Universidad Andrés Bello

Facultad de Ingeniería

Escuela de Informática

Proyecto de Título:

Soporte automático para la generación de perfiles UML asociados a la definición modelos de Safety

Autor:

Jorge Farías Verdugo

Profesor Guía:

Giovanni Giachetti Herrera

Santiago de Chile, Chile.

2016

Contenidos

Índice de Tablas 3
Índice de Ilustraciones 3
Capítulo 1: Introducción 6
1.1 Motivación 7
1.2. Identificación del Problema 8
1.3. Objetivos 9
1.3.1. Objetivo General 9
1.3.2. Objetivos Específicos 9
1.4. Alcance 10
Capítulo 2: Estado del Arte 11
2.1. Marco Teórico 11
2.2. Estado del Arte 21
Capítulo 3: Metodología y Planificación 30
3.1. Enfoque Metodológico 30
3.2. Planificación del Proyecto 34
Capítulo 4: Desarrollo de la Solución 37
4.1. Introducción a Modelización Conceptual 37
4.2. Desarrollo del Proyecto 38
4.2.1. Definición del Metamodelo MOF que Representa el Estándar Safety 39
4.2.2. Generar Perfil para Determinar la Equivalencia de las Clases 48
4.2.3. Definición de Reglas ATL para la generación Automática de Perfil UML de Safety 50
4.2.4. Evaluación del Correcto Funcionamiento de las Reglas ATL 75
Capítulo 5: Guía de Usuario 84
Capítulo 6: Conclusiones y Trabajo Futuro 96
Capítulo 7: Bibliografía 98

Índice de Tablas

Tabla 1: Métricas para los Objetivos Específicos. 10
Tabla 2: Tareas Principales del Proyecto. 35
Tabla 3: Definición de Conceptos del paquete Conceptos de procesos. 78

Índice de Ilustraciones

Ilustración 1: Ejemplo de clases UML 15
Ilustración 2: Ejemplo de Asociaciones UML. 16
Ilustración 3: Ejemplo agregación UML. 17
Ilustración 4: Ejemplo de composición UML. 18
Ilustración 5: Ejemplo de generalización UML. 18
Ilustración 6: Ejemplo de Paquetes UML. 19
Ilustración 7: Ejemplo de enumeration UML. 20
Ilustración 8: Ejemplo Estereotipo UML. 21
Ilustración 9: Proceso de revisión sistemática. Fuente: Presentación profesor Giovanni Giachetti. 21
Ilustración 10: Esquema del proceso de Integración. Fuente (Giachetti et al., 2009). 27
Ilustración 11: Metodología para la creación de pruebas de un estándar de seguridad. Fuente (Panesar-Walawege et al., 2013). 28
Ilustración 12: Paquete conceptos de sistema. 40
Ilustración 13: Paquete conceptos de riesgo. 41
Ilustración 14: Paquete Conceptos de Requerimientos. 42
Ilustración 15: Paquete Conceptos de Procesos. 42
Ilustración 16: Paquete Conceptos de Artefactos. 43
Ilustración 17: Paquete conceptos de emisión. 44
Ilustración 18: Paquete conceptos de gestión de configuración. 44
Ilustración 19: Paquete conceptos de justificación. 45
Ilustración 20: Paquete conceptos de orientación. 45
Ilustración 21: Paquete conceptos dominio especifico. 46
Ilustración 22: Modelo Conceptual Estándar IEC 61508 y sus relaciones. 47
Ilustración 23: Perfil UML IMProfile. 49
Ilustración 24: Fragmento metamodelo Safety después de aplicar el perfil IMProfile. 50
Ilustración 25: Ejemplo de cabecera de una ATL. 53
Ilustración 26: Ejemplo de librería ATL. 53
Ilustración 27: Ejemplo de Helper ATL. 54
Ilustración 28: Ejemplo de matched Rules ATL. 55
Ilustración 29: Ejemplo de called rules ATL. 56
Ilustración 30: Ejemplo de lazy rule ATL. 57

Ilustración 31: Ejemplo de unique lazy rule ATL. 57
Ilustración 32: Ejemplo genérico para la Regla 1. 58
Ilustración 33: Regla genStereotypes primera parte. 59
Ilustración 34: Regla genStereotypes segunda parte. 60
Ilustración 35: Helper getTargetElement(). 61
Ilustración 36: Ejemplo genérico para la regla 2. 62
Ilustración 37: Código ATL que implementa la regla 2. 63
Ilustración 38: Código ATL que implementa la regla 2 64
Ilustración 39: Ejemplo genérico de la regla 3. 65
Ilustración 40: Código ATL que implementa la regla 3, regla genEnumeration. 66
Ilustración 41: Código ATL que implementa la regla 3, regla genEnumerationLiteral. 66
Ilustración 42: Ejemplo genérico para las reglas 4, 5 y 6. 68
Ilustración 43: Código ATL que implementa la regla 4. Regla genGeneralization. 68
Ilustración 44: Código ATL que implementa la regla genExtension. 70
Ilustración 45: Cabecera Transformación ATL. 70
Ilustración 46: Código ATL implementado para el helper hasStereotype. 71
Ilustración 47: Código ATL implementado para el helper getTargetMetaclasses. 72
Ilustración 48: Código ATL implementado para el helper getImportedTypes. 72
Ilustración 49: Código ATL que implementa la regla genProfile. 73
Ilustración 50: Código ATL implementado para la regla genPackage. 74
Ilustración 51: Código ATL implementado en la regla ref_classes. 74
Ilustración 52: unique lazy rule imp_type. 75
Ilustración 53: Package Process Concepts y sus relaciones. 76
Ilustración 54: Ejemplo de mapeo entre la clae Phase y la metaclase class. 78
Ilustración 55: Perfil UML generado de forma automática. 79
Ilustración 54: Perfil UML generado de forma automática. 79
Ilustración 57: Perfil UML de Safety generado automáticamente. 81
Ilustración 58: Perfil UML de Safety generado manualmente. 81
Ilustración 59: Un fragmento del modelo de dominio de un sistema de control de producción submarina. 83
Ilustración 60: Modelo de dominio después de aplicar el perfil UML. 84
Ilustración 61: Abrir proyecto ATL. 85
Ilustración 62: Ventana para importar proyectos. 86
Ilustración 63: Proyecto listo para ser utilizado. 86
Ilustración 64: Acceso a configuración de ejecución de la regla ATL. 87
Ilustración 65: Run Configurations parte 1. 88
Ilustración 66: Run Configurations parte 2. 88
Ilustración 67: Procedimiento para cargar modelo. 89
Ilustración 68: Cuadro que se carga la ruta del archivo que contiene al perfil. 89
Ilustración 69: Ejemplo de procedimiento. 90
Ilustración 70: Ventana Apply Prrofile, para aplicar el perfil IMProfile. 90
Ilustración 71: Metamodelo de entrada, una vez aplicado el perfil. 91

Ilustración 72: Definición del perfil UML. 92
Ilustración 73: Definición del perfil UML. 92
Ilustración 74: Procedimiento para cargar el perfil UML. 93
Ilustración 75: Ventana Load Resource, para cargar el perfil UML. 93
Ilustración 76: Procedimiento para aplicar el perfil UML al modelo de dominio. 93
Ilustración 77: Procedimiento para aplicar el perfil UML al modelo de dominio. 94
Ilustración 78:Procedimiento para aplicar los estereotipos en las clases del modelo de dominio. 94
Ilustración 79: Procedimiento para aplicar los estereotipos en las clases del modelo de dominio. 95

Capítulo 1: Introducción

En la construcción de sistemas críticos es fundamental contar con estándares de calidad adecuados para garantizar su correcto funcionamiento. Entendemos como sistema crítico a todo “sistema cuyo fallo que puede ocasionar pérdidas económicas significativas, daños físicos, y en el peor de los casos, amenaza a vidas humanas” (Knight, 2002; Smith & Simpson, 2016). Este tipo de sistemas han aumentado su complejidad con el tiempo, incorporando cada vez mayor automatismo, controlado por productos de software.

Por lo tanto, toma especial relevancia poder controlar y certificar la seguridad funcional (Safety) de la componente software de estos sistemas, para que no afecte el desempeño general, ni ponga en riesgo finalmente a las personas que utilizan o interactúan con sistemas críticos.

La seguridad funcional es parte de la seguridad global de un sistema o pieza de un equipo, y su foco generalmente es electrónico y relacionado a software. Se centra en aspectos de seguridad que se relacionan con la función de un dispositivo o sistema, garantizando que funcione correctamente en respuesta a los comandos que recibe. En un enfoque sistémico, la seguridad funcional identifica condiciones potencialmente peligrosas, situaciones o acontecimientos que puedan ocasionar un accidente que podría perjudicar a alguien o destruir algo. Permite acciones correctivas o preventivas para evitar o reducir el impacto de un accidente. La seguridad funcional tiene como objetivo, llevar al riesgo a un nivel tolerable y reducir su impacto negativo (International Electrotechnical, 2000).

Los sistemas críticos de seguridad en los diferentes dominios deben ser certificados por estándares internacionales, donde encontramos un estándar genérico que se puede aplicar en cualquier dominio IEC 61508, de este se derivan estándares internacionales para dominios específicos, dentro de los

cuales identificamos los siguientes: ISO 26262 (Automóvil), IEC 60601 (Medicina), EN 50129 (Trenes), entre otros.

En el presente trabajo se definirá un soporte automático para la generación de perfiles UML en consistencia al estándar Safety (IEC 61508), permitiendo la reducción de errores o ambigüedades presentes en la generación manual de perfiles UML.

En las siguientes secciones presentamos lo siguiente: Sección 2 identificación de problema a resolver. Sección 3 objetivos generales y específicos. Sección 4 alcance del proyecto tesis. Sección 5 marco teórico. Sección 6 estado del arte realizado para identificación de modelos, herramientas o conceptos que apoyen en el desarrollo del proyecto. Sección 7 enfoque metodológico. Finalmente, sección 8 plan de tesis.

1.1 Motivación

En el desarrollo de software críticos para la seguridad, se consideró como alternativa para documentar o diseñar las pruebas de seguridad en base al desarrollo dirigido por modelo (MDE), ya que es una herramienta utilizada en varios ámbitos de la ingeniería y tiene argumentos sólidos que permite definir con mayor solidez modelos de dominio.

En este ámbito se utilizan metamodelos para definir el diseño de la aplicación, perfiles UML para diseñar los conceptos específicos que no son considerados por el lenguaje unificado de modelado (UML), permitiendo definir de manera íntegra la aplicación o dominio que se desea certificar.

Sumado al diseño de la aplicación se debe considerar el o los estándares internacionales presentes, garantizando las propiedades necesarias de la aplicación, reduciendo el riesgo para los usuarios.

La totalidad de la definición de la aplicación en la actualidad se genera de forma manual, lo que requiere un gran esfuerzo y de varios especialistas conectados

(ingenieros, especialistas del estándar, entre otros). Con el presente trabajo se pretende reducir ese esfuerzo automatizando la generación de perfiles de acuerdo al estándar IEC 61508. Sin embargo, se puede interpolar a cualquier estándar ya que son derivados del estándar mencionado.

1.2. Identificación del Problema

Los sistemas críticos deben cumplir con estándares de seguridad para entregar garantía que no representen un riesgo para los usuarios, propiedades o el medio ambiente. El cumplimiento de la seguridad es una actividad que requiere mucho esfuerzo, como los estándares contienen numerosas páginas y se suele tener profesionales para demostrar el cumplimiento de miles de criterios relacionados con la seguridad. Además, los textos de la norma pueden ser ambiguos, contradictorios, y difícil de entender, lo que determina una mayor dificultad para definir los criterios de seguridad.

Durante el desarrollo de software crítico, las pruebas de seguridad aún se definen de forma manual, documentando los criterios de seguridad que cumplen con el estándar aplicado en el dominio del software. Para la definición de las pruebas de seguridad se utiliza el enfoque de desarrollo dirigido por modelos (MDE), donde se definen metamodelos (contiene los conceptos comunes del estándar aplicado en cualquier dominio) y perfiles UML (contiene los conceptos específicos del dominio de aplicación).

En el presente trabajo se presenta un enfoque que automatiza la generación de perfiles UML en base a metamodelos MOF que son definidos en cumplimiento al estándar safety (IEC 61508), además se presenta un modelo de verificación del metamodelo MOF esté alineado al estándar.

1.3. Objetivos

1.3.1. Objetivo General

Garantizar la completitud y la consistencia de un perfil UML generado automáticamente a partir de un metamodelo de Safety basado en el estándar IEC 61508.

1.3.2. Objetivos Específicos

A continuación, se detallan los objetivos específicos, con los cuales se busca validar el cumplimiento del objetivo general del proyecto:

1. Asegurar que el modelo de entrada para la generación del perfil UML para safety, esté alineado al estándar IEC 61508.

2. Garantizar la correcta identificación de extensiones UML que serán implementadas en el perfil UML para Safety.

3. Automatizar la generación de perfiles UML, de acuerdo al estándar IEC 61508, reduciendo errores de definición que conllevan una mala interpretación y representación de los modelos de Safety.

Objetivo Específico	**Métrica**	**CEM**
OE1	Identificación de características necesarias para modelos de Safety a partir de un metamodelo MOF de entrada.	Número de Características Identificada = 10
OE2	Identificación de diferencias entre metamodelo MOF de entrada y metamodelo UML.	Porcentaje de Diferencia

		encontradas = 100%
OE3	Estereotipos y valores etiquetados generados correctamente en un perfil UML a partir de un metamodelo MOF de Safety.	Porcentaje de Completitud del Perfil generado = 100%

Tabla 1: Métricas para los Objetivos Específicos.

Según lo establecido en la tabla 1, el criterio de éxito para el objetivo específico OE1, es identificar en el metamodelo MOF de entrada 10 características necesarias para modelos safety. Como prueba se utilizarán dos metamodelos MOF. Para el objetivo específico OE2, el criterio de éxito está definido al identificar todas las diferencias existentes entre el modelo MOF de entrada y un metamodelo UML. Como prueba se utilizarán dos metamodelos MOF. Por último, el criterio de éxito para el objetivo específico OE3 es definir los valores etiquetados y estereotipos generados correctamente en un perfil UML en su totalidad. Como prueba se utilizarán 10 modelos UML para verificar la correcta generación de perfiles UML.

1.4. Alcance

El proyecto busca automatizar la generación de perfiles para el estándar safety. En particular, nos centraremos en el dominio de aviones y automóviles, donde ya se encuentran perfiles y metamodelo definidos que serán utilizados como casos de estudio para validar la propuesta desarrollada. Por lo tanto, nuestro proyecto estará finalizado cuando el perfil generado cumpla los aspectos de modelado en el dominio de nuestro caso de estudio.

Capítulo 2: Estado del Arte

2.1. Marco Teórico

Para el presente trabajo de tesis, se darán a conocer los conceptos claves que son tratados durante el desarrollo del trabajo. Los conceptos son los siguientes:

- Estándar IEC 61508.
- Perfiles UML.
- Metamodelo MOF.
- Metamodelo.
- Transformación de Modelo.

A continuación, se describe brevemente cada concepto.

Estándar IEC 61508

El estándar consiste en establecer los requerimientos necesarios que nos asegura que un sistema se diseña, implementa, opera y se mantiene, para proveer el nivel necesario de integridad de seguridad (SIL). El estándar identifica 4 niveles de seguridad, donde el nivel 1 representa el menor nivel de seguridad, y el nivel 4 establece la mayor certificación de seguridad.

El estándar internacional consiste en 7 partes, y se identifican como:

- IEC 61508-1, Requerimientos generales.
- IEC 61508-2, Requerimientos para eléctricos / electrónicos / electrónicos programables relacionados con la seguridad de sistemas.
- IEC 61508-3, Requerimientos de software.
- IEC 61508-4, Definiciones y abreviaciones.
- IEC 61508-5, Ejemplos de métodos para la determinación de niveles de integridad de seguridad.
- IEC 61508-6, Directrices sobre la aplicación de IEC 61508-2 y IEC 61508-3.

- IEC 61508-7, Descripción de las técnicas y medidas.

Este estándar internacional presenta un dominio genérico, donde se derivan estándares específicos para diferentes dominios tales como: ISO 26262 (Automóvil), IEC 60601 (Medicina), EN 50129 (Trenes), entre otros.

Para el caso del trabajo, el estándar IEC 61508 debe estar definido explícitamente en un metamodelo MOF definido como entrada, según lo especificado en el OE 1.

Perfiles UML

Los perfiles UML representan una solución ligera que permite extender el Metamodelo UML para un dominio específico. Un perfil permite generar nuevos conceptos, notaciones y restricciones mediante estereotipos específicos del contexto, atributos y restricciones. Los estereotipos son un medio de ampliar la base de metaclase de un modelo UML. [2]

Para efectos de este trabajo, un perfil UML se utilizará para definir las características necesarias para modelos safety con el objetivo de comprobar que el metamodelo MOF de entrada este alineado al estándar IEC 61508. Además, un perfil UML en relación al objetivo general será definido de forma automática a partir de un metamodelo MOF de entrada.

Metamodelo MOF

Es un lenguaje estándar de Metamodelado, diseñado por la OMG (Object Management Group), que posee herramientas de apoyo para la generación de tecnologías orientadas a modelos y un formato de intercambio estandarizado, lo cual facilita la aplicación de tecnologías basadas en este estándar. El uso de MOF para la especificación de lenguajes de modelado también facilita la identificación de similitudes estructurales (equivalencias) entre construcciones conceptuales. MOF está diseñado como una arquitectura de 4 capas, en la cual

se define como un modelo M3 que se compone a sí mismo, en donde cada sub elemento representa una instancia y posee una correspondencia a una capa superior de su arquitectura, definiendo así la estructura y sintaxis del modelo.

Metamodelo

Un metamodelo define la estructura que todos los modelos UML deben tener, es decir, un metamodelo es una abstracción de un lenguaje de modelado y el modelo es una instancia de un metamodelo en conformidad de la estructura definida.

El metamodelo solo define una estructura y no una semántica. Generalmente se define en un diagrama de clase, donde se instancian clases, atributos, operaciones y componentes.

Para el presente trabajo, los metamodelos construidos corresponden al metamodelo MOF de entrada (en base al estándar IEC 61508) y el metamodelo UML correspondiente al lenguaje UML que son la base para generar automáticamente a través de reglas el perfil UML.

Transformación de Modelos

Según la *OMG* en el contexto de *MDA* (*model-driven architecture*) define una transformación como; el proceso de convertir un modelo en otro modelo del mismo sistema.

Se distinguen dos tipos de transformaciones de modelo: *Modelo a Modelo* y *Modelo a Texto*.

Modelo a Modelo: Consiste en crear elementos en el modelo de destino. Los elementos presentes en el modelo de origen, se asignan a elementos del modelo de destino.

