AAS) [3]. Un Asset Administration Shell es la representación digital de un asset o activo de una empresa, como, por ejemplo, máquinas, sensores, productos, planos, software, etc. Un Asset Administration Shell organiza la información de un activo mediante submodelos, los cuales pueden ser estandarizados. Los submodelos basados en estándares garantizan la interoperabilidad entre dos o más AAS que representan diversos assets. La implementación de un AAS permite la creación de un Digital Twin (DT). En [4], los autores proponen una categorización de un DT dependiendo del nivel de integración: un DT puede ser solo una representación digital del modelo (DM), en el cual no hay un intercambio de información entre el objeto físico y el objeto virtual, un digital shadow (DS), en el cual el objeto físico envía información automáticamente al objeto virtual o un DT, en el cual se realiza un intercambio de información automatizada entre el objeto físico y el objeto virtual. La implementación de un DT en la industria de procesos puede ayudar en las actividades de planeación y programación de la producción [5], [6] y mejorar la eficiencia de los sistemas de producción [7].

La creación de un DT para la industria de procesos batch a partir de un AAS, requiere la creación de un submodelo que capture la información de dicho proceso. El estándar ISA 88 [8] provee un conjunto de modelos para el diseño y especificación de procesos batch. Un concepto importante en el estándar ISA 88 es la receta, la cual provee la información acerca de qué se va a producir, en qué cantidad, cuando y cuales recursos se requieren para su producción. La creación de un submodelo de AAS basado en el estándar ISA 88 permitiría la digitalización de la información de la receta y la interoperabilidad entre los recursos que intervienen en un proceso de producción batch. La receta contiene cinco elementos principales: a) el Header, donde se encuentra la información administrativa de la receta (versión, información de aprobación, estado, entre otras), b) la Formula, que incluye los parámetros de entrada, salida y de procesos (materia prima, otros recursos adicionales, temperatura, presión, rendimiento esperado del batch), c) los Requerimientos de Equipo, que contiene la información de los posibles equipos a ser usados en la producción, d) el Procedimiento, el cual define la estrategia para llevar a cabo el proceso y e) Otra Información, como por ejemplo la relacionada a cumplimientos normativos, embalaje y demás requerimientos del cliente o legales.

La receta de procedimiento puede ser expresada en 3 niveles: unidad de procesamiento, operación y fase. Al nivel de fase se encuentran los equipos de control como lo son los controladores lógicos programables (PLC). El objetivo del presente trabajo es proveer un submodelo de un AAS que capture la información de la receta de procedimiento a nivel de fase. Para ello, se utilizarán los conceptos del lenguaje gráfico Sequential Function Chart (SFC), el cual es definido en el estándar IEC 61131-3 [9] como lenguaje de programación para PLC.

Este artículo está organizado de la siguiente manera: La sección 2 provee información relevante acerca de los conceptos de AAS, SFC y trabajos relacionados. En la sección 3 se detalla el modelo propuesto. Mientras que en la sección 4 se presenta una discusión del trabajo realizado. Finalmente, en la sección 5, se presentan las conclusiones y desafíos pendientes por abordar en la especificación de un submodelo de AAS basado en la ISA 88 para la industria de procesos batch.

2 Antecedentes y trabajos relacionados

En esta sección se detallan los conceptos más importantes tratados en el presente trabajo. La sección 2.1 aborda los conceptos relacionados al AAS. La sección 2.2 analiza los conceptos del lenguaje SFC, mientras la sección 2.3 detalla algunos trabajos relacionados que se han realizado a la fecha relacionados al desarrollo de AAS y DT en la industria de procesos.

2.1 Asset Administration Shell

La organización Alemana Plattform Industrie 4.0 presenta un metamodelo para la especificación de AAS [3], del cual se presenta una versión resumida en la Figura 1.

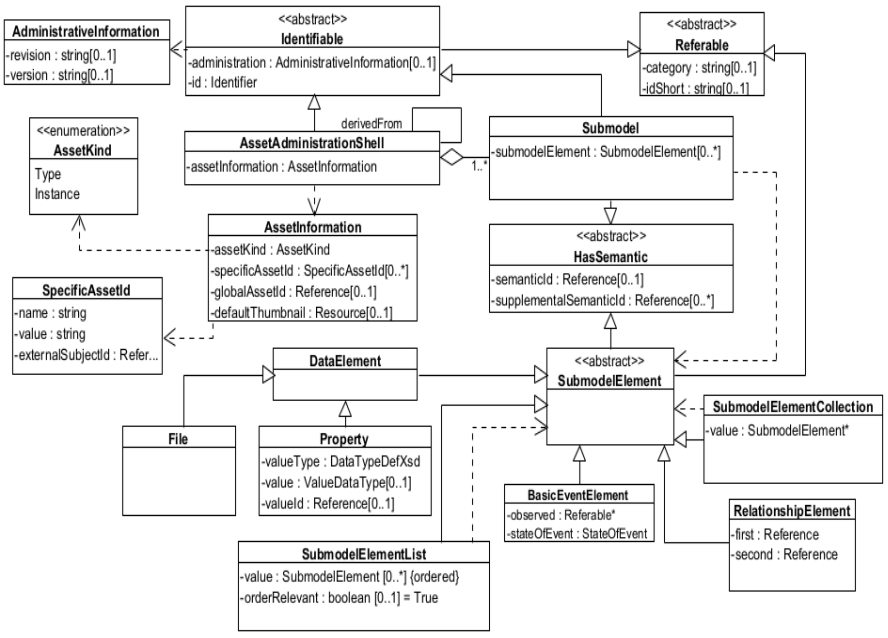

Fig. 1. Metamodelo del AAS. Tomado de [3].

La clase *AssetAdministrationShell* hace referencia a un solo asset por medio de la clase *AssetInformation*. El asset puede ser distinguido por medio del atributo *AssetKind* que puede tener el valor de *Type* (asset en estado de desarrollo o previa a su producción)

o *Instance* (asset en etapa posterior a su fabricación). Un asset puede o no, contar con uno o más *specifiAssetId* (identificación propia como número de serial o código RFID). La clase *AssetAdministrationShell* puede contar con cero o más *Submodel* que representen las características de un dominio, por ejemplo, puede existir un submodelo de capacidad que provea información acerca de las capacidades que tenga un asset para realizar determinadas operaciones, o un submodelo de identificación que provea información del asset para cumplir con determinada normativa. Un *Submodel* puede contar con cero o más *SubmodelElement* que puede ser: una *Property* (característica que permite describir un asset), un *File* (un archivo que contenga información del asset), un *BasicEventElement* (monitorea la ejecución de operaciones y actualiza valores de *submodelElements*), una *RelationshipElement* (especifica una relación entre *submodelElements*) o una colección de los anteriores abarcados en un *SubmodelElementColection* cuando tienen un conjunto estático de propiedades con una semántica definida o en una *SubmodelElementList* cuando se trata de conjuntos, bolsas o listas de *submodelElements*, entre otros.

Las clases *AssetAdministrationShell* y *Submodel*, son especializaciones de la clase *Identifiable*, por lo que heredan los atributos *administration* (información sobre la versión/revisión del AAS o del submodelo) y el atributo *id* que es un identificador que los hace globalmente único. Para identificar un AAS se puede usar un IRI (URL), mientras que para los *submodels* se pueden utilizar IRDI o IRI. Los AAS y submodelos necesitan ser globalmente únicos. El uso de estos identificadores es obligatorio para los AAS y los submodels. La clases *Submodel* y *SubmodelElement* especializan la clase *HasSemantic*, heredando así los atributos de *semanticId* y *supplementalSemanticId*. Es recomendable el uso de *semanticId* en ambos casos usando IRDI o IRI. La clase *SubmodelElement* especializa a la clase *Referable*. Los *Referables* son elementos que pueden ser identificados mediante *idShort*, pero esta identificación no es global. Los *Referable* contienen un atributo llamado *category*, que indica la clase de un elemento. Un elemento puede ser: 'constant' si, no varía a lo largo del tiempo, un 'parameter' si, una vez configurado, no cambia a lo largo del tiempo o 'variable', si es un elemento que requiere ser calculado durante una ejecución.

2.2 Sequential Function Chart

SFC es un lenguaje gráfico provisto por el estándar IEC 61131-3 para especificar programas en controladores lógicos. Estos programas son descompuestos en elementos llamados *POU* (Program Organization Unit). Un *POU* puede especificar un programa, un bloque de funciones o una función. El modelo conceptual propuesto para el SFC se muestra en la Figura 2. Un *POU* está compuesto por una o más redes (*Network*). Una *Network* está compuesta por dos o más pasos (*Step*), una o más transiciones (*Transition*) y enlaces (*Link*) que vinculan a pasos y transiciones. Una combinación alternada de pasos y transiciones da lugar a una secuencia. SFC permite representar secuencias simultáneas (ejecución de pasos en simultáneo), secuencias divergentes (se escoge la ejecución de una secuencia entre varias factibles), convergencias de secuencias simultaneas (fin de secuencia simultánea), convergencias de secuencias divergentes (fin de secuencia divergente) y Loop (enlazar una transición a un paso previo). En el presente

trabajo los conceptos de paso (*Step*) y transición (*Transition*) son generalizados como un nodo (clase *Node*). De igual forma, para poder representar los hitos de inicio de secuencias simultáneas y divergentes, así como sus convergencias, se crearon unas clases que las representen y fueron generalizadas en una clase llamada *Control* que también especializa a la clase *Node*. Los enlaces están representados por las asociaciones existentes entre los conceptos de paso, transición y control, tal como se observa en la Figura 2. Los extremos de asociación cuyo prefijo es *orig* denotan cual es el nodo predecesor, de igual forma el prefijo *end* indica cual es el nodo sucesor. Por ejemplo, el extremo asociación *origTS* indica que el link va de una transición (predecesora) a un paso (sucesor).