Modelo a Texto: Crea un texto arbitrario. Elementos en el modelo de origen mapea al fragmento de texto arbitrario. El texto carece de estructura, la transformación modelo a texto es más difícil de analizar.

En el presente trabajo se utiliza la transformación Modelo a Modelo, con el objetivo de transformar el metamodelo MOF de entrada definido bajo los términos de safety, en un perfil UML para un caso de estudio en particular.

Unified Modeling Languaje (UML)

El lenguaje UML hoy en día es el más utilizado en la modelización conceptual, ya que cuenta con una gran documentación donde apoyarse y es impulsado por la OMG (Object Management Group), organización que se encarga de promover estándares relacionados con los sistemas orientados a objetos.

UML consiste en un estándar que permite crear esquemas, diagramas y documentación relativa al desarrollo de software.

UML define trece tipos de diagramas, que son divididos en tres categorías, y son las siguientes:

- Diagrama de Estructura: incluye los diagramas de clases, diagrama de objetos, diagrama de componentes, etc.
- Diagrama de Comportamientos: Incluye diagrama de casos de uso, diagrama de actividad y diagrama de máquina de estado.
- Diagrama de Interacción: Todos los derivados del diagrama de comportamiento más general, incluye diagrama de secuencia, diagrama de comunicación, cronograma y esquema de interacción.

En las siguientes secciones, se describen los elementos utilizados en el desarrollo de la generación automática de perfiles UML. Presentarlos todos esta fuera del alcance de este proyecto, ya que el estándar UML es extenso. El lector que desea profundizar en los demás elementos del estándar, puede revisar la especificación formal en (Group, 1997-2016).

Entidad

Se denomina entidad al elemento clase de un diagrama de clase, y se define como un concepto cuyas instancias en un momento determinado corresponde a objetos existentes dentro del dominio que se está trabajando. Una clase en UML está compuesta por su nombre, atributos y funciones.

Si consideramos como ejemplo la representación de un sistema de información que ayuda en el proceso de asignación de ramos en alguna universidad. En este contexto interactúan clases como alumno, asignatura y entre otras. A modo de ejemplo modelamos este par de clases, empleando UML.

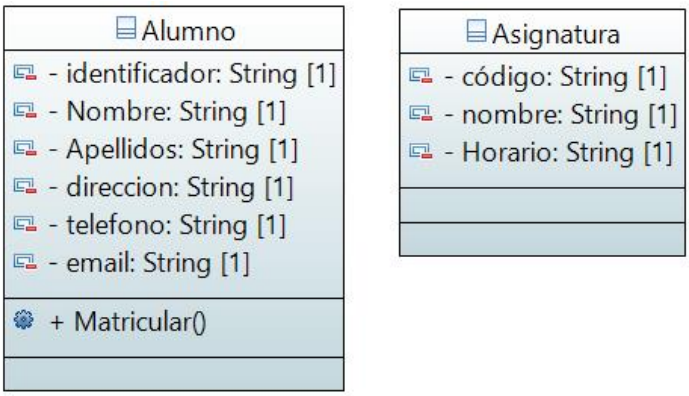

Ilustración 1: Ejemplo de clases UML

Como se observa en la ilustración 1, al modelar estos dos conceptos después no resulta difícil de interpretar. Cada uno es representado por un rectángulo que está dividido en secciones. La primera sección corresponde al nombre de la clase. En la segunda sección se presentan los diferentes atributos que posee el dicho concepto junto con su tipo de dato (En UML los tipos básicos son cuatros: Integer, String, Boolean y Real). En la tercera sección se presentan las operaciones o funciones asociadas a este concepto. En la ilustración se aprecia una cuarta sección y esto se debe que la herramienta utilizada para modelar el ejemplo agrega en esta sección otros elementos que se relacionan con un elemento clase.

Los atributos y operaciones o funciones pueden ser de tres tipos: públicos (es accesible desde cualquier punto), privado (es accesible desde su misma clase) y protegido (es accesible desde su misma clase y por las clases que son herencia de ella).

En el ejemplo se puede apreciar que la clase alumno, presenta los siguientes atributos: identificador, nombre, apellidos, dirección, teléfono y email, todos ellos de tipo string y multiplicidad 1. Además, esta clase presenta una función llamada matricular (). Por otro lado, la clase asignatura, presenta los siguientes atributos, código, nombre y horario, todos de tipo string.

Asociación

UML permite asociar dos entidades representadas como clase. Una asociación se representa como una línea donde los extremos están unidos a las respectivas clases. En los extremos de la asociación se definen la multiplicidad que indica el número de instancias de esa clase que puedan participar en la asociación. En ciertas ocasiones la asociación recibe un nombre, entregando más información sobre la descripción del contexto que describe el diagrama de clase. (Ver ilustración 2).

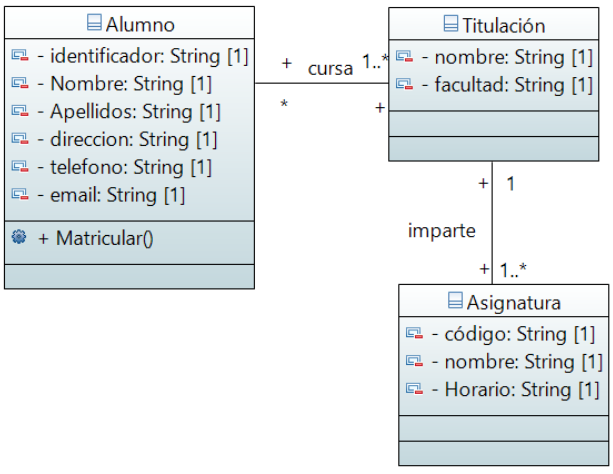

Ilustración 2: Ejemplo de Asociaciones UML.

En la ilustración 2, observamos dos relaciones. La primera, cursa relaciona a la clase alumno con titulación e indica que un alumno puede cursar una o más titulación y titulación puede tener cualquier número de alumnos inscritos. La segunda, imparte relaciona titulación con asignatura y se refiere que una titulación puede impartir uno o más asignatura, mientras que una asignatura puede pertenecer a una titulación.

Agregación

La agregación es un tipo de relación dinámica, donde el tiempo de vida del objeto incluido es independiente del que lo incluye (el objeto base utiliza el incluido para su funcionamiento). Ver ilustración 3.

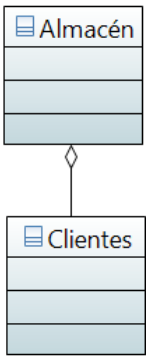

Ilustración 3: Ejemplo agregación UML.

En la ilustración 3 observamos dos clases (almacén y clientes) que se encuentran en asociación de agregación. La agregación se representa como un rombo transparente, y se ubican en el objeto que posee la referencia. En el caso que se destruye el objeto almacén no se ve afectado el objeto clientes.

Composición

La composición es un tipo de relación estática, donde el tiempo de vida del objeto incluido está condicionado por el tiempo de vida del que lo incluye (el objeto base se construye a partir del objeto incluido, es decir, “parte/todo”). Ver ilustración 4.

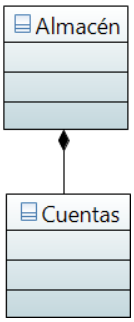

Ilustración 4: Ejemplo de composición UML.

En la ilustración 4 observamos dos clases (almacén y cuentas) que se encuentran en asociación de composición. La composición se representa como un rombo relleno, y se ubica en el objeto que posee la referencia. En el caso que se destruye el objeto almacén también se ve afectado el objeto cuentas.

Generalización

Indica que una subclase hereda las funciones y atributos especificados por una súper clase, es decir, una subclase además de los atributos y funciones propias, poseerá las características y atributos visibles de la súper clase (público y protegido).

En UML una generalización se representa como una flecha que va desde la subclase hacia la súper clase. Ver ilustración 5.

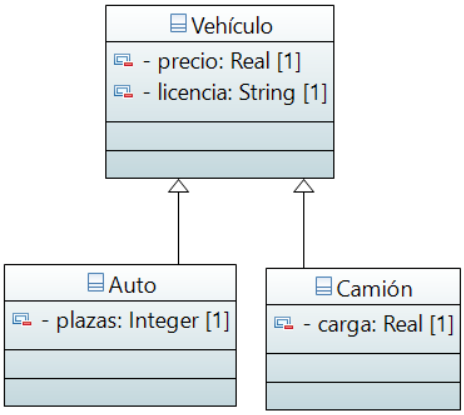

Ilustración 5: Ejemplo de generalización UML.

En la ilustración 5, se observa que la clase Vehículo es la súper clase y la clase auto y camión son las subclases. Como se aprecia ambas subclases representan un vehículo, pero ambos contienen información diferente. En la subclase auto interesa saber la capacidad de pasajeros (plazas) y en la subclase camión la capacidad de carga. Sin embargo, ambas subclases pertenecen al tipo vehículo y comparten información en común como el precio y licencia.

Paquetes

Los paquetes es un elemento que posee UML, que permite agrupar otros elementos de una forma más comprensible. Este tipo no resulta ningún impacto en la semántica del diagrama de clase, simplemente facilita la interpretación.

En UML los paquetes se representa mediante una figura similar a una carpeta. También es permitido importar elementos y paquetes enteros externos. Para representar el hecho señalado se realiza mediante una flecha discontinua que va desde el paquete en cuestión hasta el elemento importado. Ver ilustración 6.

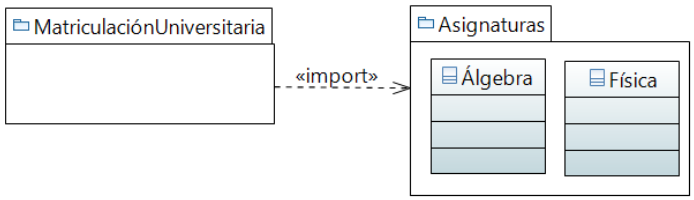

Ilustración 6: Ejemplo de Paquetes UML.

En la ilustración 6, observamos dos paquetes uno que no contiene ningún elemento, pero importa los elementos del segundo paquete el cual contiene dos clases (algebra y física). Nos damos cuenta, de modo de ordenar las distintas asignaturas se asignan dentro del paquete asignaturas y el paquete principal importa dichos elementos.

Enumeración

En UML una enumeración representa un elemento que tiene un número definido de valores. Este número definido de valores se denomina literal de enumeración, que son elementos de modelo en el diagrama de clase. Ver la ilustración 7.

Ilustración 7: Ejemplo de enumeration UML.

En la ilustración 7 se observa un elemento enumeración de nombre continente, y contiene cinco posibles valores (América del sur, América del norte, Europa, Asia y Oceanía).

Estereotipo

En ciertos casos es necesario ampliar los mecanismos que proporciona UML para poder adaptarlo a las necesidades de un dominio en particular que se está modelando. Una de las soluciones que proporciona UML para estos casos, es el uso de estereotipos.

En UML, un estereotipo es una clase que se define con el objetivo de poder extender las características que posee un elemento perteneciente a un modelo determinado. Existen casos en donde un estereotipo se utilizan como un elemento diferenciador, para agrupar los elementos de un mismo tipo en distintas categorías.

En las clases que se aplica un estereotipo, este aparece representado encima del nombre entre las cadenas de caracteres “<<” y “>>”. Ver ilustración 8.

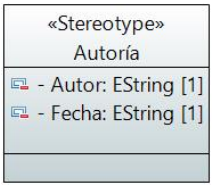

Ilustración 8: Ejemplo Estereotipo UML.

En la ilustración 8 observamos la clase autoría a la cual se aplicó el estereotipo, y esta contiene dos atributos, autor y fecha, ambos de tipo string.

2.2. Estado del Arte

En el estado del arte se realizó una búsqueda sistemática de la literatura, que consiste en un medio de evaluación e interpretación de todos los estudios relevantes para una pregunta de investigación, área temática, o fenómeno de interés.

Para el presente trabajo se buscan herramientas, verificación de modelos, documentos basado en el estándar IEC 61508, metamodelos, lenguaje de modelado y perfiles UML o extensiones.

Para realizar una revisión sistemática de la literatura se define el siguiente proceso (ver ilustración 9).

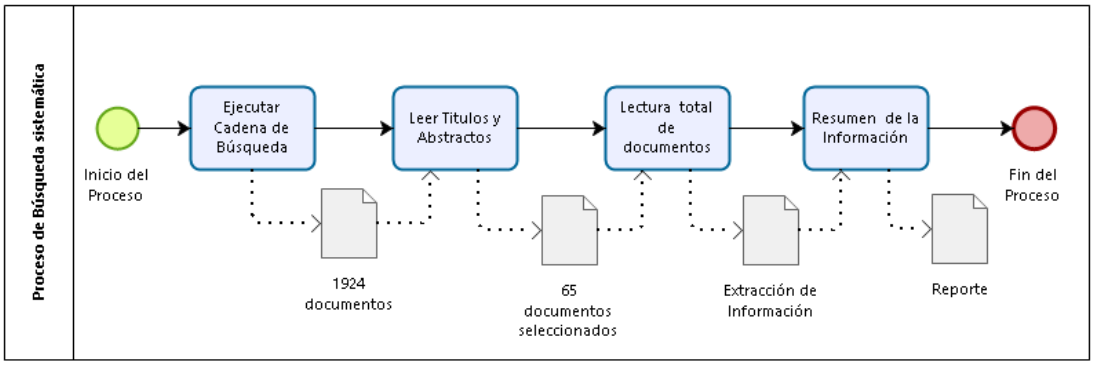

Ilustración 9: Proceso de revisión sistemática. Fuente: Presentación profesor Giovanni Giachetti.

Según lo establecido en el proceso el primer paso es definir una cadena de búsqueda, para ello se utilizará la metodología PICOC, la cual divide la cadena

en 5 términos diferentes. Para el presente trabajo la cadena de búsqueda según la metodología es la siguiente:

- Population: ("UML" OR "Profile") AND
- Intervation: ("Model-Driven" OR "DSML" OR "Metamodel") AND
- Comparison: para este caso no aplica
- Outcome: ("61508" OR "Safety") AND
- Context: ("Critical System")

Según la metodología se debe respetar el orden de los términos, los conceptos de cada término deben ir dentro de comilla doble y separados por el operador lógico OR y encerrados en paréntesis. Para pasar al siguiente término se debe anteponer el operador lógico AND.

Y por último debemos definir los criterios de inclusión y exclusión para la selección de los documentos que identifique la búsqueda.

Criterios de Inclusión

- Definen meta-modelos o modelos de referencia.
- Definen extensiones UML o utilizan perfiles.
- Establecen un mecanismo de verificación para los modelos definidos.
- Cuentan con herramientas propias o usa alguna existente.

Criterios de Exclusión

- Artículos publicados desde el año 2010 en adelante. Dicha fecha es que apareció el estándar de referencia.
- Todo lo que no cumpla con algún criterio de inclusión.
- Artículos o documentos que tengan referencia a Safety o IEC 61508 de forma genérica.

Al ejecutar la cadena de búsqueda, el buscador académico localiza 671 documentos inicialmente.

Segundo paso, es realizar una lectura de títulos y abstractos de los 671 documentos, y según los criterios de exclusión e inclusión, fueron seleccionados 64 documentos.

Tercer paso, es realizar una lectura completa de los 64 documentos seleccionados, y según el análisis realizado se extrae 13 documentos relevantes, que sostienen enfoques que apoyan el desarrollo del presente trabajo.

Finalmente realizar un resumen de los diferentes documentos seleccionados, entregando los enfoques que plantean y definir las ventajas para el presente trabajo y los puntos que aún quedan por definir o desarrollar. Los documentos seleccionados son los siguientes:

1. **Characterizing the Chain of Evidence for Software Safety Cases: A Conceptual Model Based on the IEC 61508 Standard** (Panesar**-**Walawege**,** Sabetzadeh**,** Briand, & Coq, 2010)

El objetivo de este documento es proporcionar un modelo conceptual que caracteriza la evidencia necesaria para argumentar sobre la seguridad de software. El modelo considera los requisitos esenciales para demostrar el cumplimiento de la norma IEC 61508, la trazabilidad y los enlaces necesarios para crear una transición fluida sobre pruebas, llamadas cadenas de pruebas.

El modelo conceptual es formalizado en un diagrama de clases UML. El modelo presente en este documento es de forma reducida, y fue dividido en 10 paquetes que reducen la complejidad del modelo, la especificación técnica está definido en otro documento (Panesar-Walawege et al., 2010).

Los 10 paquetes definidos son los siguientes:

- Conceptos de sistemas: describe los elementos básicos necesarios para conceptualizar los sistemas de mando relativos a la seguridad que involucran tanto hardware y software.

- Conceptos de peligro: captura los peligros y los riesgos que se presentan, que constituyen motivos de requisitos de seguridad y los niveles de integridad de seguridad.
- Conceptos de requerimientos: Conceptos necesarios para describir los requisitos para la creación, funcionamiento, mantenimiento y desmantelamiento de un sistema.
- Conceptos de procesos: Desarrollo de software para un PES que sigue un determinado proceso.
- Conceptos de Artefactos: caracteriza las entradas y salidas de las actividades de desarrollo.
- Emitir conceptos: Son modelos los conceptos que permitan la descripción de problemas.
- Conceptos de gestión de configuración: Conceptos necesarios para la gestión del cambio y para asegurar que los requisitos de seguridad siguen siendo satisfecho cómo evoluciona el sistema.
- Conceptos de justificación: Desarrollo del sistema entraña diversas decisiones que necesitan ser justificados por el razonamiento y basada en suposiciones sobre el dominio y los artefactos.
- Conceptos de orientación: Por ejemplo, un sector o un estándar específico de práctica recomendada podrá encargar ciertos requisitos que deben ser cumplidos por las empresas públicas; o la ejecución de código fuente puede esperarse que se basa en un cierto estándar de codificación.
- Conceptos de dominio específico: tipos de enumeración que pueden personalizarse mediante la definición de los valores de enumeración específica para un determinado contexto.

Además, en el documento se define un modelo conceptual de especialización, donde el estándar IEC 61508 puede ser definido en un dominio en específico, por ejemplo, automóvil, avión, tren, entre otros.

Para el presente trabajo de tesis, este documento nos define las características necesarias para cumplir con el estándar safety, modelados en un diagrama de clases UML.

2. **A Model-Driven Engineering Approach to Support the Verification of Compliance to Safety Standards** (Panesar-Walawege, Sabetzadeh, & Briand, 2011a)

En este documento basándose en los mismos principios subyacentes MDE de un trabajo previo, desarrolla un nuevo enfoque para ayudar a los diseñadores de sistemas a relacionar los conceptos de un dominio de aplicación a las pruebas de los requisitos de las normas que se aplican al dominio.