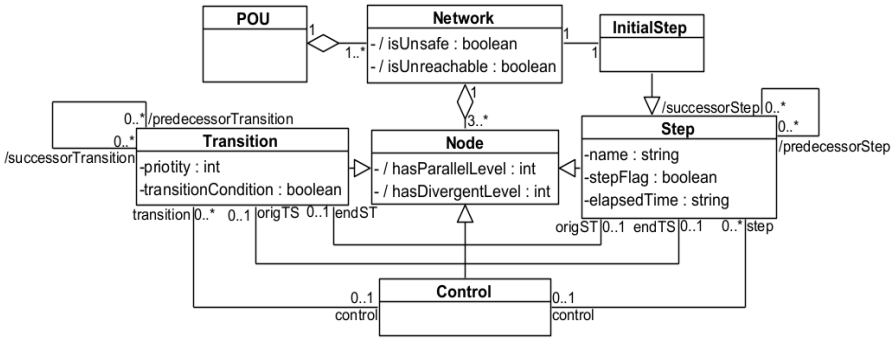

Fig. 2. Modelo conceptual propuesto para representar el lenguaje Sequential Function Chart.

Cuando un paso está activo, se ejecutan un conjunto de acciones. Estas acciones pueden ser escritas por medio de bloques de acciones (*ActionBlock*). Cada acción (*Action*) puede poseer un calificador (*Qualifier*) que indica cómo y cuándo se debe ejecutar la acción [10]. La relación entre los pasos y las acciones se muestra en la Figura 3.

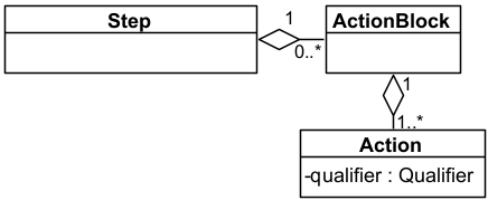

Fig. 3. Relación entre *Step*, *ActionBlock* y *Action*.

Durante la especificación de un modelo SFC se debe evitar generar redes inseguras (múltiples pasos activados en un mismo momento) y redes inalcanzables (redes que nunca serán ejecutadas en su totalidad) [10]. En las redes SFC los pasos son ilustrados mediante rectángulos, las transiciones mediante líneas horizontales entre pasos y los enlaces mediante líneas verticales entre nodos, tal como se ve en las redes de la Figura 4. También, se puede observar el inicio de una secuencia simultánea (doble línea horizontal posterior a la transición *t1*) y el fin de una secuencia simultánea (doble línea horizontal posterior a los pasos *S2* y *S4*).

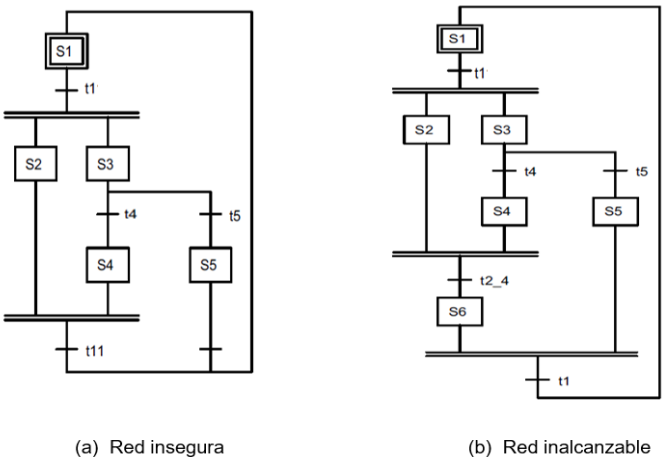

Fig. 4. (a) Red insegura; (b) Red inalcanzable. Tomado de [10].

En la Figura 4.a se muestra una red insegura. En ella, luego de que se dispare la transición *t1* (evaluarse *true*), se ejecutan en simultáneo los pasos *S2* y *S3*. Una vez ejecutado *S3*, se inicia una secuencia divergente en la que se debe escoger entre la secuencia que empieza con la transición *t4* o la que empieza con la transición *t5*. Si *t5* es evaluada *true*, se ejecuta el paso *S5* y una vez ejecutado se puede volver al paso *S1* y volver a empezar la secuencia simultánea con los pasos *S1 – S2* cuando aún *S2* puede no haberse terminado de ejecutar. La Figura 4.b muestra una red inalcanzable. Esta red se ejecuta igual que la anterior hasta el paso *S5*, una vez ejecutado este paso, se converge en un fin de secuencia simultánea con *S6*, pero la transición previa a *S6*, *t2_4*, nunca será evaluada *true*, dado que esa transición espera que tanto los pasos *S2* como *S4* sean ejecutadas y *S4* nunca será ejecutado (dado que se había activado *t5* en vez de *t4*). La diferencia entre las redes inseguras y las redes inalcanzables es que algunas secuencias de la primera pueden ser ejecutadas de tal forma que se puede volver a ejecutar la secuencia desde el principio, mientras que en la segunda red, existe al menos una secuencia que nunca se ejecuta [10]. El inconveniente en las redes inseguras e inalcanzables surge cuando transiciones que están dentro de una secuencia en paralelo o divergente son enlazadas con pasos que están por fuera de dichas secuencias. En la sección 3 del presente trabajo se presenta una propuesta para la detección de tales redes.

2.3 Trabajos relacionados

La aplicación de DT para soportar la toma de decisiones en la planeación y control de la producción ha ganado interés en los últimos años. En [6] se presenta un metamodelo para la creación de digital shadows en un proceso de moldeado de plástico por inyección que soporte la toma de decisiones en las actividades de planeación y control de la producción. El DS usa modelos de ingeniería que describen la estructura y comportamiento de los componentes del sistema de producción, sin embargo, dichos modelos no fueron desarrollados. En [11] se propone el uso de AAS para un proceso de producción

de jugo de naranja para la integración horizontal en la cadena de valor. El objetivo del AAS propuesto es poder informar al cliente en tiempo real, el estado de su orden. El AAS propuesto para el jugo de naranja, contiene una descripción del producto, pero contiene pocas descripciones y no está basado en ningún estándar. Una propuesta para la implementación de un DT que soporte las actividades de simulación y programación de la producción fue realizada en [5]. En su propuesta, los autores consideran un enfoque basado en la receta que permita la representación de las tareas y recursos necesarios para la producción. Sin embargo, ese trabajo sólo fue una propuesta que pretendía mostrar los beneficios de un DT en la industria alimenticia. A la fecha, no se han encontrado trabajos en la literatura que propongan la creación de submodelos de AAS a partir de estándares pre-existentes en la industria de procesos batch.

Los autores del presente trabajo previamente presentaron una propuesta [12], en la cual se propone un submodelo de AAS para la industria de procesos batch basado en la ISA 88, sin embargo, la estructura de procedimiento no fue tratada en dicho trabajo. Tal como se mencionó previamente, el objetivo del presente trabajo es el uso del lenguaje SFC para definir un submodelo de AAS que defina la receta de procedimiento a nivel de fase.

3 Propuesta de submodelo de AAS basado en SFC

El objetivo de la presente sección es sintetizar una representación de los conceptos relevantes al lenguaje gráfico SFC en un AAS. Para lograr dicho objetivo, inicialmente se presenta el modelado del lenguaje SFC. A partir de este modelo generamos un submodelo de AAS que representa el lenguaje SFC.

3.1 Propuesta de representación del lenguaje SFC

Los distintos constructores del lenguaje SFC fueron especificados empleando los lenguajes UML [13] y OCL [14] y especificados en el ambiente USE [15]. Los conceptos principales fueron presentados en la Figura 2 y Figura 3. Para solucionar el problema de las redes inseguras e inalcanzables introducidos en la Figura 4, se decidió proponer dos atributos derivados para los nodos del lenguaje SFC, llamados *hasParallelLevel* y *hasDivergentLevel* (ver Figura 2) y se propusieron reglas de derivación OCL para la especificación de los mismos. Un valor mayor a cero en estos atributos, indica que el paso, la transición o el nodo de control pertenece a una o más secuencias paralelas o divergentes, según sea el caso. El objetivo es poder garantizar que un nodo sólo se enlaza a otro nodo que pertenezca a su misma secuencia. La especificación de la regla de derivación creada para el atributo *hasParallelLevel* de la clase *Step* se muestra en la Figura 5. La notación empleada es la utilizada en el ambiente USE. En la regla se indica que, todo paso inicial (*InitialStep*) tiene un valor de *hasParallelLevel* (y *hasDivergentLevel*) igual a cero. Para el resto de pasos que conforman una red, el valor de este atributo siempre será el mismo que el valor del nodo que lo precede.

```
class Step < Node
attributes
hasParallelLevel: Integer    derive: if self.oclIsTypeOf(InitialStep) then 0 else
                             if self.origLoopS->notEmpty() then self.origLoopS.hasParallelLevel else
                             if self.origTS->notEmpty() then self.origTS.hasParallelLevel else
                             if self.origCOSS->notEmpty() then self.origCOSS.hasParallelLevel else
                             self.origISSS.hasParallelLevel
                             endif
```

Fig. 5. Extracto de regla de derivación para el atributo *hasParallelLevel* de la clase *Step.*

Para las transiciones la regla es similar, siempre hereda el valor del atributo del nodo predecesor. En cambio, para el caso de los controles, se produce el cambio del valor de los atributos *hasParallelLevel* y *hasDivergentLevel*, tal como se ve en la Figura 6. Allí, se especifica la derivación del atributo *hasParallelLevel* para el inicio de secuencia simultánea (*InitialSimultaneousSequence*).

```
class InitialSimultaneousSequence < Control
attributes
hasParallelLevel: Integer      derive: if self.origTISS->notEmpty() then self.origTISS.hasParallelLevel + 1
                                       else self.origTISS.hasParallelLevel
                                       endif
```

Fig. 6. Regla de derivación del atributo *hasParallelLevel* de la clase *InitialSimultaneous-Sequence.*

La regla en la Figura 6, indica que si un inicio de secuencia simultanea tiene una relación con una Transición predecesora (*origTISS*), entonces el valor para el atributo *hasParallelLevel* será el valor de este mismo atributo que tenía la transición predecesora, más uno. Luego, como se observa en la Figura 5, el paso siguiente al inicio de secuencia simultánea tomará como valor para este atributo, el mismo del inicio de secuencia simultanea y la transición que le sigue tomará el mismo del paso previo y así sucesivamente hasta el final de secuencia simultánea. La regla para el cálculo del valor del atributo *hasParallelLevel* para la clase fin de secuencia simultánea (*ConvergenceOfSimultaneousSequence*) se muestra en la Figura 7. En ella se indica que, si un final de secuencia simultánea tiene una relación con un paso predecesor, el cálculo del valor de su atributo *hasParallelLevel* va a ser el valor para este atributo que tiene el paso, menos uno. Mediante la derivación de estos atributos se puede determinar si un paso o una transición que pertenece a una secuencia en paralelo o divergente, no ha convergido en su respectivo final de secuencia.

```
class ConvergenceOfSimultaneousSequence < Control
attributes

hasParallelLevel: Integer       derive: if self.origSCOSS-> notEmpty() then
                                self.origSCOSS.hasParallelLevel -> min()-1 else 0
                                endif
```

Fig. 7. Regla de derivación del atributo *hasParallelLevel* de la clase *ConvergenceOfSimultaneousSequence*.