Más precisamente, comienza con el desarrollo de un perfil basado en el modelo conceptual de una determinada norma. El perfil es argumentado con limitaciones verificables que el sistema ayuda a los proveedores a relacionar sistemáticamente los conceptos del estándar con la aplicación de un dominio especifico. La relación resultante proporciona un camino claro para el proveedor para demostrar cómo el desarrollo de artefactos puede utilizarse para demostrar el cumplimiento de la norma.

Utiliza perfiles UML para comprobar automáticamente las limitaciones que deben poseer para el cumplimiento del estándar de seguridad.

3. **Model-Driven Development with MECHATRONIC UML** (Schäfer & Wehrheim, 2010)

En este documento se define el modelado de un perfil UML para sistemas mecatrónicas (componentes mecánicos y eléctricos relacionados con software).

4. **Using UML Profile and OCL to Impose Regulatory Requirements on Safety-Critical System** (Lin, Shen, & Kountanis, 2013)

En este documento se propone el desarrollo de un perfil, con el objetivo de cumplir con todos los requerimientos regulatorios (FDA) para el caso específico de la insulina.

En conjunto con el lenguaje modelado UML y restricciones OCL definen un perfil UML.

5. **A profile and tool for modelling safety information with design information in SysML** (Biggs, Sakamoto, & Kotoku, 2016)

En este documento genera un perfil llamado SafeML, que permite integrar información de seguridad con el diseño de sistema de información, como una ayuda para la coherencia de la información y la comunicación entre los equipos de desarrollo y entre los miembros de un equipo.

6. **Quantitative Analysis of UML Models** (Leitner-Fischer & Leue, 2011)

En este documento genera una extensión UML, con el fin de tener toda la información necesaria para el análisis de desempeño. En dicha extensión o perfil define los estereotipos y propiedades que se utilizan para definir análisis estocástico. (Propiedades CSL).

7. **Using UML Profiles for Sector-Specific Tailoring of Safety Evidence Information** (Panesar-Walawege, Sabetzadeh, & Briand, 2011b)

En este documento genera perfiles UML, para capturar la relación entre los requisitos de un estándar genérico con un estándar específico.

Además, se propone una metodología para especializar un estándar genérico.

8. **UML Automation Profile: Enhancing the Efficiency of Software Development in the Automation Industry** (Ritala & Kuikka, 2007)

En este documento genera perfiles UML para modelar aplicaciones de automatización, para el desarrollo de software en la industria del automóvil.

9. **Using UML as a Domain-Specific Modeling Language: A Proposal for Automatic Generation of UML Profiles** (Giachetti, Marín, & Pastor, 2009)

En este documento presenta un proceso que integra un DSML en UML mediante la generación automática de un perfil UML, este proceso facilita el correcto uso de UML en un contexto MDD y proporciona una solución para aprovechar las ventajas de UML y DSML.

Este documento presenta una solución completamente automatizada para una generación de perfil UML utilizando como entrada el metamodelo DSML relacionados con un enfoque MDD. Por lo tanto, este documento muestra cómo las extensiones de UML requeridas pueden ser identificadas automáticamente y detalla las reglas de transformación para obtener el perfil de UML que implementa estas extensiones.

El proceso definido se muestra en la siguiente imagen:

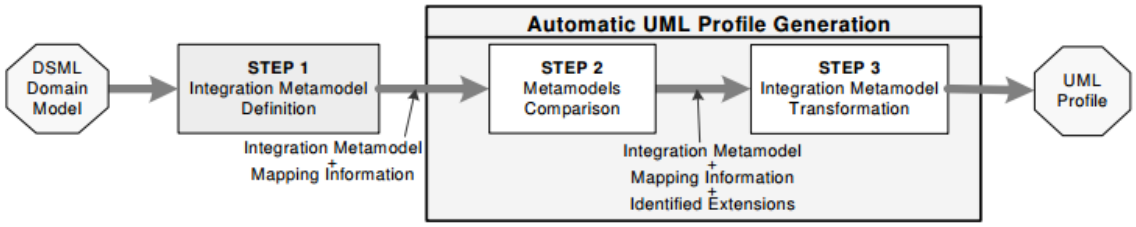

Ilustración 10: Esquema del proceso de Integración. Fuente (Giachetti et al., 2009).

10. **Supporting the verification of compliance to safety standards via model-driven engineering: Approach, tool-support and empirical validation** (Panesar-Walawege, Sabetzadeh, & Briand, 2013)

Este documento propone un nuevo enfoque para ayudar a los proveedores en la creación de las pruebas necesarias para obtener la certificación conforme a los estándares. El enfoque se basa en Model-Driven Engineering (MDE) y aborda los desafíos de utilizar estándares de certificación al proporcionar asistencia con el cumplimiento.

El método utilizado es el siguiente: Dado un estándar de seguridad, se construyó un modelo conceptual que proporciona una sucinta y explícita interpretación del estándar. Este modelo se utiliza para crear un perfil UML que ayuda a proveedores de sistemas en relacionar los conceptos del estándar de seguridad para los dominios de aplicación, lo cual permite a su vez los proveedores para demostrar cómo sus artefactos de desarrollo del sistema cumplen con la norma.

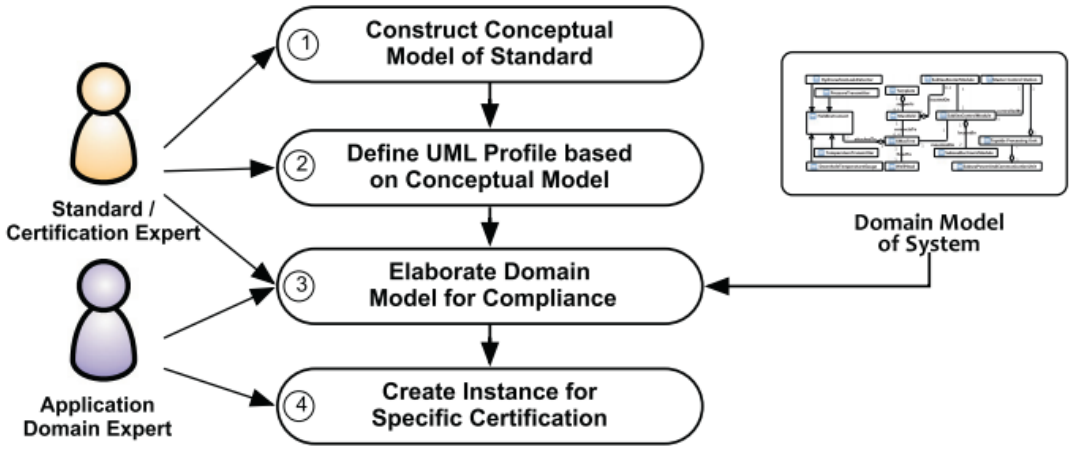

Ilustración 11: Metodología para la creación de pruebas de un estándar de seguridad. Fuente (Panesar-Walawege et al., 2013).

11. **Model-based specification of safety compliance needs for critical systems: A holistic generic metamodel** (de la Vara et al., 2016)

Este documento proporciona un metamodelo para la especificación de las necesidades de cumplimiento de los estándares de seguridad para los sistemas críticos.

El metamodelo es holístico y genérico, y resume conceptos comunes para demostrar el cumplimiento de la seguridad en diferentes niveles y dominios de aplicación.

El metamodelo se denomina “Reference Assurance Frameworks” (RAF), que modela los distintos criterios para demostrar la conformidad de un sistema crítico con un estándar de seguridad. El metamodelo incluye conceptos y relaciones en

forma de clases y asociaciones que son comunes a distintas normas de seguridad y a los distintos dominios de aplicación. El cumplimiento de las normas de seguridad se aborda desde varias perspectivas, explícitamente relacionadas con información relacionada con el proceso, datos y objetivos necesarios para demostrar el cumplimiento, y su aplicabilidad.

Finalmente, este documento extiende los resultados de un documento previo, donde se presenta una versión inicial del metamodelo.

12. **A UML Profile for the Development of IEC 61508 Compliant Embedded Software** (Kuschnerus, Bruns, Bilgic, & Musch, 2012)

Este documento propone un perfil UML que extiende el lenguaje unificado de modelado (UML) para apoyar el desarrollo de software integrado crítico en seguridad en conformidad con la norma IEC 61508.

El perfil diseñado proporciona una base para nuevas técnicas de modelización. Su objetivo es ofrecer mejoras significativas para el desarrollo de software integrado críticos para la seguridad en el uso del plan UML y otros enfoques.

La metodología propuesta es establecer en un primer paso un modelo de dominio, el cual contiene los elementos básicos del dominio en virtud de la investigación y las relaciones de esos elementos. Además, las limitaciones en cuanto a las relaciones y los elementos pueden incluirse en el modelo. La principal ventaja de este enfoque es que separa el proceso de recopilación de información de un dominio y el proceso de creación de un perfil de UML a partir de esta información.

El segundo paso es definir el perfil UML, el cual tiene como entrada el modelo de dominio definido en el párrafo anterior. El perfil UML se define a partir de las diferencias existentes del modelo de dominio y el metamodelo UML, estas diferencias son definidas como estereotipos en el perfil UML.

13. **Modeling safety and airworthiness (RTCA DO-178B) information: conceptual model and UML profile** (Zoughbi, Briand, & Labiche, 2011)

Este documento presenta un enfoque para mejorar la comunicación y la colaboración entre los ingenieros de seguridad, ingenieros de software y autoridades de certificación en el contexto del estándar RTCA DO-178B (Estándar para el control de la seguridad en sistemas aeroespaciales).

Se utiliza un perfil UML que permite a los ingenieros de software modelar los conceptos relacionados con la seguridad y propiedades en UML, del estándar definido. Un meta-modelo conceptual se define según RTCA DO-178B, y luego el correspondiente perfil de UML, llamado SafeUML, está diseñado para permitir su modelización precisa.

Capítulo 3: Metodología y Planificación

3.1. Enfoque Metodológico

Como metodología para desarrollar este trabajo de tesis se utilizará "*Action Research*", que consiste en el principio de identificar un problema, recolectar información en relación al problema para tener mayor detalle y plantear una o más soluciones potencialmente posibles, una vez seleccionada una solución se debe plantear las acciones a seguir y se implementa, evaluar los datos obtenidos de los resultados. En este punto, el problema es re-valuado y el proceso comienza un nuevo ciclo. El proceso continúa hasta que se resuelva el problema.

En la metodología se identifican las siguientes 5 fases:

1. Diagnóstico: Definir o Identificar un problema a resolver.
2. Acción: Considerar alternativas posibles de acción.
3. Acción a Tomar: Seleccionar una alternativa de acción.
4. Evaluación: Estudiar o evaluar los resultados de una acción.
5. Especificar el aprendizaje: Aprender de los resultados para seguir en una nueva iteración.

Action Research es utilizado en acciones reales en vez de situaciones creadas o estudios experimentales, desde su enfoque principal se centra en la solución de problemas reales.

La metodología *Action Research* será abordado en el presente trabajo de la siguiente manera.

Diagnóstico

El primer paso de la metodología es definir o plantear un problema, del cual se requiere obtener una solución satisfactoria. Para el presente trabajo definimos como problema el objetivo general definido en secciones anteriores, “Garantizar la completitud y la consistencia de un perfil UML generado automáticamente a partir de un metamodelo de Safety basado en el estándar IEC 61508”. Como motivo de abordar de mejor manera el problema en base al objetivo general, dividimos en sub problemas que nos permite entregar apoyo al problema general, los cuales son:

1. Verificar el Metamodelo MOF de entrada que contenga las características necesarias de un metamodelo safety en base al estándar IEC 61508.
2. Identificar diferencias entre un metamodelo MOF de entrada y un metamodelo UML.
3. Generar correctamente un perfil UML a partir de un metamodelo MOF de Safety.

Acción

El siguiente paso es diseñar una o varias soluciones posibles en base a una investigación en diversas fuentes que ayuden afrontar los problemas planteados en el paso anterior.

Para el presente trabajo, se realizó una búsqueda sistemática en el ámbito académico, utilizando un buscador confiable para obtener información de

documentos, papers, conferencias, libros, entre otros, que definan lineamientos para tener conocimiento de cómo afrontar los problemas planteados anteriormente.

Una vez finalizada la investigación sobre posibles elementos o herramientas que ayuden a definir una solución a cada problema y con ello dar solución al problema general, se puede identificar varias propuestas de diferentes autores que nos permiten resolver los problemas, y son las siguientes:

1. Definir un perfil UML que contiene 10 características necesarias que identifican un Metamodelo MOF alineado al estándar IEC 61508.
2. Definir un modelo de comparación entre un metamodelo MOF y metamodelo UML.
3. Definir un modelo de transformación del tipo *modelo a modelo* que permitirá generar de forma automática un perfil UML, a partir de un metamodelo MOF.

Todos los puntos estarán apoyados por documentos que definen la generación de perfiles, validación de perfiles, transformación de modelos, y comparación de metamodelos.

Acción a Tomar

Luego de haber definido una propuesta de solución para cada problema del presente trabajo, se entrega una planificación de cómo afrontar el desarrollo de las diferentes soluciones.

Teniendo en cuenta el problema general de acuerdo al objetivo general planteado en secciones anteriores, los pasos serán los siguientes:

1. Definiremos un caso de estudio, que será el contexto en que está confeccionado el Metamodelo MOF de entrada (Avión o automóvil).

2. Desarrollar un perfil UML que represente al Estándar internacional IEC 61508.
3. Comparar el Metamodelo MOF de entrada con un Metamodelo UML, si existen elementos que no estén contenidos en UML, surge la necesidad de generar un perfil UML.
4. Realizar una transformación de la diferencia de elementos identificados en el paso anterior y generar un perfil UML con el objetivo de dar completitud al problema general.

Evaluación

El Metamodelo MOF de entrada modelado no será evaluado su confección, la evaluación en el paso 1 es considerar si cumple con las características necesarias que identifican que está alineado al estándar internacional, y esto se logra comprándolo con el perfil UML modelado en base al estándar. Si el Metamodelo MOF cumple con todas las características, se pasa al siguiente paso de lo contrario se debe realizar un nuevo ciclo de mejora. En el caso de que al Metamodelo MOF de entrada no cumpla con al menos una característica que debe ser modelada, se considera la solución propuesta como no satisfactoria, por lo que se documenta e incluye todo lo aprendido, se pasa al siguiente paso y se reinicia el ciclo en búsqueda de una solución satisfactoria. Para las soluciones a los problemas siguientes al problema 1 se considera el mismo esquema, evaluando si los resultados obtenidos con la acción de la solución propuesta al problema son satisfactorios o no, lo cual determinará si se repite el ciclo de diseño de una nueva propuesta de solución con el aprendizaje obtenido o si se sigue al siguiente paso con las siguientes soluciones propuestas a los problemas posteriores.

Especificar el aprendizaje

El último paso a considerar en la metodología *Action Search* es determinar el aprendizaje obtenido a partir de la solución planteada y los resultados que fueron obtenidos en el paso anterior.

Evaluar las ventajas y desventajas de la solución en base a un análisis que permita identificar errores producidos, por que la solución fue correcta, se puede mejorar la solución, se puede descartar aspectos de diseño, etc.

En el caso de una solución satisfactoria se documenta todo lo referente a la solución para luego emplearlo en la solución de problemas de una nueva propuesta, es decir, en el caso de los metamodelos iniciales del trabajo se documentan las clases, atributos, relaciones entre clases, tipos de datos, imagen del modelo, etc. Todo lo que pueda usarse por el autor del trabajo para el avance en otros aspectos del trabajo. En cambio, en el caso de poseer una solución no satisfactoria, se documenta lo que si se utilizará para la siguiente solución propuesta y lo que se desechó se documenta solo en el caso de que el cambio desde la primera solución hacia la segunda solución hubiera sido demasiado drástico, es decir, empezar con una nueva solución desde cero, ya que representaría información valiosa sobre lo que se hizo y que no se debe volver a hacer, en cambio en el otro caso representan “mejoras” a lo ya realizado.

3.2. Planificación del Proyecto

Definición de Tareas, Duración y Esfuerzo

En base a lo definido en la metodología Action Research en la sección anterior, definimos las principales tareas a ser desarrolladas en este trabajo, como muestra la siguiente tabla.

N°	Tarea	Duración	Esfuerzo
1	Definir Metamodelo MOF de entrada en un caso particular de estudio (Automóvil o Avión)	1 mes	4 HH al día
2	Definir un perfil UML alineado al estándar internacional (IEC 61508)	1 mes	4 HH al día
3	Definir un modelo de verificación para el metamodelo MOF	1 mes	4 HH al día
4	Desarrollar un modelo de comparación entre Metamodelo MOF y Metamodelo UML	2 mes	4 HH al día
5	Desarrollar un modelo de transformación en base a reglas de la forma Model-to-Model	1 mes	4 HH al día
6	Definir pruebas y analizar los resultados obtenidos	0,5 mes	3 HH al día
	Total, Horas Hombre		765 HH

Tabla 2: Tareas Principales del Proyecto.

Para la duración y esfuerzo definidos en la tabla anterior, es una estimación ajustada a la realidad, lo que no quiere decir que no puede sufrir modificaciones y va depender de la evolución de las iteraciones en base a la metodología (Action Research), lo que significa mientras más iteraciones la duración y esfuerzo sufrirán cambios.

Hitos

Para el presente trabajo de tesis, se definirán 4 fases como hitos, donde estarán contenidas las tareas anteriormente mencionadas.

Fase 1 – Definición de Metamodelo MOF

La primera fase del presente proyecto corresponde a definir el metamodelo MOF, lo que significa definir el dominio o caso de estudio que será definido el Metamodelo y verificar el alineamiento del Metamodelo con el estándar internacional IEC 61508.

Sus principales tareas son:

1. Definir un dominio.
2. Seleccionar 2 Metamodelos MOF en base al dominio seleccionado.
3. Modelar un Perfil UML que contenga 10 características necesarias para un modelo safety.
4. Verificar el Metamodelo MOF de entrada.

Fase 2 – Comparar Metamodelos

La segunda fase del presente proyecto de tesis, corresponde a comparar el Metamodelo MOF con el Metamodelo UML, permitiendo identificar si los elementos de un metamodelo están contenidos en el otro, al existir una diferencia es necesario modelar un perfil UML.

Las principales tareas son:

1. Desarrollar un modelo de comparación entre Metamodelos.
2. Comparar Metamodelo MOF con el Metamodelo UML, los siguientes elementos: clases, propiedades (atributos y asociaciones), enumeraciones, enumeración literal y tipo de dato.
3. Identificar las diferencias existentes para ser transformado en un Perfil UML.

Fase 3 – Transformar Metamodelo MOF

La tercera fase del presente proyecto de tesis, corresponde a desarrollar un modelo de transformación que consiste en el tipo Model-to-Model, que permite transformar los elementos no contenidos en el Metamodelo UML en un perfil UML.

Las principales tareas son:

1. Desarrollar un modelo de transformación en base a reglas ATL.
2. Transformar los elementos no contenidos en el Metamodelo UML en un perfil UML.