Cuando una transición se enlaza con un paso previo, está realizando un *Loop*, por ello se establecieron reglas para la clase *Loop*, la cual se puede observar en la Figura 8. En esta regla se deriva el atributo *hasParallelLevel* de un *Loop*, el cual es igual al menor valor para este mismo atributo que posean las transiciones que le preceden.

```
class Loop < Control
attributes
hasParallelLevel: Integer       derive: self.origTLoop->select(t|not self.successorsTLoop
                                        ->includes(t)).hasParallelLevel->min()
```

Fig. 8. Regla de derivación del atributo *hasParallelLevel* de la clase *Loop*.

Luego, para determinar si una red es insegura e inalcanzable, se especificaron los atributos *isUnsafe* e *isUnreachable*, respectivamente, para la clase *Network* y se determinaron reglas para su derivación. En la Figura 9 se especifica la regla de derivación del atributo *isUnsafe*. Esta regla evalúa *True* si las transiciones que se relacionan con el *Loop* pertenecen a distinta secuencia.

```
class Network
attributes
isUnsafe: Boolean  derive: self.control->select(c| c.oclIsTypeOf(Loop))->collect(c|c.oclAsType(Loop))}
                     ->exists(l| l.origTLoop->exists(t1, t2| t1.hasParallelLevel <> t2.hasParallelLevel))
                     or self.control->select(c| c.oclIsTypeOf(Loop))->collect(c|c.oclAsType(Loop))
                     ->exists(l| l.origTLoop->exists(t1, t2| t1.hasDivergentLevel <> t2.hasDivergentLevel))
```

Fig. 9. Regla de derivación del atributo *isUnsafe* de la clase *Network*.

Para el caso de las redes inalcanzables se especificó la regla de derivación de la Figura 10. En ella se indica que, una red es inalcanzable si el valor para el atributo *hasParallelLevel* o *hasDivergentLevel* para el *Loop* es menor a cero. Recapitulando las reglas anteriores, para que un *Loop* tenga un valor para el atributo *hasParallelLevel* o *hasDivergentLevel*, este debe ser el mínimo valor entre las Transiciones que se relación con el *Loop*. Las Transiciones que preceden al *Loop* pudieron tomar este valor de un fin de secuencia simultanea que estaba relacionada con un paso predecesor que tenía valor de *hasParallelLevel* o *hasDivergentLevel* igual a cero, es decir un paso que estaba convergiendo sin haber pertenecido a un inicio de secuencia paralela o divergente.

```
isUnreachable: Boolean  derive: self.control->select(c| c.oclIsTypeOf(Loop))->collect(c|c.oclAsType(Loop))
                                ->exists(l| l.hasParallelLevel<0 or l.hasDivergentLevel<0)
```

Fig. 10. Regla de derivación del atributo *isUnreachable* de la clase *Network*.

El modelo propuesto fue especificado en USE con el objetivo de poder validar la propuesta. La Figura 11 muestra la implementación en USE de una red insegura, en la que se puede observar como el programa mediante la ejecución de las reglas creadas, evalúa *True* el atributo *isUnsafe* del objeto *N1*, de tipo *Network*. La formalización explicada en la presente sección permite la construcción de modelos válidos y la definición inequívoca de los conceptos propios del lenguaje SFC, posibilitando su reúso en diferentes sistemas.

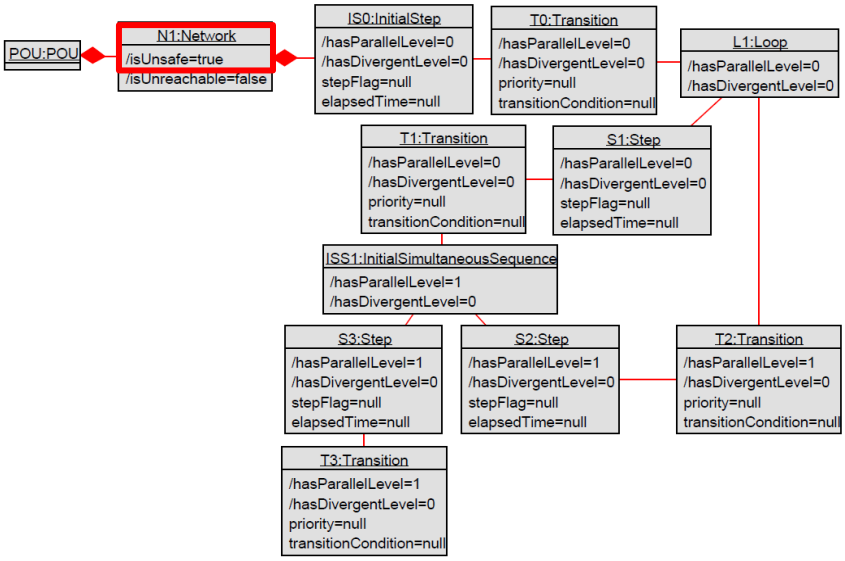

Fig. 11. Implementación de red insegura en USE.