Fase 4 – Pruebas y Documentación

La fase 4 y ultima del presente proyecto de tesis, corresponde donde se verificará que el perfil UML está correctamente generado, cumpliendo en completitud con el dominio utilizado en el caso de estudio y además este de acuerdo al dominio del caso de estudio utilizado. Luego documentar los resultados y finalizar con el análisis de dichos resultados y las conclusiones del trabajo de tesis.

Las principales tareas son:

1. Realizar pruebas con un Metamodelo MOF en un dominio en particular.
2. Analizar los resultados obtenidos.

Capítulo 4: Desarrollo de la Solución

4.1. Introducción a Modelización Conceptual

Para entender los conceptos utilizados en la sección 4.2, que consisten en el desarrollo de la herramienta que genera de forma automática el perfil UML, es necesario conocer una serie de aspectos relacionados con la modelización conceptual.

La modelización conceptual en el ámbito de ingeniería de software, consiste en especificar y describir el conocimiento general que un sistema de información necesita conocer para poder realizar sus funciones correctamente.

Los modelos conceptuales, son un tipo de modelos relativamente poco sofisticados y por lo tanto más fácil de entender. Sin embargo, los modelos conceptuales se han utilizado con distintos propósitos, por consiguiente, distintos colectivos los interpretan de modo distinto.

El principal objetivo de la modelización conceptual es definir un esquema conceptual de un sistema concreto, con el fin de tener el conocimiento necesario de las funciones para ejecutarlas con éxito.

Los modelos conceptuales contienen los conceptos relevantes para el sistema de información diseñado, denominado entidades, además de las relaciones que se establecen entre las entidades, denominada asociaciones.

Los lenguajes que permiten definir modelos conceptuales se conocen con el nombre de lenguaje de modelización conceptual. En el presente proyecto el lenguaje que se utilizara es Unified Modeling Languaje (UML). En el siguiente capítulo se describe los aspectos más relevantes del lenguaje UML.

4.2. Desarrollo del Proyecto

En el desarrollo del presente proyecto distinguimos diferentes pasos, en los cuales se realizó un proceso de ingeniería para generar el metamodelo MOF de entrada de la transformación y la posterior generación automática del perfil UML que representa al estándar safety. Los pasos son los siguientes.

1. Definición del metamodelo MOF que representa el estándar Safety.
2. Generar Perfil para determinar la equivalencia de las clases.
3. Definir reglas ATL para generación automática del perfil UML.
4. Evaluación del Correcto Funcionamiento de las Reglas ATL.

4.2.1. Definición del Metamodelo MOF que Representa el Estándar Safety

El primer paso del proceso corresponde a definir el metamodelo safety con los conceptos propios del dominio asociado al sistema que se desea representar. Para la definición de este metamodelo consideramos el estándar MOF ya que presenta dos beneficios:

1. Se puede utilizar un editor de modelos de clases UML.
2. Su definición permite la generación de editores específicos para los modelos del dominio utilizando herramientas como Eclipse Modeling Framework (EMF). El resultado de esta actividad corresponde al Metamodelo MOF de Safety.

Para ejemplificar este paso, nos apoyamos en lo descrito en (Panesar-Walawege et al., 2010), donde se definieron 10 paquetes diferentes que contienen los elementos esenciales de un sistema o dispositivo necesarios para obtener una certificación de seguridad funcional en base al estándar IEC 61508 (Safety).

Estos 10 paquetes son modelados a través de un diagrama de clase UML, representando los elementos como clases, las relaciones entre las clases como asociaciones y generalizaciones, además integrando las propiedades de las clases como atributos.

Para definir de una manera gráfica el diagrama de clases utilizamos Papyrus, un “plugin” para añadir propiedades a los modelos en el marco de trabajo de Eclipse, generándose un diagrama de clases completo del modelo conceptual.

Los 10 paquetes representados gráficamente son los siguientes. No existe una descripción de cada paquete, ya que fue realizada en la sección de estado del arte.

Paquete Conceptos de Sistema

Este paquete cuenta con 8 clases, 9 asociaciones y 5 generalizaciones. Además, dos clases cuentan con atributos que son de tipo *Enumeration BlockType* y *Enumeration softwareLevel.* Para obtener detalle de los conceptos puede revisarlos en (Lionel Briand, 2009). Ver la ilustración 12.

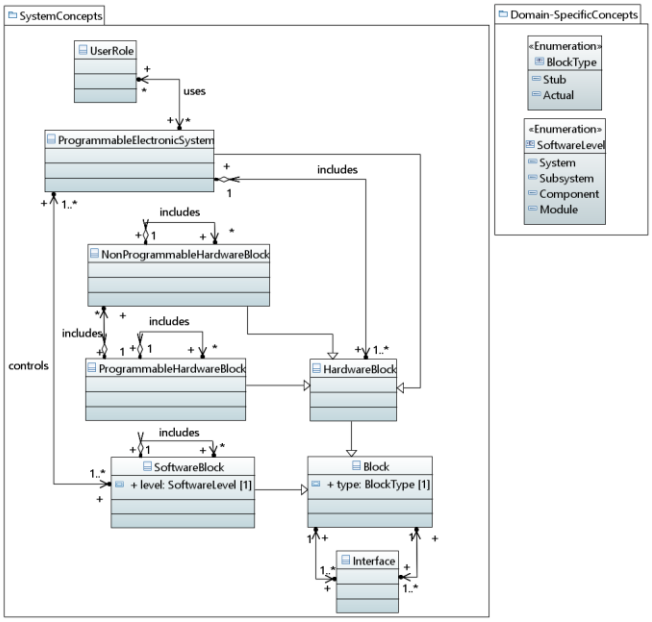

Ilustración 12: Paquete conceptos de sistema.

Paquete Conceptos de Riesgo

Este paquete está compuesto por 5 clases y 4 asociaciones. Además, la clase *Risk* posee 4 atributos de tipo *Risk*. Para obtener detalle de los conceptos puede revisarlos en (Lionel Briand, 2009). Ver la ilustración 13.

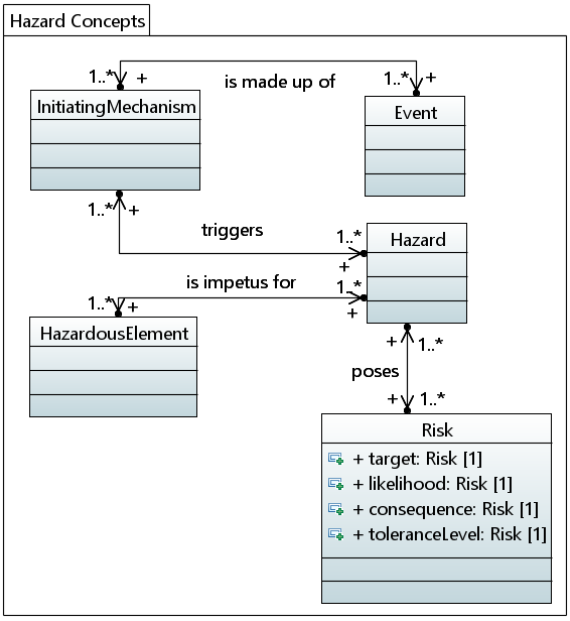

Ilustración 13: Paquete conceptos de riesgo.

Paquete Conceptos de Requerimientos

Este paquete está compuesto por 9 clases, 6 asociaciones y 3 herencias. Además, la clase *RequirementLink* tiene un atributo de tipo *Enumeration RequirementLinkType*. La clase *Requirement* tiene un atributo de tipo *Enumeración RequirementType*. La clase *OperatingMode* tiene un atributo de tipo *Enumeration OperatingModeType*. Y por último la clase *SafetyIntegrityLevel* cuenta con dos atributos, donde el tipo es de referencia a la misma clase. Para obtener detalle de los conceptos puede revisarlos en (Lionel Briand, 2009). Ver la ilustración 14.

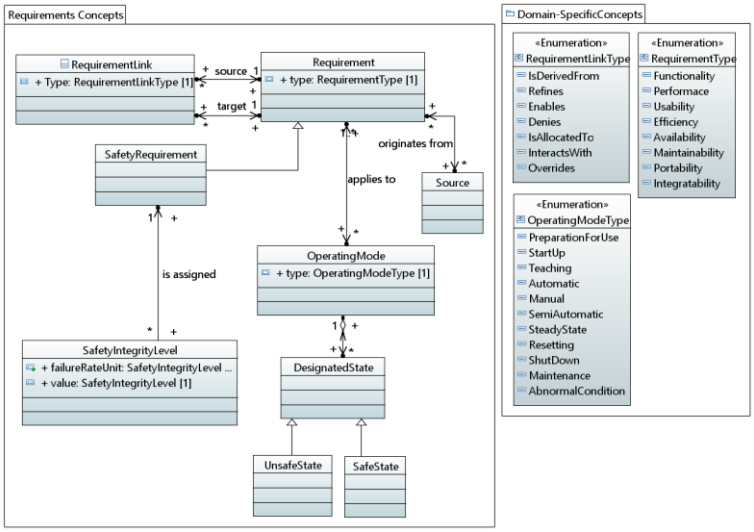

Ilustración 14: Paquete Conceptos de Requerimientos.

Paquete Conceptos de Procesos

Este paquete está compuesto por 7 clases, 8 asociaciones y 2 herencias. La clase *Agente* posee un atributo de tipo *Enumeration AgentType*. Para obtener detalle de los conceptos puede revisarlos en (Lionel Briand, 2009). Ver la ilustración 15.

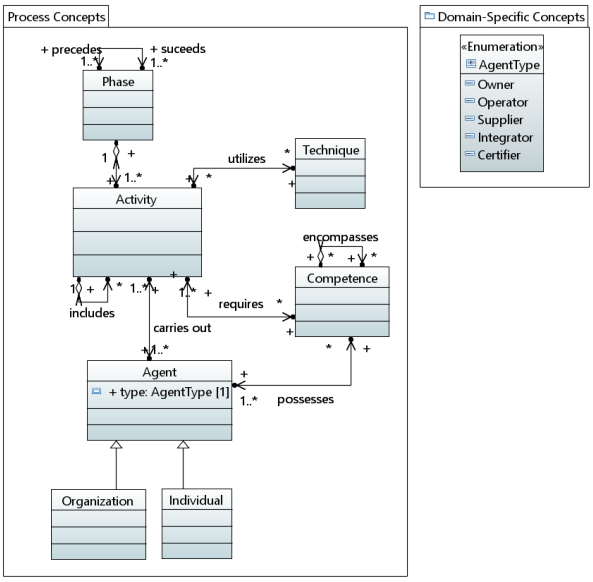

Ilustración 15: Paquete Conceptos de Procesos.

Paquete Conceptos de Artefactos

Este paquete está compuesto por 11 clases, 2 asociaciones y 9 herencias. La clase *Artifact* posee un atributo de tipo *Enumeration ArtifactStateType*, y la clase *ArtifactLink* posee un atributo de tipo *Enumeration ArtifactLinkType*. Para obtener detalle de los conceptos puede revisarlos en (Lionel Briand, 2009). Ver la ilustración 16.

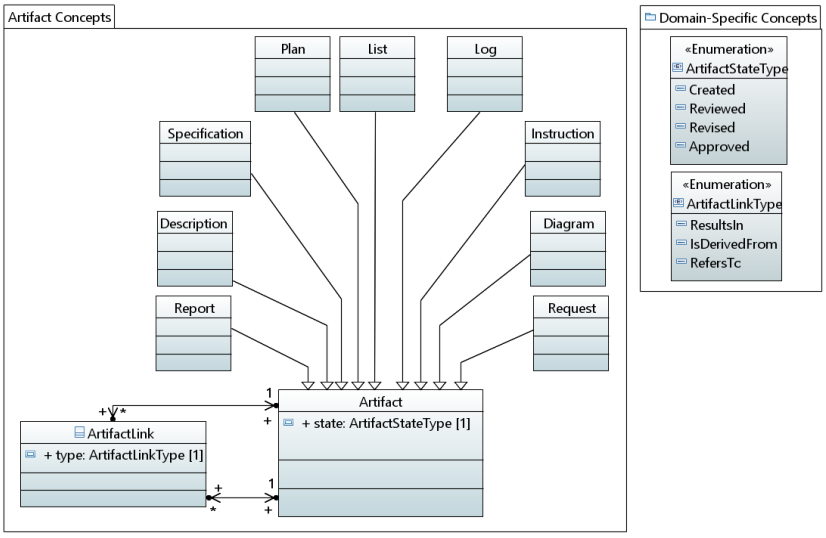

Ilustración 16: Paquete Conceptos de Artefactos.

Paquete Conceptos de Emisión

Este paquete está compuesto por 7 clases, 4 asociaciones y 6 herencias. Para obtener detalle de los conceptos puede revisarlos en (Lionel Briand, 2009). Ver la ilustración 17.

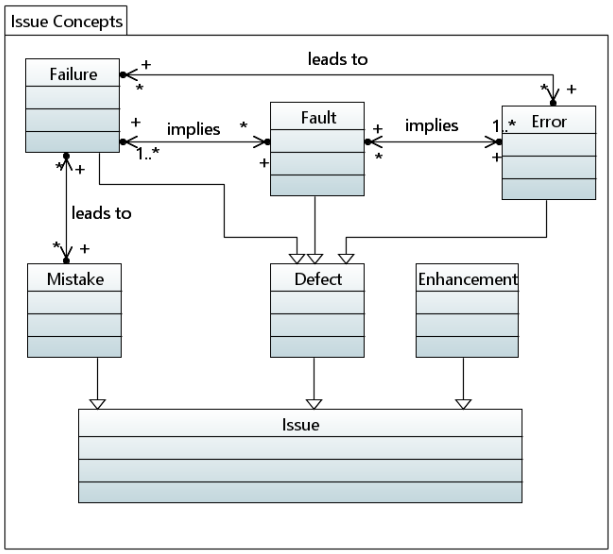

Ilustración 17: Paquete conceptos de emisión.

Paquete Conceptos de Gestión de Configuración

Este paquete está compuesto por 2 clases y 2 asociaciones. La clase *ControlledItem* tiene un atributo de tipo *ControlledItem*. Para obtener detalle de los conceptos puede revisarlos en (Lionel Briand, 2009). Ver la ilustración 18.

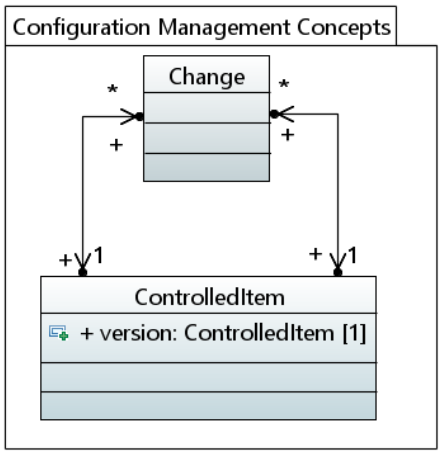

Ilustración 18: Paquete conceptos de gestión de configuración.

Paquete Conceptos de Justificación

Este paquete está compuesto por 2 clases y una asociación. Para obtener detalle de los conceptos puede revisarlos en (Lionel Briand, 2009). Ver la ilustración 19.

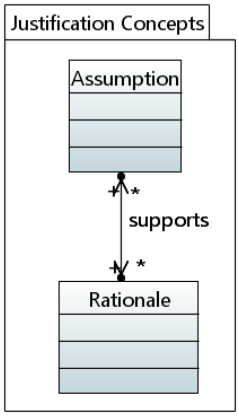

Ilustración 19: Paquete conceptos de justificación.

Paquete Conceptos de orientación

Este paquete está compuesto por 4 clases y 2 herencias. Para obtener detalle de los conceptos puede revisarlos en (Lionel Briand, 2009). Ver la ilustración 20.

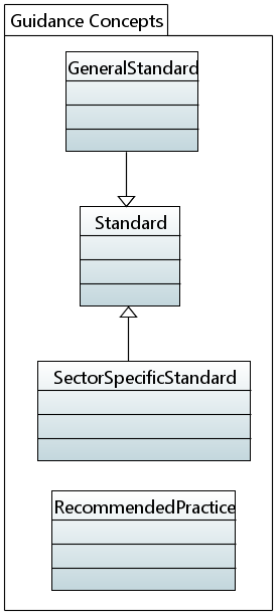

Ilustración 20: Paquete conceptos de orientación.

Paquete Conceptos Dominio Especifico

Este paquete está compuesto por 9 enumeraciones *(Enumeration*), cada una con distinta cantidad de literales, donde en conjunto son 47. Estas enumeraciones son utilizadas por las clases que fueron descritas en los paquetes anteriores. Para obtener detalle de los conceptos puede revisarlos en (Lionel Briand, 2009). Ver la ilustración 21.

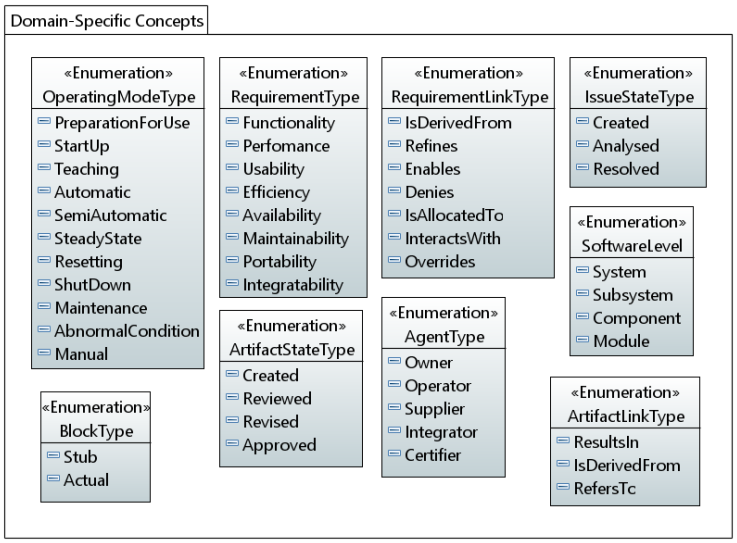

Ilustración 21: Paquete conceptos dominio especifico.

Metamodelo Conceptual Estándar IEC 61508

Una vez definido cada paquete de forma independiente, es necesario presentar el modelo conceptual por completo, donde se observan las relaciones existentes entre los diferentes paquetes. En la ilustración 22, se presenta el metamodelo conceptual.