3.2 Construcción del submodelo de AAS

A partir de la conceptualización realizada, detallada en las secciones previas, se instancia el metamodelo de AAS para especificar el submodelo propuesto, basado en el estándar ISA-88. En la Figura 12, se incluye tal instancia de *Submodel*, denominada ISA-*88BasedSubmodel*. El submodelo está definido por el *submodelElement ProceduralStructureSEC*.

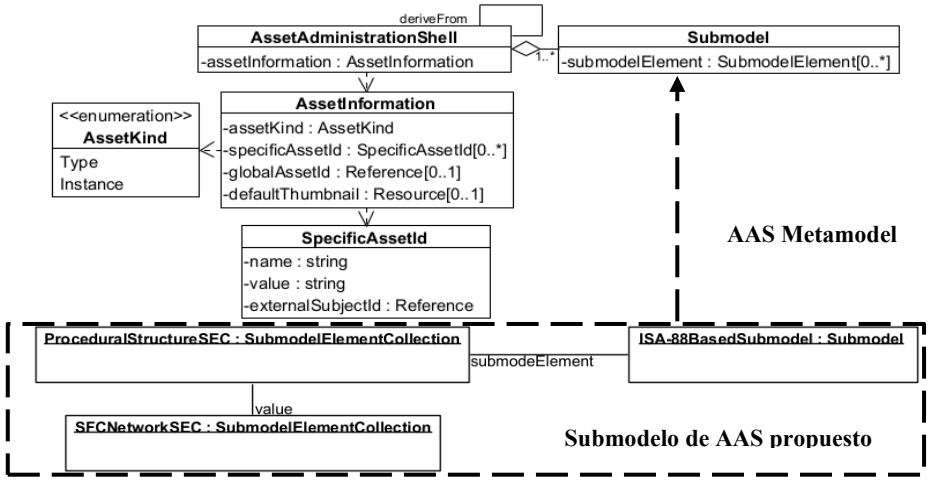

Fig. 12. Submodelo Propuesto.

ProceduralStructureSEC representa la colección de toda la información relevante que conforma la receta de procedimiento. Dado que se definió este objeto como una instancia de *SubmodelElementCollection* (*SEC*), toda esta información relevante va a estar vinculada a este objeto mediante la propiedad *value*. En SFC el procedimiento es representado por medio de redes. De esta manera, el procedimiento de la receta está representada por la red *SFCNetworkSEC* y vinculada a *ProceduralStructureSEC* mediante la propiedad *value* (Figura 12). Una red está compuestas por (conjuntos de) pasos, transiciones, controles y las relaciones (enlaces) entre estos nodos. El Metamodelo de AAS permite la representación de conjuntos por medio de *SubmodelElementList*. En consecuencia, en la presente propuesta, los pasos, transiciones, controles y enlaces son agrupados en las listas *StepSEL*, *TransitionSEL*, *ControlSEL* y *LinkSEL*, respectivamente. Dado que el metamodelo de AAS no tiene el concepto de composición empleado en el modelo realizado en UML, estos conjuntos serán incorporados a la red (*SFCNetwrokSEC* en Figura 13) como valores de la collection (*value*). Además, en la propuesta de formalización que se presentó en la sección 3.1 se especificaron en la clase *Network* los atributos *isUnsafe* e *isUnreachable*, los cuales son características propias de esta clase, por lo cual, pueden ser representados por medio de propiedades (concepto *Property* en el metamodelo de AAS introducido en la Figura 1). Estas propiedades también son parte del *value* de *SFCNetworkSEC*.

Cada nodo contiene un conjunto de atributos, por lo tanto, es representado mediante un *SubmodelElementColection*. Por ejemplo, la clase *Step* (Figura 2) contiene los atributos *elapsedTime*, *stepFlag*, *hasDivergentLevel* y *hasParallelLevel*, estos dos últimos heredados de *Node*. Además, cada *Step* tiene un nombre. De esta manera, un *Step* es representado por un *SEC*, denominado *StepSEC* (Figura 13). Los atributos de la clase *Step* son representados como *Properties* dentro del submodelo de AAS y son agrupados dentro del *SEC StepSEC*. La composición de *SFCNetworkSEC* y *StepSEL* se muestra en la Figura 13.