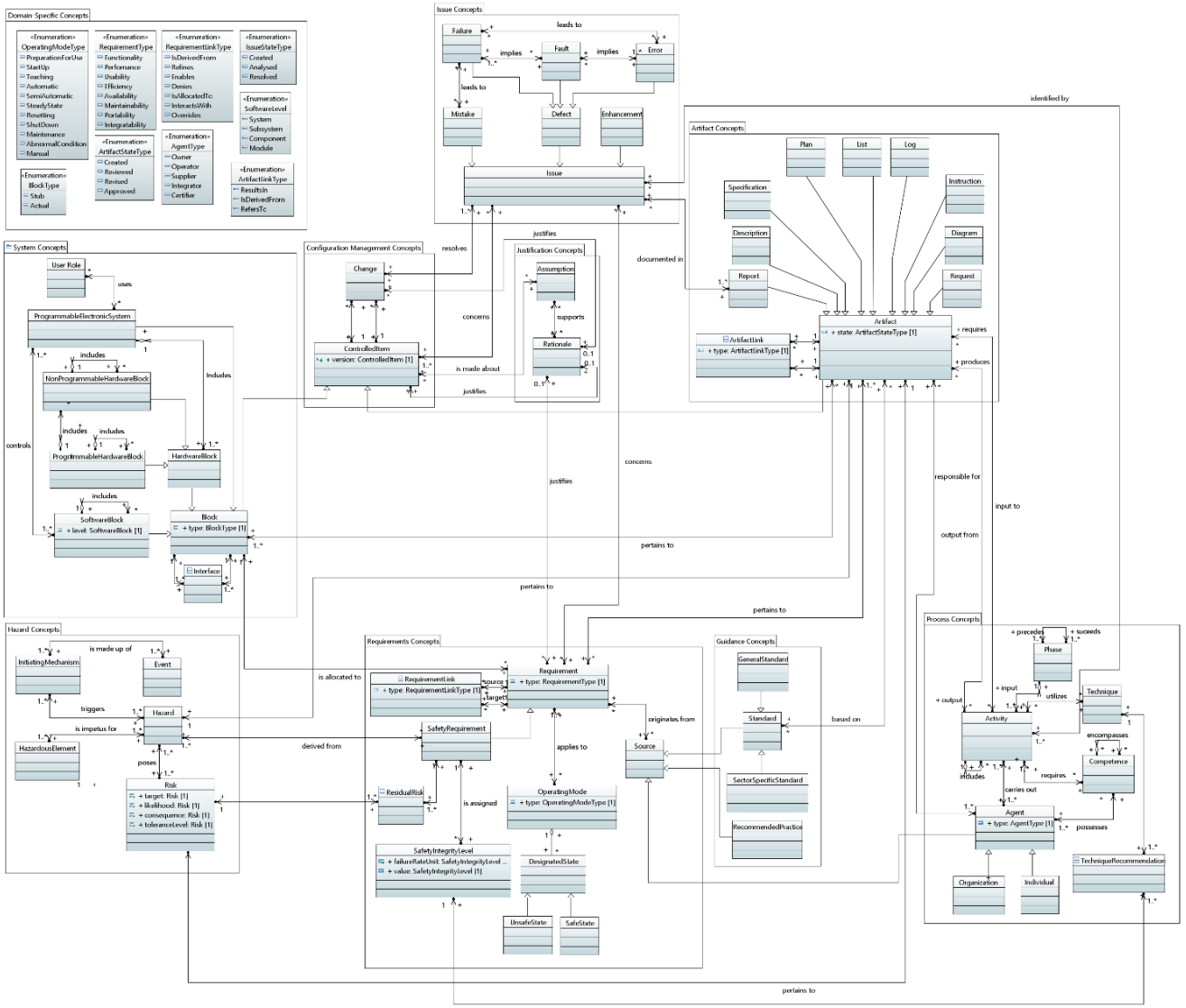

Ilustración 22: Modelo Conceptual Estándar IEC 61508 y sus relaciones.

4.2.2. Generar Perfil para Determinar la Equivalencia de las Clases

El segundo paso, corresponde a establecer el mapeo entre los elementos del metamodelo de Safety de entrada y las metaclases de UML que deben ser extendidas para generar el perfil UML correspondiente. Para realizar este mapeo, se ha propuesto un mecanismo sencillo basado en la generación de un nuevo perfil UML de mapeo. El resultado de esta actividad es el Mapeo Safety–UML.

Si bien existen propuestas de mapeo basadas en el concepto de model-weaving que proponen la generación de un modelo intermedio para almacenar esta información, la complejidad del mapeo para la generación de perfiles orientados a Safety sólo requiere indicar las metaclases que serán extendidas en UML sin necesidad de establecer operaciones complejas entre los constructores de los distintos metamodelos. Además, al usar un perfil UML se puede utilizar el mismo editor UML empleado en la definición del metamodelo de dominio, sin necesidad de crear editores específicos para este fin que aumentan la complejidad en la implementación y aplicación de la propuesta.

La ilustración 23, muestra el perfil definido para establecer el mapeo entre el metamodelo Safety y el metamodelo UML. El perfil denominado *IMProfile* se ha definido a partir de una propuesta orientada a integrar modelos específicos de dominio en UML. Este perfil cuenta con un estereotipo (*Stereotype*) denominado *MappingInfo* que contiene dos propiedades (*Property*) de tipo *UML Element* con el fin de indicar el elemento de origen en el metamodelo Safety (propiedad *base_Element*), y el elemento UML destino (propiedad *TargetElement*).

Además, se define una extensión (Extension) permitiendo definir el vínculo entre el metamodelo UML y las clases presentes en el metamodelo Safety.

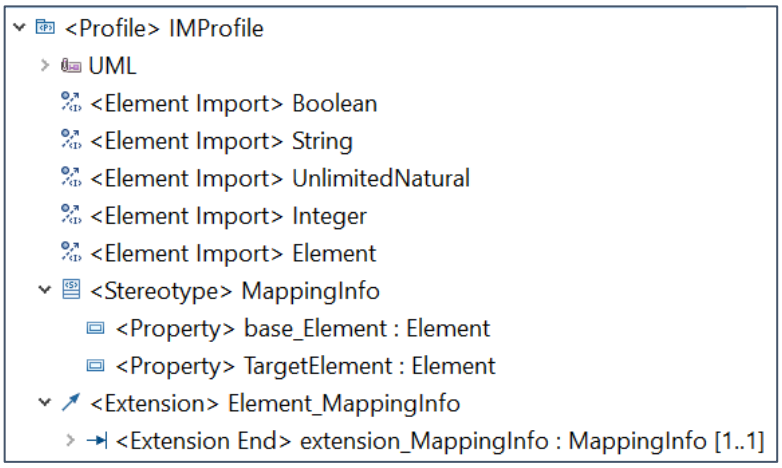

Ilustración 23: Perfil UML IMProfile.

Para ejemplificar este paso, utilizaremos un fragmento del metamodelo Safety descrito en la sección anterior. Seleccionamos el paquete conceptos de sistema con sus respectivas relaciones con los demás paquetes.

Aplicando el perfil *IMProfile* al fragmento del metamodelo Safety seleccionado, se obtiene el resultado presentado en la ilustración 24. Este mapeo establece las equivalencias entre el metamodelo Safety y el metamodelo de clases UML. En particular, en la ilustración 24 se observa que la metaclase *Phase* se ha mapeado con la metaclase *Class* de UML. Para el metamodelo Safety utilizado de ejemplo, todas las clases se han mapeado con la metaclase *Class* de UML, mientras que los atributos, asociaciones, y enumeraciones no se han mapeado, ya que no tiene equivalencia con alguna propiedad o enumeración del metamodelo de UML.

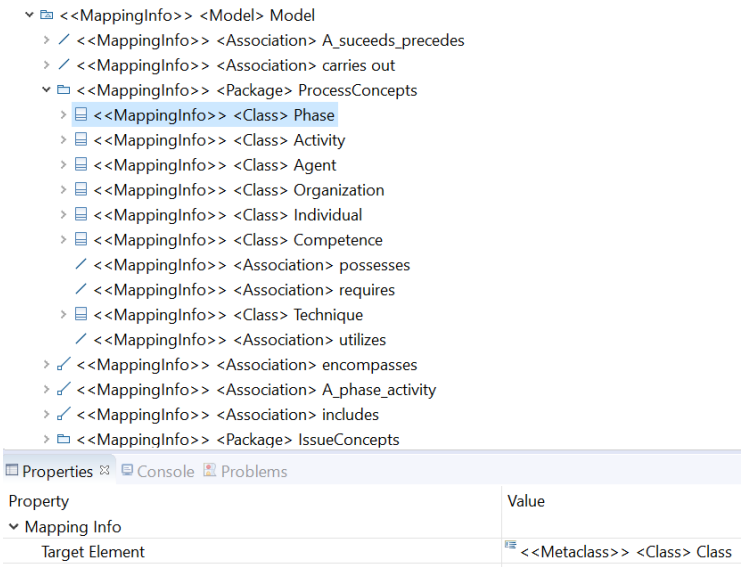

Ilustración 24: Fragmento metamodelo Safety después de aplicar el perfil IMProfile.

4.2.3. Definición de Reglas ATL para la generación Automática de Perfil UML de Safety

El tercer paso establece un conjunto de reglas de transformación para generar el perfil UML de Safety a partir del metamodelo de Safety de entrada y el mapeo con el metamodelo de UML. Este último paso se realiza de forma automática mediante la comparación de los metamodelos a partir de la información de mapeo para identificar las extensiones que deben ser implementadas en el perfil UML. Esta comparación permite determinar las metaclases de UML que serán extendidas y las diferencias entre las propiedades de ambos metamodelos. Son precisamente estas diferencias la base para definir los valores etiquetados que se incorporarán en cada estereotipo del perfil UML para representar las propiedades presentes en el metamodelo de Safety. Los elementos considerados en la comparación entre metamodelos que pueden ser incorporados como extensiones en el perfil UML son atributos, asociaciones, generalizaciones, enumeraciones, valores literales, y tipos de datos. Las clases no pueden ser nuevos elementos, ya que los perfiles sólo pueden generar extensiones a partir de las metaclases UML existentes (no se pueden agregar nuevas metaclases).

Por lo tanto, para garantizar la completitud del perfil generado, todas las clases del metamodelo de safety deben estar mapeadas con una clase del metamodelo de UML. Es importante destacar que las asociaciones del metamodelo de entrada, como la asociación *utilizes* o *requires* del ejemplo, no se mapean a la metaclase *Association* de UML ya que estas representan asociaciones entre conceptos y no un nuevo tipo de asociación. Este es un error bastante frecuente en la definición de perfiles UML, incluyendo perfiles existentes para estándares de Safety como el presentado en (Panesar-Walawege et al., 2013). Es precisamente este tipo de interpretaciones y definiciones las que se ven facilitadas mediante la propuesta ya que encapsula las decisiones en un conjunto de reglas de transformación que se aplican sobre el metamodelo de Safety para la generación automática del perfil UML.

Antes de pasar a describir las diferentes reglas utilizadas en este proyecto, es necesario familiarizarse con el concepto de trasformación de modelos y la forma de definir las reglas ATL.

En el contexto de transformaciones M2M (model-to-model) no existe una gran cantidad de lenguajes para definir reglas de transformación de modelos. Dentro de ellos se encuentran los siguientes:

- QVT: No es un software, sino una especificación en la que se basan herramientas como ATL.
- QVT Smart: Es una implementación de QVT pero con poca documentación.
- ModelMorf: Es una herramienta propietaria y con escasa documentación.
- Henshin: Las reglas que permiten definir son muy simples y no son suficientes robustas para este proyecto.

Finalmente se encuentra el lenguaje ATL, es el más utilizado y cuenta con una gran cantidad de documentación, ya que forma parte como solución estándar por

el proyecto Eclipse M2M que es el que contiene todos los plugins necesarios para trabajar en modelización conceptual sobre la plataforma Eclipse Modelling.

ATL posee múltiples características que se describen en secciones siguientes, convirtiéndose en la mejor opción para este proyecto, ya que se puede realizar transformaciones desde metamodelos, por lo que podemos utilizar el metamodelo Safety definido anteriormente, y como utiliza reglas OCL como sintaxis se puede realizar consultas complejas para definir el perfil UML.

Características del lenguaje ATL

El lenguaje ATL consiste en la transformación de modelos construidos en Eclipse. Desde un modelo de entrada hasta un modelo de salida, se define una transformación.

Este lenguaje se considera híbrido, pues permite el uso tanto de construcciones imperativas como declarativas, es decir, para el caso imperativo es necesario que el programador controle el flujo del programa, y para el caso declarativo el programador define lo que el programa debe hacer declarando condiciones, restricciones, etc. Pero no indica exactamente cómo.

Al definir un archivo ATL, este debe contener los siguientes elementos:

- Un nombre, se define un identificador del archivo. Este debe ir acompañado de la palabra clave *module.*
- Una cabecera, se definen los metamodelos que intervienen en la transformación.
- Una sección opcional de importación de librerías, en caso que resulten necesarias.
- Un conjunto de métodos, similares a los presentes en lenguajes como C++ o Java, que en el lenguaje ATL se conocen con el nombre de *helper*.
- El conjunto de reglas de transformación que permiten convertir instancias de un metamodelo hacia instancias de otro.

En la siguiente ilustración, se muestra un ejemplo de como se declara la cabecera de un modulo ATL, y consiste en transformar un metamodelo F*amilies* a un metamodelo *Persons*.

```
module Families2Persons;
create OUT: Persons from IN: Families;
```

Ilustración 25: Ejemplo de cabecera de una ATL.

El nombre del archivo ATL aparece despues de la palabra clave *module*, en el caso mostrado es *Families2Persons*, mientras que los metamodelos utilizados en la transformación deben indicarse siguiendo la estructura que se puede ver en la segunda linea del ejemplo, es decir, *create OUT*: Persons from *IN*: Families, donde Persons es el nombre del metamodelo de destino y el metamodelo de origen es Families. Además para que esta linea tenga sentido, es necesario haber declarado previamente cual es el metamodelo de origen (*IN*) y el de destino (*OUT*). Esto se debe realizar en la configuraciones de ejecución de Eclipse Modelling, como se detallara en la sección guia de usuario.

En relación a la importación de librerias, se declaran mediante la instrucción *uses*. De este modo, si se quisiera importar la librería que contiene las funciones para tratar *strings*, se realiza como indica la siguiente ilustración.

```
uses string;
```

Ilustración 26: Ejemplo de librería ATL.

En cuanto a los *helper*, su declaración es similar a una función Java. Los *helper* constan de:

- Un nombre, se define para diferenciar de otros *helper*. Es definido luego de la palabra "*def:"* y lleva acompañado parentesis.
- El tipo del objeto al que aplica, puede existir el caso que se omita. Dentro del *helper* se define luego de la palabra clave "*context*".

- El tipo del objeto que retorna. Dentro del *helper* se define despues del nombre.
- Un conjunto de parametros que puede ser vacío. Dentro del *helper* se define dentro de los parentesis que estan presentes en el nombre.
- Una expresión ATL que define como se calcula el resultado a retornar. Dentro del helper se define despues del tipo de dato que retorna.

```
helper context Families!Member def: isFemale(): Boolean =
    if not self.familyMother.oclIsUndefined() then
        true
    else
        if not self.familyDaughter.oclIsUndefined() then
            true
        else
            false
        endif
    endif;
```

Ilustración 27: Ejemplo de Helper ATL.

En la ilustración 27, observamos un ejemplo de *helper*, y se denomina como *isFemale(),* no recibe ningún parámetro como entrada. Está presente en el contexto de *Families!Member* y retorna como valor un *boolean*. El *helper* indica si el miembro de la familia es mujer o no, comprobando que *familyMother* o *familyDaughter* está definido o no.

Con respecto a las reglas de transformación, se identifican las *matched rule*, *called rule* y *lazy rule*, esta última se identifican dos tipos, *lazy rule* y *unique lazy rule*.

Las reglas *matched rule* corresponden al tipo de programación declarativa. En la siguiente ilustración se muestra un ejemplo.

```
rule Member2Male {
    from
        s: Families!Member (not s.isFemale())
    to
        t: Persons!Male (
            fullName <- s.firstName + ' ' + s.familyName
        )
}
```

Ilustración 28: Ejemplo de matched Rules ATL.

En la ilustración 28, observamos la regla *Member2Male* que considera desde el metamodelo de origen los miembros de la familia (*from s: Families!Member*) que no representa una mujer. Para establecer esta condición se utilizó el *helper isFemale().* Luego se crea una persona de sexo masculino (*Persons!Male*), agregando el atributo *fullName*, el cual está compuesto por el primer nombre (*firstName*) y el apellido (*familyName*).

Las *called rule*, proveen al programador de mecanismos propios de la programación imperativa. Este tipo de programación se asemeja a los *helper*s que fueron descritos anteriormente, ya que pueden ser invocadas y reciben parámetros. La diferencia con los *helpers* es que pueden crear objetos en el metamodelo de destino para ser incluidos en la instancia resultado de la transformación. En la siguiente ilustración visualizamos un ejemplo.

```
rule NewPersons (_name: String, _familyName: String,
                 listPersonsCreate: Set(String)){
    to
        p : MMPersons!Person (
            name <- _name
        )
    do {
        p.familyName <- _familyName;
        listPersonsCreate.add(p.name + " " + p.familyName);
    }
}
```

Ilustración 29: Ejemplo de called rules ATL.

En la ilustración 29, observamos que las *called rule* tiene dos secciones: *to* y *do*. En la primera de ellas se define el objeto y propiedades que se desea crear, al igual que para el tipo *matched rule*. En el ejemplo se crea el objeto *MMPersons!Persons* con la propiedad *name*.

En la sección do, no se pueden crear objetos, pero si es posible asignar un valor a cualquiera de los atributos de los objetos creados, llamar a otra *called rule*, etc. En el ejemplo, se añade el nombre completo de la persona creada en una lista que se recibe por parámetro. Este hecho no se puede realizar en la sección to, puesto que se modifica una variable externa a la clase que se está creando.

En resumen, el ejemplo mostrado consiste en crear una instancia de la clase *Persons* del metamodelo *MMPersons*, además de los atributos *name* y *familyName*.

Las *lazy rule* es un tipo de regla ATL que permite generar un tipo de objeto. Al igual que las *matched rule* está compuesta por las secciones: *from* y *to*. La diferencia es que las *lazy rule* deben ser invocadas dentro de una *matched rule* o *called rule*, de no ser invocada no genera ningún cambio en el modelo de salida. Aquí se distinguen dos tipos: *lazy rule* simple y *unique lazy rule*. La diferencia entre ellas, es que la primera genera un elemento en el modelo de destino por

cada invocación. En la segunda en cambio, genera un elemento en el modelo de destino, pero revisa los elementos generados, de esta forma no se crean elementos repetidos. En las siguientes ilustraciones se aprecia ejemplos de ambos tipos.

```
lazy rule getCross {
   from
     i: ecore!EObject
   to
     rel: metamodel!Relationship (
     )
 }
```

Ilustración 30: Ejemplo de lazy rule ATL.

```
unique lazy rule getCross {
   from
     i: ecore!EObject
   to
     rel: metamodel!Relationship (
     )
 }
```

Ilustración 31: Ejemplo de unique lazy rule ATL.

Reglas de transformación para la generación de un Perfil UML para Safety

Las reglas de transformación definidas en este proyecto toman como referencia el estándar UML para garantizar el correcto diseño de un perfil UML, y la correcta y completa traducción de cada concepto y propiedad del metamodelo de Safety en un elemento equivalente del perfil UML generado. En total, se han implementado 4 reglas de transformación derivadas de las reglas propuestas en (Giachetti et al., 2009) para la generación de perfiles UML. Además, existe una quinta regla de trasformación que se detalla en este trabajo que no afecta la definición del perfil. Esta quinta regla está asociada a la generación de Paquetes (UML Packages) que mejoran la visualización y comprensión del perfil generado, ya que mantiene la estructura de paquetes que agrupan las clases del metamodelo de entrada en el perfil generado para agrupar los estereotipos. Para

la implementación de las reglas de transformación se ha utilizado el lenguaje de transformaciones modelo-a-modelo ATL. Estas reglas de transformación se aplican sobre el metamodelo de Safety extendido con el perfil *IMProfile* que incorpora la propiedad *TargetElement* para especificar el mapeo con el metamodelo UML destino. Las reglas definidas se describen a continuación.

Regla 1: Generación de estereotipos para representar los conceptos del Metamodelo de Safety de Entrada.

Esta regla consiste en generar un estereotipo por cada clase del metamodelo de safety. El estereotipo extiende a la clase UML referenciada por el atributo *TargetElement* del perfil *IMProfile*. El nombre del estereotipo generado es igual al nombre de la clase relacionada del Metamodelo de Safety.

Esta regla es la principal, ya que genera los estereotipos que representan el elemento principal de un Perfil UML. Las demás reglas son generadas acorde a los resultados obtenidos por esta regla.

La ilustración 32 ejemplifica esta regla, las líneas punteadas con flecha representan la información almacenada en la propiedad *TargetElement*.

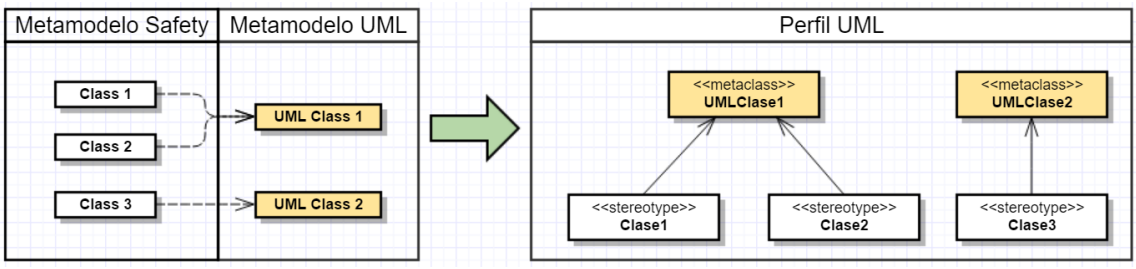

Ilustración 32: Ejemplo genérico para la Regla 1.

La ilustración 33 muestra el código ATL que implementa esta primera regla, donde se puede observar cómo se recorren las clases *UML!Class* (Metaclases) del metamodelo safety de entrada *IN*. Estas clases se han extendido con el estereotipo *MappingInfo* que tiene la propiedad *TargetElement* que es recuperada mediante el helper ATL *getTargetElement* (ver ilustración 35) y es

utilizado para generar el estereotipo correspondiente que tiene el mismo nombre que la clase del metamodelo de Safety analizado.

```
75 rule genStereotypes{
76     from
77         s: UML!Class in IN (if s.hasStereotype('MappingInfo') then
78                                 if s.getTargetElement().oclIsUndefined() then
79                                     false
80                                 else
81                                     true
82                                 endif
83                             else
84                                 false
85                             endif
86         )
87
88     to
89         t: UML!Stereotype (
90             name <- s.name,
91             ownedRule <- s.ownedRule,
92             ownedAttribute <- Set{ s.ownedAttribute->flatten()},
93             ownedConnector <- s.ownedConnector,
94             ownedBehavior <- s.ownedBehavior,
95             classifierBehavior <- s.classifierBehavior,
96             ownedOperation <- s.ownedOperation
97         )
```

Ilustración 33: Regla genStereotypes primera parte.

La ilustración 33, muestra la regla que genera los estereotipos, y cuya descripción es la siguiente:

- Línea 76-85: Para recorrer las clases del metamodelo safety, deben cumplirse dos condiciones, la primera de ella consiste en que las clases del metamodelo de Safety deben tener aplicado el estereotipo *MappingInfo*. La segunda consiste en la propiedad *TargetElement*, esta debe contener la metaclase que extienda el estereotipo generado.
- Línea 89-97: Se genera el estereotipo, por cada clase del metamodelo de Safety, que cumpla las condiciones antes descritas. Las propiedades generadas, son las siguientes.
    - *Name*: Nombre del estereotipo.

- *OwnedRule*: Asignar restricciones a los estereotipos, cuando sea el caso.
- *OwnedAttribute*: Permite asignar elementos de tipo atributos, extensión y port.
- *OwnerConnector*: Permite asignar un dato de tipo conector al estereotipo.
- *ownedBehavior*, *classifierBehavior* y *ownedOperation*.

La regla genStereotypes cuenta con una tercera parte (*do*), dónde son definidas las condiciones que deben cumplirse para los diferentes casos presentes en la regla 4 (descrita más adelante) y además son llamadas las reglas *genGeneralization* y *genExtension*. (ver la ilustración 34).

```
104 do {
105     if (s.superClass->size() > 0) {
106         for (e in s.superClass){
107             if (e.getTargetElement() = s.getTargetElement()) {
108                 t.generalization <- thisModule.genGeneralization(e);
109             }
110             else {
111                 thisModule.genExtension(s.name, s.getTargetElement(), base_element, s);
112             }
113         }
114     }
115     else {
116         thisModule.genExtension(s.name, s.getTargetElement(), base_element, s);
117     }
118 
119         ('[IM].' + s.name + ' --> [PRO].' + s.name).println();
120         ('================//').println();
121         (' ').println();
122     }
123 }
```

Ilustración 34: Regla genStereotypes segunda parte.

En la ilustración 34 observamos el código ATL que implementa la regla *genStereotypes*. Se evalúa si los estereotipos generados representan herencia de otro estereotipo generado. Por otra parte, se evalúa si el estereotipo padre e hijo están mapeados con la misma metaclase del metamodelo UML. Cuando ambas condiciones se cumplen, la regla genera el elemento herencia (*generalization*) entre el estereotipo padre e hijo (regla *genGeneralization*) y entre

el estereotipo padre y la metaclase UML se genera la extensión correspondiente (*genExtension*). En secciones más adelante serán descritas las reglas mencionadas.

```
7  helper context UML!Element def: getTargetElement(): UML!Element =
8      self.getValue(self.getAppliedStereotype('IMProfile::MappingInfo'), 'TargetElement');
```

Ilustración 35: Helper getTargetElement().

La ilustración 35 muestra el helper *getTargetElement*, que es utilizado por la regla 1. El helper cuenta con dos funciones, y la descripción es la siguiente:

- *getValue*: Obtiene el valor de la propiedad *TargetElement* del metamodelo de Safety después de aplicar el perfil *IMProfile*.
- *getAppliedStereotype*: Obtiene los elementos del metamodelo de Safety que tenga aplicado el perfil *IMProfile*.

En resumen, el helper tiene el objetivo de obtener los elementos *UML!Class* del metamodelo de Safety, siempre y cuando este aplicado el perfil *IMProfile* y la propiedad *TargetElement* posea algún dato.

Regla 2: Generación de valores etiquetados (propiedades) en el perfil UML.

Por cada estereotipo generado se analizan las propiedades de la clase (metaclase) correspondiente del metamodelo de Safety. Estas propiedades pueden representar atributos o links (relaciones) entre clases. Cuando el tipo de la propiedad es dato-valuado está representando un atributo de la clase, y cuando el tipo es objeto-valuado (una clase del modelo) está representando un link entre clases. Para aquellas propiedades que no estén mapeadas, es decir, que no tengan valor asignado en la extensión *TargetElement*, se define una nueva propiedad (valor etiquetado) en el estereotipo correspondiente, generado a partir de la clase que contiene la propiedad no mapeada. La propiedad tiene el mismo nombre, tipo y cardinalidad que la propiedad del metamodelo de Safety. La ilustración 36 ejemplifica de manera genérica esta regla de transformación.

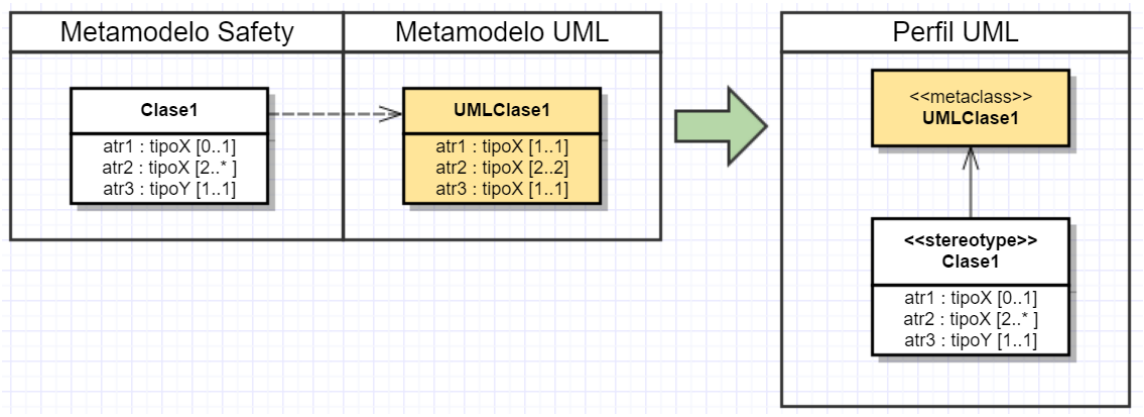

Ilustración 36: Ejemplo genérico para la regla 2.

Las propiedades mapeadas no son consideradas en la generación del perfil, ya que el mapeo indica que en UML existe una propiedad que representa lo mismo y por lo tanto no requiere incorporarse como extensión en el perfil UML. Pueden existir propiedades en el metamodelo de Safety equivalentes semánticamente con propiedades del metamodelo UML, pero que presenten diferencias sintácticas que no permitan su mapeo, por ejemplo, diferencias de cardinalidad, o diferencias de tipo de datos. En estos casos, para establecer el mapeo se prioriza la equivalencia sintáctica, y por lo tanto el mapeo no se debe establecer. Al no estar mapeados, estas propiedades serán generadas como nuevos valores etiquetados de acuerdo a la regla 2.

La ilustración 37 muestra el código ATL que implementa la segunda regla, para el caso de las propiedades que representan un atributo de la clase, donde se puede observar cómo se recorren las propiedades *UML!Property* del metamodelo Safety de entrada *IN*. Estas propiedades se han extendido con el estereotipo *MappingInfo* que tiene la propiedad *TargetElement* que es obtenida mediante el *helper* ATL *getTargetElement.* Sin embargo, para el caso de las propiedades no fue necesario realizar el mapeo con alguna metaclase UML, por tanto, la propiedad *TargetElement* estará vacía durante la ejecución de las reglas ATL.

```
126 rule newProperties {
127     from
128         s: UML!Property in IN ( if s.hasStereotype('MappingInfo') then
129             if  s.getTargetElement().oclIsUndefined() then
130                 if s.association.oclIsUndefined() then
131                     true
132                 else
133                     false
134                 endif
135             else
136                 false
137             endif
138         else
139             false
140         endif
141         )
142     to
143         t: UML!Property (
144             name <- s.name,
145             type <- s.type,
146             lower <- s.lower,
147             upper <- s.upper
148         )
149     do {
150         ('[IM].' + s.class.name + '.' + s.name + ' --> [PRO].' + s.class.name + '.' + s.name).println();
151         ('================//').println();
152         (' ').println();
153     }
154 }
```

Ilustración 37: Código ATL que implementa la regla 2.

La ilustración 37 muestra la regla newProperties, donde para recorrer las propiedades deben cumplirse tres condiciones. La primera condición consiste en tener aplicado el estereotipo *MappingInfo* en cada una de las propiedades del metamodelo Safety, la segunda condición consiste en la propiedad *TargetElement* se encuentre vacía, y la tercera condición consiste en no estar definido el elemento asociación. Posteriormente se generan las propiedades correspondientes por cada una existente que cumpla las condiciones. Las propiedades generadas tienen los siguientes atributos:

- *name*: nombre de la propiedad. Es el mismo que existe en el metamodelo Safety de entrada.
- *type*: tipo de la propiedad. Es el mismo definido en el metamodelo Safety de entrada.
- *lower*: Cardinalidad inferior de la propiedad. Es la misma definida en el metamodelo Safety de entrada.

- *Upper:* Cardinalidad superior de la propiedad. Es la misma definida en el metamodelo Safety de entrada.
- *association y aggregation:* se agregan las asociaciones y agregaciones existentes en el metamodelo Safety de entrada.

La ilustración 38 muestra el código ATL que implementa la regla 2 para el caso de las propiedades que representan una asociación entre dos clases.

```
rule genProperty {
    from
        s: UML!Property in IN ( if s.hasStereotype('MappingInfo') then
            if  s.getTargetElement().oclIsUndefined() then
                if not s.association.oclIsUndefined() then
                    true
                else
                    false
                endif
            else
                false
            endif
        else
            false
        endif
        )
    to
        t: UML!Property (
            name <- s.name,
            type <- s.type,
            lower <- s.lower,
            upper <- s.upper,
            association <- s.association
        )
    do {
        ('[IM].' + s.class.name + '.' + s.name + ' --> [PRO].' + s.class.name + '.' + s.name).println();
        ('================//').println();
        (' ').println();

    }
}
```

Ilustración 38: Código ATL que implementa la regla 2

La regla *genProperty* con respecto a la regla anterior, presenta dos diferencias. La primera diferencia se refiere a la tercera condición que se debe cumplir para generar la regla, y consiste en considerar las propiedades que tienen definido el elemento asociación. La segunda diferencia es el atributo *association* que se genera al aplicar la regla. Los demás elementos son iguales a la regla anterior, por lo tanto, no es necesario definirlos nuevamente.

Regla 3: Generación de enumeraciones en el perfil UML.

Por cada enumeración del Metamodelo de Safety que no tenga equivalencia (no esté mapeada) con el metamodelo UML, se creará una nueva enumeración en el perfil UML con el mismo nombre y atributos literales que la enumeración del metamodelo de Safety correspondiente. Los valores etiquetados generados a partir de atributos del metamodelo de Safety que hagan referencia a la enumeración transformada, tendrán como tipo dato la enumeración generada en el perfil UML. En el caso de enumeraciones del metamodelo de Safety mapeadas al metamodelo UML que presenten diferencias en sus valores literales, también deben ser tratadas como nuevas enumeraciones y por lo tanto no deben estar mapeadas con la enumeración UML. Esto es debido a que mediante un perfil UML no es posible agregar valores literales a enumeraciones existentes en el metamodelo UML. En estos casos, los atributos del metamodelo de Safety cuyo tipo de dato haga referencia a estas enumeraciones no deben ser mapeados y, por lo tanto, serán tratados como nuevos atributos de acuerdo a lo establecido en la Regla 2. La ilustración 39 muestra el mapeo realizado de forma correcta con la generación del perfil correspondiente aplicando la Regla 3.

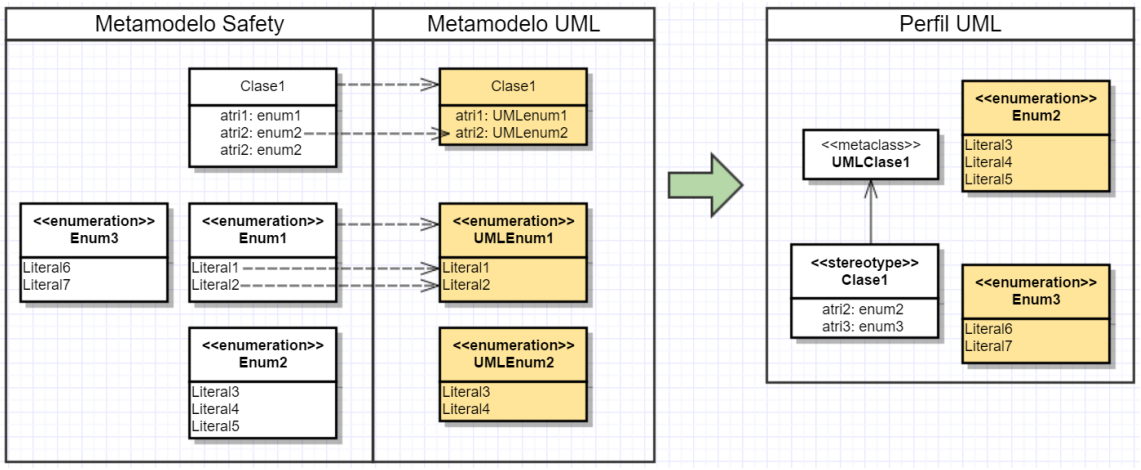

Ilustración 39: Ejemplo genérico de la regla 3.

La ilustración 40 y 41 muestran el código implementado para generar la regla 3, dónde se puede observar cómo se recorren los elementos enumeraciones y literales del metamodelo Safety de entrada *IN.*

```
rule genEnumeration {
    from
        s: UML!Enumeration
    to
        t: UML!Enumeration (
            name <- s.name,
            ownedLiteral <- s.ownedLiteral
        )

    do {
        ('ENUMERATION:' + s.name).println();
        ('================//').println();
        (' ').println();
    }
}
```

Ilustración 40: Código ATL que implementa la regla 3, regla genEnumeration.

```
rule genEnumerationLiteral {
    from
        s: UML!EnumerationLiteral
    to
        t: UML!EnumerationLiteral(
            name <- s.name
        )
    do {
        ('LITERAL:' + s.name).println();
        ('================//').println();
        (' ').println();
    }
}
```

Ilustración 41: Código ATL que implementa la regla 3, regla genEnumerationLiteral.

Regla 4: Generación de relaciones de generalización entre estereotipos

Esta regla considera 3 casos diferentes para manejar las generalizaciones entre estereotipos.

Caso 1: Si hay dos clases relacionadas mediante una asociación de generalización en el metamodelo de Safety y ambas clases están mapeadas a la misma clase del metamodelo UML, y además la relación de generalización no tiene equivalencia en el metamodelo de UML, entonces se representa la misma generalización entre los estereotipos correspondientes en el perfil UML y la extensión a la clase UML sólo se aplica al estereotipo generado a partir de la clase padre del metamodelo de Safety. Este caso se ve reflejado entre Clase1 y Clase3 del ejemplo en la ilustración 42.