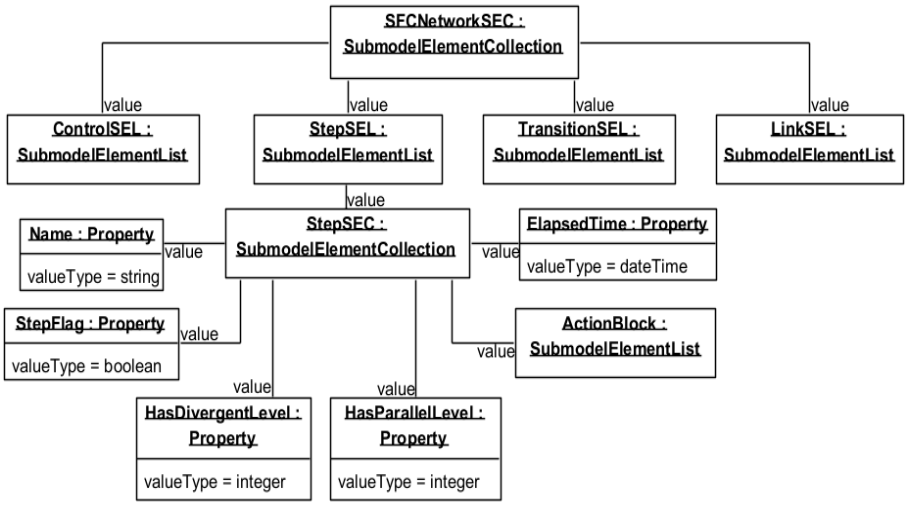

Fig. 13. Composición de una *SFCNetworkSEC* y *StepSEL*.

Con las transiciones se procede de una manera similar. Cabe destacar que en el metamodelo de AAS no podemos representar la generalización/especialización, en consecuencia, no es posible representar las clases más abstractas, como *Node* (Figura 2) y se debe repetir la definición de sus propiedades en cada uno de las clases concretas. De esta manera, el conjunto de atributos de cada transición es agrupado en un *SEC*. Los atributos de una transición son: *priority*, *transitionCondition*, *hasParallelLevel* y *hasDivergentLevel* (Figura 2), estos dos últimos se representan igual que para el *StepSEC*. Dado que la prioridad se asigna con un valor numérico, el *valueType* (Figura 1) de este atributo es un *integer*. Como su nombre lo indica, la *transitionCondition* evalúa que se den las condiciones para disparar la Transición, la misma es definida como una propiedad (instancia de *Property*) cuyo *valueType* es *boolean*. La Figura 14, muestra la representación de *TransitionSEC*. Además, se incluye la definición de uno de los enlaces, *RorigSTendST*. Los enlaces han sido modelados en AAS como instancias de *RelationshipElement* (Figura 1), donde los extremos de la asociación son representados por *first* y *second* (Figura 1). De esta manera, una *RorigSTendST* permite representar un arco desde un paso (*StepSEC*) a una transición (*TransitionSEC*). De forma análoga fueron definidos los distintos enlaces entre los nodos pasos, transiciones y de control de la Figura 2.

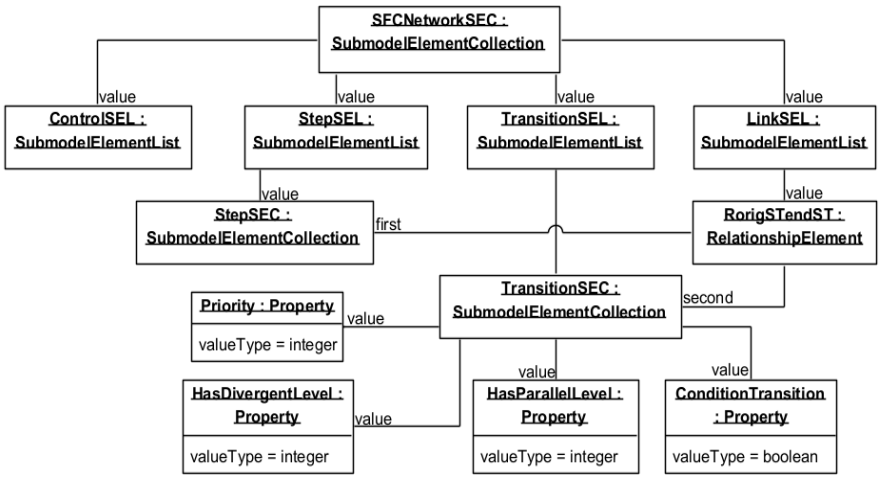

Fig. 14. Representación de *TransitionSEC* y *RelationshipElement*.

4 Discusión

A continuación, se presentan nuestras principales reflexiones sobre la propuesta del submodelo de AAS basado en el estándar ISA-88. En el presente trabajo se presentaron avances en comparación al trabajo previo [12]. El presente trabajo presenta los lineamientos para la especificación de procedimientos de producción en un submodelo de AAS. A pesar de haber realizado una propuesta para abordar el problema de las redes inseguras e inalcanzables, sólo es posible hacer una validación previa en la herramienta USE, sin embargo, no es factible validar la red al momento de especificar el submodelo

de AAS. Las reglas de derivación previamente especificadas deberían ser programadas al momento de implementar el AAS. Para poder interoperar, los AAS y sus submodelos deben ser estandarizados, al igual que las propiedades de los assets. Las propiedades tratadas en este trabajo (*priority*, *hasDivergentLevel*, *hasParallelLevel*, entre otras) no están estandarizadas. La organización Plattform Industrie 4.0 en asociación con ECLASS están trabajando para poder desarrollar una metodología para estandarizar propiedades que no se encuentran en el diccionario ECLASS [16].

A pesar de estar enfocado en la industria de procesos, la propuesta formulada en el presente trabajo puede ser utilizada en otro tipo de industrias.