Caso 2: Si hay dos clases relacionadas mediante una asociación de generalización en el metamodelo de Safety y las clases están mapeadas a distintas clases del metamodelo de UML, entonces la generalización no es representada en el perfil UML y las propiedades heredadas se repiten en cada estereotipo generado. Esto se ve ejemplificado en la Clase4 de la ilustración 42. Esta decisión de eliminar la herencia entre estereotipos responde a que las relaciones de extensión también se heredan, por lo tanto, si el estereotipo Clase4 mantuviera la relación de herencia con Clase3, este estereotipo también podría extender a la metaclase UML Clase2, lo que es incorrecto.

Caso 3: Si hay dos clases del metamodelo de Safety relacionadas con una asociación de generalización que tiene equivalencia en el metamodelo de UML, entonces la generalización no es representada en el perfil UML, y se aplica la regla 1 de forma normal. Este caso se ve reflejado entre las clases Clase1 y Clase2 de la ilustración 42. Es importante señalar que una generalización equivalente se puede reconocer de manera automática y por lo tanto no es necesario incorporar esta información de mapeo.

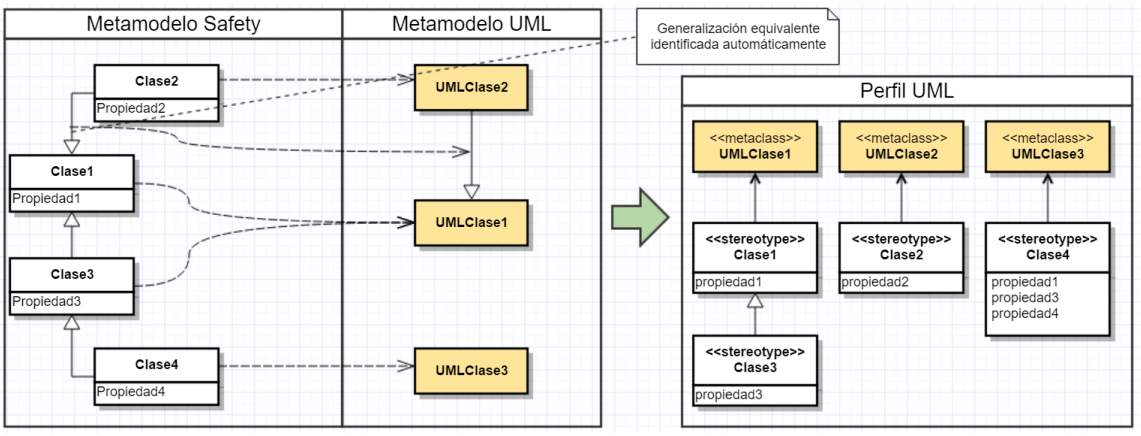

Ilustración 42: Ejemplo genérico para las reglas 4, 5 y 6.

La ilustración 43 muestra el código ATL que implementa la regla 4, dónde se puede observar cómo se recorren los estereotipos generados en la regla 1. Para implementar los tres casos descritos anteriormente, fue necesario definir en la regla 1 las condiciones que permitieran identificar la existencia de herencia entre dos estereotipos. En la regla 1 se llama a la regla *genGeneralization*, generando la herencia correspondiente.

La herencia generada cuenta con dos atributos, *general* que representa el nombre de la herencia y *isSubstitutable* que es valorado como *true*.

```
186 lazy rule genGeneralization {
187     from
188         s : UML!Stereotype
189     to
190         t : UML!Generalization(
191             general <- s,
192             isSubstitutable <- true
193         )
194     do{
195         ('Generalization:' + s.name).println();
196         ('================//').println();
197         (' ').println();
198     }
199 }
```

Ilustración 43: Código ATL que implementa la regla 4. Regla genGeneralization.

Cuando no corresponda aplicar una relación de herencia entre dos estereotipos o es necesario establecer la extensión entre el estereotipo padre y la metaclase

UML, se genera una extensión. En la ilustración 44 muestra la regla *genExtension*, que genera las extensiones correspondientes. La regla recibe tres parámetros:

- *Name*, representa el nombre de las extensiones.
- *getTargetElement*, helper que retorna las metaclases UML mapeadas con las clases del metamodelo de Safety.
- *clase,* variable que retorna las instancias clases del metamodelo de Safety.

Las instancias generadas por la regla ATL son tres. La primera de ellas *UML!Extension* que extiende el estereotipo con la metaclase UML, para el caso particular de este proyecto corresponde a *metaclase Class*. Los atributos que posee la instancia corresponden a *name* (compuesto por el nombre del atributo y la metaclase UML). Atributo *memberEnd* (conjunto de las variables *extensionEnd* y *base_element*). Atributo *ownedEnd* (conjunto de la variable *extensionEnd*).

La segunda corresponde a la instancia *UML!ExtensionEnd* (variable *extensionEnd)*, que extiende la metaclase UML con el estereotipo. Los atributos que posee la instancia corresponden a *name* (compuesto por la exprension *extensión_* sumado al nombre del estereotipo). Atributo *aggregation* (el valor corresponde al elemento *composite* de UML). Atributo *type* (el valor corresponde al nombre de la clase). Atributo *association* (se relaciona con la instancia *UML!Extension*).

La última instancia corresponde a UML!Property, generando la propiedad base_element de tipo a la clase correspondiente asociada al estereotipo.

```
196 rule genExtension (name : String , getTargetElement: UML!Element, clase : UML!Class){
197     to
198         t : UML!Extension (
199             name <- name + '_' + getTargetElement.name,
200             memberEnd <- Set {extensionEnd,base_element},
201             ownedEnd <- Set{extensionEnd}
202         ),
203         extensionEnd: UML!ExtensionEnd (
204             name <- 'extension_' + name,
205             aggregation <- #composite,
206             type <- clase,
207             association <- t
208         ),
209         base_element: UML!Property (
210             name <- 'base_element',
211             isUnique <- false,
212             type <- getTargetElement
213         )
214     do {
215         ('Extension:' + t.name).println();
216         ('ExtensionEnd:' + extensionEnd.name).println();
217         ('================//').println();
218         (' ').println();
219     }
220 }
```

Ilustración 44: Código ATL que implementa la regla genExtension.

A continuación, analizamos la cabecera, reglas y *helper* que fue necesario implementar para apoyar las reglas principales, además de entregar un orden para la presentación del perfil UML generado por la transformación ATL.

Cabecera ATL

La cabecera en la transformación ATL corresponde al nombre, las entradas (*IN*, *PRO*) y salidas (*OUT*). Ver Ilustración 41.

```
1 module Standard2Profile;
2 create OUT: UML from IN: UML, PRO: UML;
```

Ilustración 45: Cabecera Transformación ATL.

En la ilustración 45 se muestra la cabecera para el presente proyecto, donde observamos que el nombre es *Standard2Profile*, que representa la transformación del metamodelo de Safety en el perfil UML en base al estándar. En relación a los archivos de entrada y salida, identificamos lo siguiente:

- IN: UML, corresponde al metamodelo de Safety utilizado para la transformación.
- PRO: UML, corresponde al perfil UML *IMProfile,* implementado para ser aplicado al metamodelo de Safety utilizado como entrada.
- OUT: UML, corresponde al archivo que guardara el perfil UML generado de forma automática.

HELPER

Los helper apoyan en la obtención de datos que son requeridos por las reglas ATL. Los implementados son los siguientes:

hasStereotype, tiene la función de identificar que en el metamodelo de Safety este aplicado el estereotipo *MappingInfo*.

```
7  helper context UML!Element def: hasStereotype(name: String): Boolean =
8      not self.getAppliedStereotypes() -> select(s | s.name = name) -> isEmpty();
```

Ilustración 46: Código ATL implementado para el helper hasStereotype.

En la ilustración 46 se observa que el helper, presenta tres funciones y un operador lógico, que se describen a continuación:

- *GetAppliedStereotype*: Esta función valida que el estereotipo esta aplicado.
- Select(): Esta función selecciona los elementos que tengan el estereotipo *MappingInfo*.
- IsEmpty(): Esta función considera solo los elementos que estén vacíos.
- Not: operador lógico que es representa lo contrario a lo descrito en el código ATL que lo acompaña.

getTargetMetaclasses, permite recorrer todas las instancias clases presentes en el metamodelo Safety de entrada *IN.* Además, llama al helper *getTargetElement* (fue descrito en secciones anteriores).

```
10 helper context UML!Element def: getTargetMetaclasses(): Set(UML!Class) =
11     UML!Class.allInstancesFrom('IN') -> collect(e | e.getTargetElement());
```

Ilustración 47: Código ATL implementado para el helper getTargetMetaclasses.

En la ilustración 47 observamos que el helper tiene dos funciones, que se describen a continuación:

- *AllInstancesFrom*: Recorre todas las instancias clases contenidas en el metamodelo Safety de entrada IN.
- *Collect*: Obtiene el conjunto de elementos retornados por el helper *getTagetElement*.

GetImportedTypes, recorre todas las instancias *UML!ElementImport* presentes en el metamodelo Safety de entrada *IN*, seleccionando aquellas que son de tipo UML!PrimitiveType y luego las retorna como un conjunto. Para el caso particular del presente proyecto, esta función retorna las metclases UML que son mapeadas con las clases del metamodelo Safety (ver ilustración 48).

```
13 helper context UML!Element def: getImportedTypes(): Set(UML!PrimitiveType) =
14     UML!ElementImport.allInstancesFrom('IN') -> select(e | e.importedElement -> oclIsTypeOf(UML!PrimitiveType))
15                                              -> collect(e | e.importedElement);
```

Ilustración 48: Código ATL implementado para el helper getImportedTypes.

Rule

Las siguientes reglas son un complemento de las anteriores reglas descritas, permitiendo tener un orden en la generación y agregar otros elementos necesarios para la completitud del perfil UML generado como referencia del estándar Safety.

Regla 5: Generación del elemento Perfil

Para el modelo que implementa el metamodelo de Safety, se genera la instancia UML!Profile, donde serán integrados ciertos elementos generados por otras reglas, a motivo de tener un orden en la presentación del perfil UML generado de forma automática. Además, la presente regla importa los elementos clases que

hacen referencia a la metaclase UML que es mapeada con las clases del metamodelo Safety de entrada (ver ilustración 49).

```
18 ----Regla que genera el perfil con las metaclases UML-----
19 rule genProfile {
20     from
21         s: UML!Model in IN ( s.oclIsTypeOf(UML!Model))
22     to
23         t: UML!Profile (
24             name <- 'Profile IEC61508',
25             nestedPackage <- s.nestedPackage,
26             ownedType <- s.ownedType,
27             packagedElement <- s.packagedElement,
28             elementImport <- Set {s.getImportedTypes()->collect (e |thisModule.imp_type(e)),
29                                   s.getTargetMetaclasses() -> collect (e |thisModule.ref_classes(e))},
30             metaclassReference <- s.getTargetMetaclasses() -> collect (e |thisModule.ref_classes(e))
31         )
32     do {
33         ('New Mapping Info').println();
34         ('================//').println();
35         (' ').println();
36     }
37 }
```

Ilustración 49: Código ATL que implementa la regla genProfile.

Regla 6: Generación de los paquetes (Package) del Metamodelo Safety

Por cada paquete (Package) contenido en el metamodelo Safety de entrada, se genera el mismo paquete en el perfil, con el fin de establecer un orden con los estereotipos, es decir, cada estereotipo generado estará contenido en el paquete correspondiente, de acuerdo a la clase que tenga relación.

La ilustración 50 observamos el código ATL implementado, donde se recorre todos los paquetes del metamodelo Safety de entrada *IN*, generando el mismo paquete con los atributos name (nombre del paquete) y ownedType (permite ciertas instancias dentro del paquete).

```
59 rule genPackage {
60     from
61         s: UML!Package in IN (s.oclIsTypeOf(UML!Package))
62     to
63         t: UML!Package (
64             name <- s.name,
65             ownedType <- s.ownedType
66     )
67     do {
68         ('Package:' + s.name ).println();
69         ('================//').println();
70         (' ').println();
71     }
72 }
```

Ilustración 50: Código ATL implementado para la regla genPackage.

Regla 7: Importación de las Metaclases UML Mapeadas

La regla *ref_classes* recorre las clases que tienen aplicado el estereotipo *MappingInfo*, obteniendo el valor de la propiedad *TargetElement,* para luego generar la instancia UML!ElementImport, que agrega las metaclases *Class* mapeadas en las clases del metamodelo Safety de entrada *IN* (ver ilustración 51).

```
39 unique lazy rule ref_classes {
40     from
41         s : Set(UML!Class)
42     to
43         t : UML!ElementImport (
44             importedElement <- s
45         )
46 }
```

Ilustración 51: Código ATL implementado en la regla ref_classes.

Regla 8: Importación de los Tipos Primitivos

La regla *imp_type* recorre los elementos primitivos en el metamodelo Safety de entrada, y luego genera por cada uno de ellos la instancia UML!ElementImport retornando los tipos de las metclases Class (ver ilustración 52).

```
48 unique lazy rule imp_type {
49     from
50         s : Set(UML!PrimitiveType)
51     to
52         t : UML!ElementImport (
53             importedElement <- s
54         )
55
56 }
```

Ilustración 52: unique lazy rule imp_type.

4.2.4. Evaluación del Correcto Funcionamiento de las Reglas ATL

Para evaluar el funcionamiento de las reglas ATL definidas, nos poyamos en el ejemplo utilizado en los documentos (Panesar-Walawege et al., 2011) y (Panesar-Walawege, Sabetzadeh, & Briand, 2013), donde se utiliza un fragmento del metamodelo MOF de Safety para luego ser aplicado a un metamodelo de dominio, que consiste en un sistema de control de producción submarina.

El fin de utilizar este ejemplo, nos permitirá obtener comparaciones entre la generación automática desarrollada en el presente proyecto y la manual utilizada por los autores, evaluando los objetivos planteados en un comienzo.

Cabe mencionar que esta fuera del alcance del proyecto la correcta definición del metamodelo MOF DE Safety y el metamodelo de dominio utilizado por los autores, ya que implica un análisis de expertos en la materia.

En las siguientes secciones se muestra el fragmento del metamodelo de Safety, la utilización de la herramienta para definir el perfil UML y el fragmento del metamodelo de dominio.

Fragmento paquete conceptos de proceso de Safety

La ilustración 53 muestra el fragmento del metamodelo de Safety utilizado como ejemplo, además de las relaciones entre los elementos, las clases y propiedades presentes. A modo de aclarar los conceptos utilizados se definen cada uno de ellos (ver tabla 3).

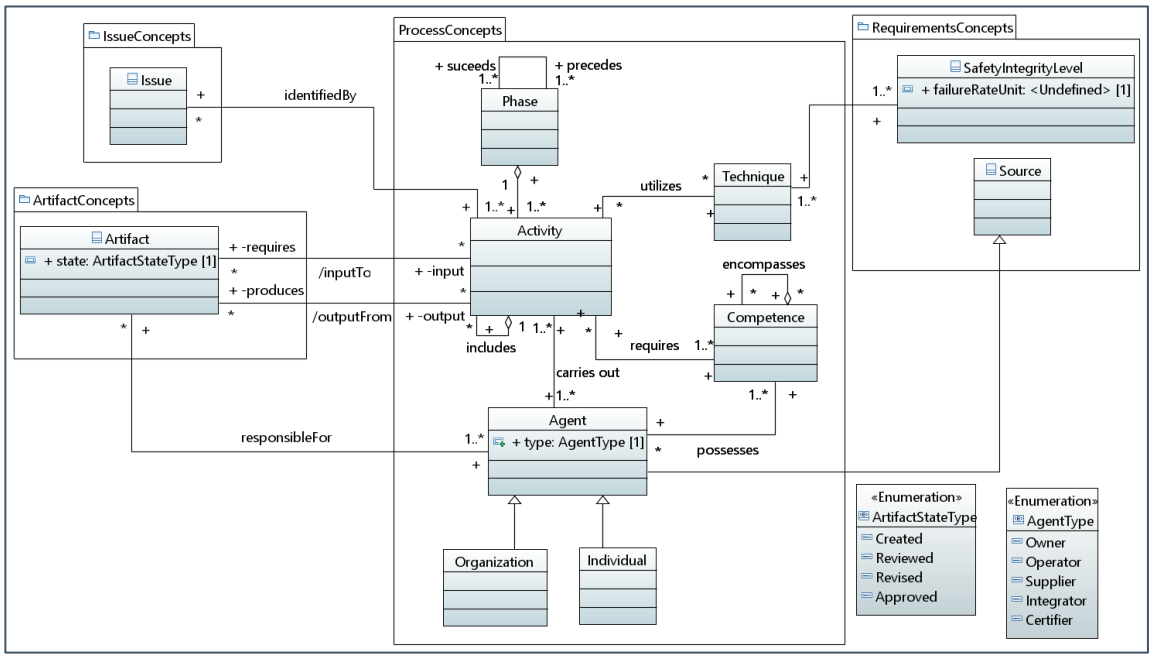

Ilustración 53: Package Process Concepts y sus relaciones.

Issue	Una unidad de trabajo para lograr una mejora en un sistema.
Artifact	Un artefacto es uno de los muchos tipos de bienes de productos producidos durante el desarrollo de software
Phase	Un conjunto actividades con determinadas entradas y salidas que se llevan a cabo en un momento determinado durante la vida de un sistema
Activity	Una actividad puede ser considerada como una unidad de comportamiento en un proceso

Agent	Un concepto abstracto que puede representar una persona u organización que tiene la capacidad y responsabilidad para llevar a cabo una actividad
Organization	Un acuerdo social que persigue objetivos colectivos, que controla su propio desempeño y que tiene un límite que lo separa de su entorno
Individual	Referente a una persona
Technique	Un procedimiento que se utiliza para realizar una tarea o actividad específica
Competence	La capacidad para realizar una determinada tarea, acción o funcionar correctamente
SafetyIntegrityLevel	Se utiliza para especificar el nivel de integridad de seguridad para las funciones de seguridad a ser implementadas por la E/E/PE los sistemas de seguridad; integridad de seguridad se define como la probabilidad de que un sistema relacionado con la seguridad satisfactoriamente las funciones de seguridad requeridas bajo todas las condiciones indicadas dentro de un período determinado de tiempo
Source	Un concepto abstracto que puede representar una persona, organización o estándar que puede ser una entrada de un requerimiento de sistema
ArtifactStateType	El estado del artefacto tal como creado, revisar, corregido, etc.
Agent Type	Enumeración para describir los diferentes roles que los agentes pueden tomar
IdentifiedBy	asociación que vincula un problema a una de las actividades que fue responsable de su creación por ejemplo cuestiones que pueden ser identificados mediante diversas actividades de verificación
Input to	Un vínculo que denota los artefactos que son un aporte a la actividad particular
Output from	Un vínculo que denota los artefactos que se encuentran una salida para determinada actividad
Responsible for	Un agente puede hacerse responsable de la creación o modificación de un artefacto
Utilizies	Una actividad puede utilizar una técnica particular para conseguir su objetivo, por ejemplo, la actividad de análisis de riesgos puede utilizar la técnica de análisis del árbol de fallos para identificar los peligros
Requires	Cada actividad requiere un cierto tipo de competencias de los agentes que llevan a cabo la actividad
Carries out	Un agente puede realizar más de una actividad

Possesses	Esta asociación considera las competencias que tiene un agente

Tabla 3: Definición de Conceptos del paquete Conceptos de procesos.