5 Conclusión y trabajos futuros

Los desafíos que enfrentan las empresas en la industria de procesos para hacer frente a la personalización de los productos, requiere una alta flexibilidad en las plantas de producción. La implementación de Digital Twins implementados mediante AAS puede ayudar en las actividades de la planeación y programación de la producción. Para lograr implementar un DT se requiere una representación real del proceso productivo. En especial, en el presente trabajo se ha presentado una propuesta para la representación del procedimiento de producción en la industria de procesos batch mediante el uso de SFC. Para poder usar el SFC en la especificación del submodelo de procedimiento, se ha realizado una formalización del lenguaje SFC y mediante reglas se logró poder detectar las redes inseguras e inalcanzables que pueden ser creadas usando SFC.

En el futuro, se procederá a continuar con la especificación del submodelo de AAS basado en ISA-88. Con la presentación de este trabajo, quedaría pendiente por abordar la representación de la receta de requerimientos de equipos en un submodelo de AAS. Una vez completado el submodelo, la intención es poder presentarlo ante la organización Plattform Industrie 4.0 para realizar la estandarización del mismo. Una vez estandarizado, una plantilla del submodelo estará disponible para su uso en la industria de procesos.

References

[1] J. Litster and I. D. L. Bogle, “Smart Process Manufacturing for Formulated Products,” *Engineering*, vol. 5, no. 6, pp. 1003–1009, 2019, doi: 10.1016/j.eng.2019.02.014.

[2] S. Weyer, M. Schmitt, M. Ohmer, and D. Gorecky, “Towards Industry 4.0 - Standardization as the crucial challenge for highly modular, multi-vendor production systems,” *IFAC-PapersOnLine*, vol. 48, no. 3, pp. 579–584, 2015, doi: 10.1016/j.ifacol.2015.06.143.

[3] S. Bader, E. Barnstedt, H. Bedenbender, M. Billman, B. Boss, and A. Braunmandl, “Details of the Asset Administration Shell Part 1 - The exchange of information between partners in the value chain of,” vol. 0, p. 473, 2018.

[4] W. Kritzinger, M. Karner, G. Traar, J. Henjes, and W. Sihn, “Digital Twin in manufacturing: A categorical literature review and classification,” *IFAC-*

PapersOnLine, vol. 51, no. 11, pp. 1016–1022, 2018, doi: 10.1016/j.ifacol.2018.08.474.

[5] A. Koulouris, N. Misailidis, and D. Petrides, "Applications of process and digital twin models for production simulation and scheduling in the manufacturing of food ingredients and products," *Food Bioprod. Process.*, vol. 126, pp. 317–333, Mar. 2021, doi: 10.1016/j.fbp.2021.01.016.

[6] P. Sapel *et al.*, "Towards digital shadows for production planning and control in injection molding," *CIRP J. Manuf. Sci. Technol.*, vol. 38, no. August, pp. 243–251, Aug. 2022, doi: 10.1016/j.cirpj.2022.05.003.

[7] C. Zhang, W. Xu, J. Liu, Z. Liu, Z. Zhou, and D. T. Pham, "A Reconfigurable Modeling Approach for Digital Twin-based Manufacturing System," *Procedia CIRP*, vol. 83, pp. 118–125, 2019, doi: 10.1016/j.procir.2019.03.141.

[8] ISA, *Batch Control, Part 1: Models and Terminology, ISA-88.01-1995 (R2006)*. 2006.

[9] International Electrotechnical Commission (IEC) and (IEC), *Programming languages*, Document I. 2013.

[10] K. H. John and M. Tiegelkamp, *IEC 61131-3: Programming Industrial Automation Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[11] J. H. Hang, Y. M. Lim, and G. S. Bhd, "Towards Mass Individualized Production : RAMI 4 . 0 Asset Data Channelling for Manufacturing Value Chain Connectivity," vol. 0, pp. 225–231, 2021.

[12] R. Pairol-Fuentes, J. Alvarado-Domínguez, M. Vegetti, L. Roldán, and S. Gonnet, "Modelo para la digitalización de activos de la industria batch basado en el estándar ISA 88," *Memorias Las JAIIO*, vol. 8, no. 13, pp. 33–46, 2022, [Online]. Available: https://publicaciones.sadio.org.ar/index.php/JAIIO/article/view/316/263.

[13] OMG, "UML Unified Modeling Language', Version 2.5.1, 2017. [online]. Available: https://www.omg.org/spec/UML/.

[14] OMG, "OCL Object Constraint Languaje", Version 2.4, 2014. [online]. Available: https://www.omg.org/spec/OCL/.

[15] M. Gogolla, F. Büttner, and M. Richters, "USE: A UML-based specification environment for validating UML and OCL," *Sci. Comput. Program.*, vol. 69, no. 1–3, pp. 27–34, 2007, doi: 10.1016/j.scico.2007.01.013.

[16] "Whitepaper Modelling the Semantics of Data of an Asset Administration Shell with Elements of ECLASS," 2021, [Online]. Available: https://eclass.eu/fileadmin/Redaktion/pdf-Dateien/Broschueren/2021-06-29_Whitepaper_PlattformI40-ECLASS.pdf.