La implementación del metamodelo de Safety es realizada por un experto en estándar IEC 61508 en conjunto con un experto en el dominio que se requiere certificar el sistema.

Generación del Perfil UML

El siguiente paso es generar el perfil UML de forma automática considerando el metamodelo Safety presentado en la ilustración 52. Antes de generar el perfil UML de Safety, es necesario aplicar el perfil *IMProfile* implementado de forma manual, con el objetivo de establecer el mapeo entre las clases contenidas en el metamodelo Safety y la metaclase UML *Class*. En la sección guía de usuario se define el procedimiento para aplicar el perfil *IMProfile*.

Aplicado el perfil *IMProfile* en el metamodelo de Safety se observa el estereotipo *MappingInfo,* como se muestra en la ilustración 54. El estereotipo contiene la propiedad *TargetElement,* esta propiedad va contener el valor de la metaclase UML *Class.*

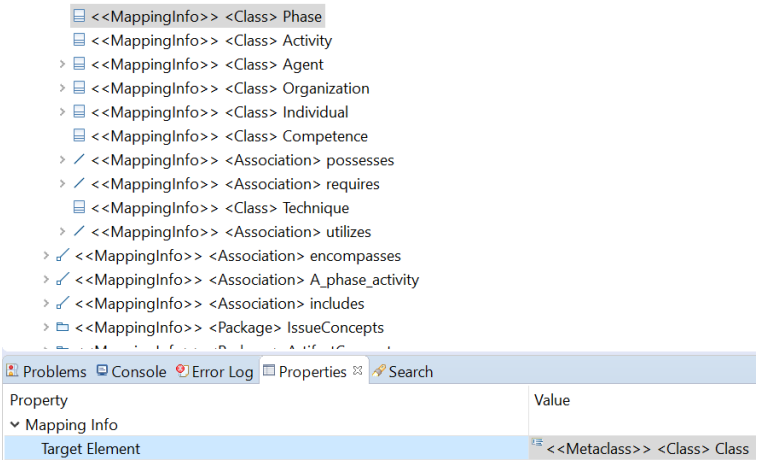

Ilustración 54: Ejemplo de mapeo entre la clae Phase y la metaclase class.

Ejecutando las reglas ATL definidas anteriormente (El procedimiento es descrito en la sección guía de usuario) al metamodelo Safety aplicado el perfil IMProfile, se obtiene el perfil UML de la ilustración 55. El perfil UML generado se muestra de forma jerárquica, por el editor UML que utiliza la herramienta eclipse. Los estereotipos son agrupados por el paquete que pertenecía la clase en el metamodelo de Safety, los atributos están contenidos en los estereotipos según corresponda, las asociaciones y extensiones se presen de forma independiente.

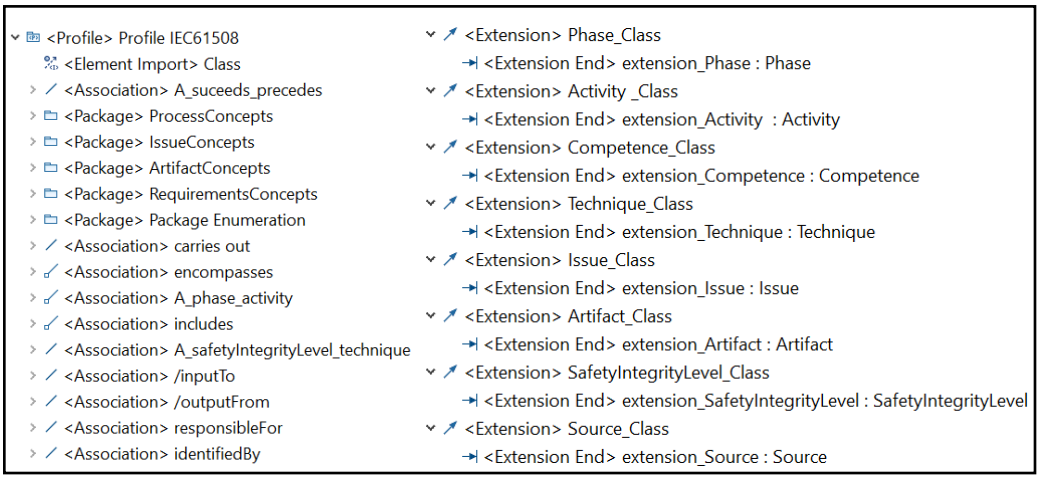

Ilustración 55: Perfil UML generado de forma automática.

Es necesario realizar una comparación entre el perfil UML generado de forma manual y el automático propuesto por el presente documento, para ser más intuitivo el análisis presentamos el perfil UML automático en forma gráfica, entender que se realiza de forma manual solo para el análisis de la siguiente sección

Perfil UML Manual versus Perfil UML Automático

La ilustración 57 presenta el perfil UML generado automáticamente aplicando las reglas de transformación definidas anteriormente al metamodelo de Safety presentado en la ilustración 53. En el trabajo presentado en (Panesar-Walawege et al., 2013) y (Panesar-Walawege et al., 2011a), se utilizó el mismo metamodelo de entrada para generar el perfil manualmente sin un

mecanismo de mapeo ni reglas de transformación concretas. Este perfil UML se ve en la ilustración 58. Al analizar ambos perfiles se pueden ver una clara diferencia: las asociaciones entre clases del metamodelo de Safety se definieron como extensiones de la metaclase UML *Association* en el perfil generado manualmente (ilustración 58) y los extremos de las asociaciones como extensiones de la metaclase UML *Property*. Mientras que en el perfil UML generado automáticamente (ilustración 57), estas asociaciones se definieron directamente sobre los estereotipos generados. Este es un error bastante común en la definición de Perfiles UML, ya que se aprovecha la posibilidad de utilizar propiedades en los estereotipos para definir nuevas asociaciones entre conceptos. Además de complicar la definición y comprensión del perfil definido, al extender la clase UML *Association* se están definiendo nuevos tipos de asociaciones que podrían aplicarse a cualquier clase del modelo extendido por el perfil, no tan sólo el conjunto de elementos que de acuerdo al metamodelo pueden estar relacionados. Para mantener la lógica del metamodelo de entrada, los autores del artículo proponen un conjunto de reglas OCL, también definidas manualmente, que permiten restringir el conjunto de elementos que se pueden relacionar con cada tipo de asociación. Esto nuevamente agrega una mayor complejidad al perfil, sobrecarga su definición, y hace uso inadecuado de las opciones de extensión que el estándar UML propone.

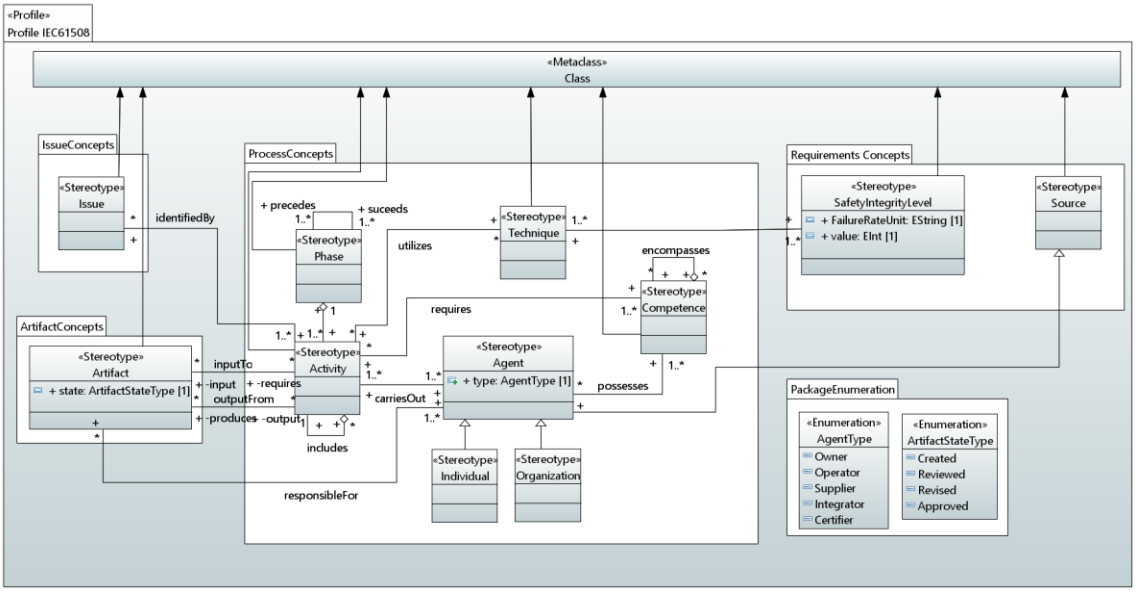

Ilustración 57: Perfil UML de Safety generado automáticamente.

Un problema similar al de las asociaciones ocurre con los elementos tipo Package. Por este motivo no se ven en la agrupación de los distintos conceptos. Finalmente, las enumeraciones y relaciones de herencia no se ven reflejadas en el perfil generado.

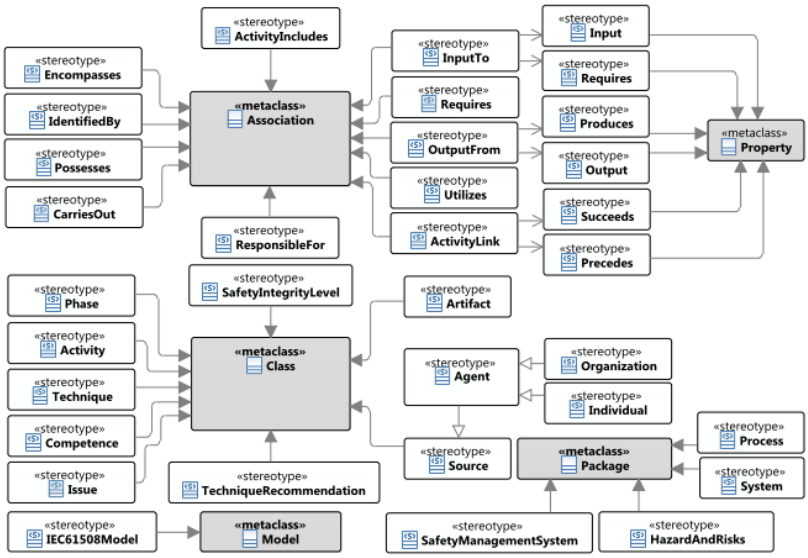

Ilustración 58: Perfil UML de Safety generado manualmente.

En la ilustración 57, el perfil UML de Safety generado automáticamente, se identifican las siguientes diferencias con la ilustración 58.

- Los elementos tipo *Association* relacionan a los diferentes estereotipos generados por las reglas ATL.
- Los elementos tipo *package* solo representan un orden en el modelo, y no influyen en el perfil.
- Los elementos tipo *Generalization* son generados según la implementación de las reglas ATL, por lo tanto, existen estereotipos que no se extienden directamente de la metaclase *Class,* son representados como herencia de otro estereotipo.

Claramente se observa la correcta definición del perfil UML de Safety aplicando las reglas ATL, además comparando la cantidad de elementos generados, existe una completitud en la definición, cumpliendo una correcta generación del perfil.

Aplicación del Perfil UML automático a un modelo de dominio

Para validar el enfoque propuesto en el presente trabajo, es necesario aplicar el perfil UML generado de forma automática en un modelo de dominio de forma concreta.

La ilustración 59 muestra un fragmento del modelo de dominio para un sistema de control submarina. Los conceptos del domino se definen en estrecha consulta con un experto en el área, y través de una lectura de la literatura pertinente donde se describe la arquitectura y los componentes de los sistemas submarinos.

Esta fuera del alcance del presente trabajo definir un modelo de dominio, por lo tanto, nos apoyamos en el fragmento presentado en el documento (Panesar-Walawege et al., 2013).

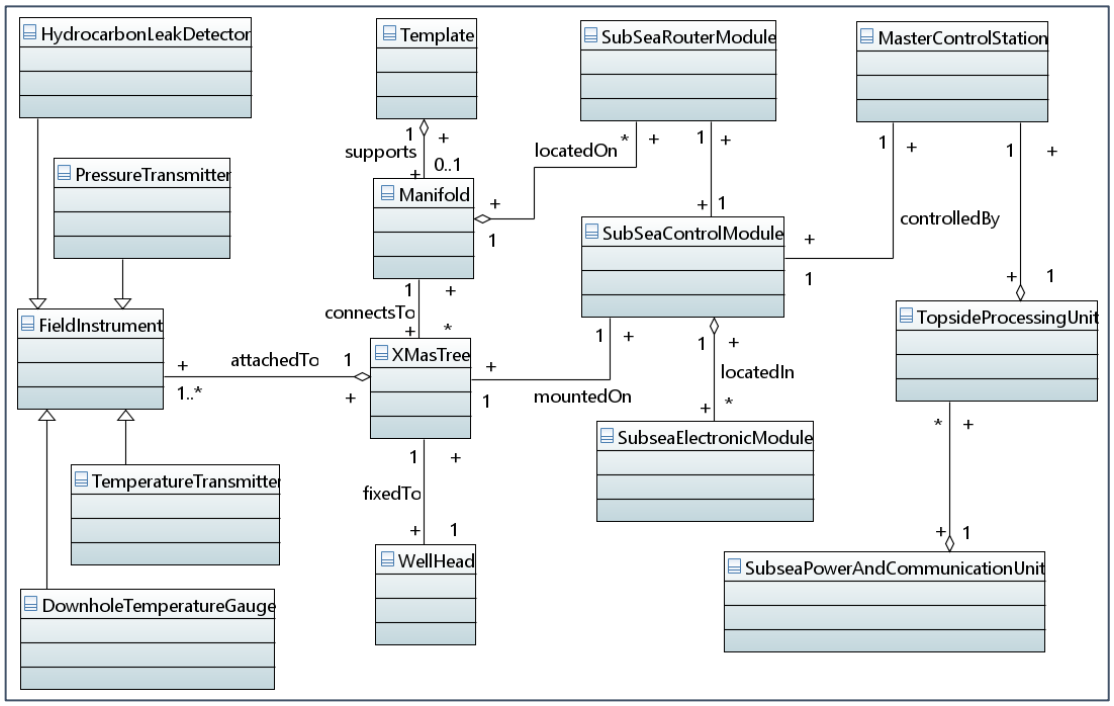

Ilustración 59: Un fragmento del modelo de dominio de un sistema de control de producción submarina.

Luego de generar el perfil UML, aplicamos los estereotipos definidos al modelo de dominio mostrado anteriormente (El proceso de aplicación del perfil UML será definido en la sección manual de usuario). La ilustración 60 muestra de forma jerárquica como queda definido el modelo de dominio luego de aplicar el perfil UML generado de forma automática.

De forma particular destacamos la *class HydrocarbonLeakDetector,* donde fue aplicado el estereotipo *Activity,* observando que fueron agregados las propiedades (*produces* y *requires*) del estereotipo a la clase del modelo de dominio.

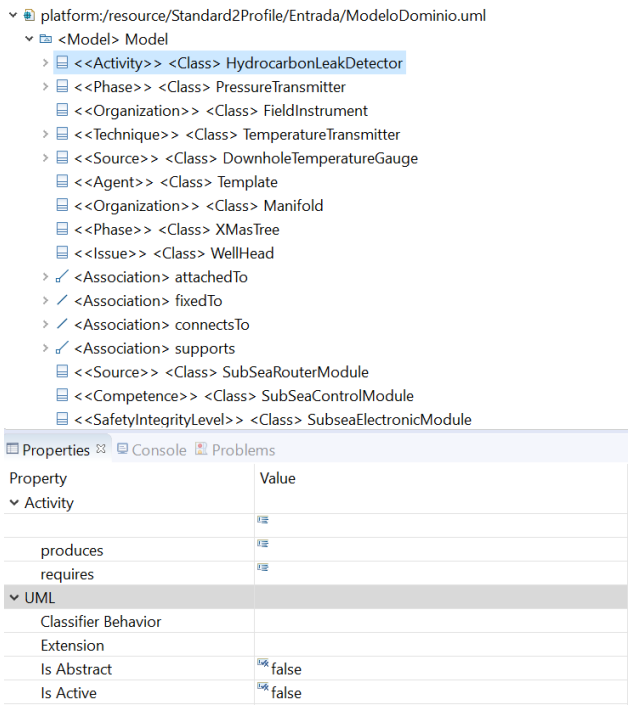

Ilustración 60: Modelo de dominio después de aplicar el perfil UML.

De manera concreta validamos la correcta definición del perfil UML generado a partir de las reglas ATL definidas en secciones anteriores, siendo aplicado de manera correcta a las clases del modelo de dominio.

Capítulo 5: Guía de Usuario

En esta sección se explica cómo debe ejecutarse la herramienta Eclipse, para generar el perfil desde el Modelo Conceptual del Estándar IEC61508, desarrollado en el presente proyecto.

Para realizar la transformación es necesario descargar e instalar la herramienta Eclipse. Se recomienda utilizar la versión Neon.1ª Release (4.6.1), es la empleada en este proyecto, y así evitar errores de compatibilidad. Además de instalar Eclipse es necesario agregar el plugin de modelado de modelos y que incluye todo lo relacionado con ATL.

Importar Proyecto

En el presente proyecto se adjunta el archivo comprimido correspondiente al proyecto ATL desarrollado en las secciones anteriores.

Antes de importar el proyecto, es necesario descomprimir el archivo Standard2Profile.zip, luego de ello se puede importar el proyecto ATL. Dentro del archivo se encuentra tres carpetas: Entrada, Perfil y Salida. La primera contiene los metamodelos utilizados en el presente proyecto, que corresponden al estándar IEC 61508 y un fragmento de este, correspondiente al paquete process concepts. La carpeta perfil contiene el modelo de perfil que es generado de forma manual para realizar el maping entre las clases y su respectiva metaclase. La carpeta salida contiene el modelo generado una vez aplicada la regla ATL (el nombre es generado en la configuración, el que aparece en la ilustración 63 es a modo de ejemplo). Y finalmente el archivo contiene la regla ATL (Standard2Profile.atl) que genera de forma automática el perfil UML. Ver ilustración 63.

Para abrir el proyecto es necesario seleccionar en el menú la opción “File” y luego seleccionar la opción “Open Projects from file systems” (ver ilustración 61).

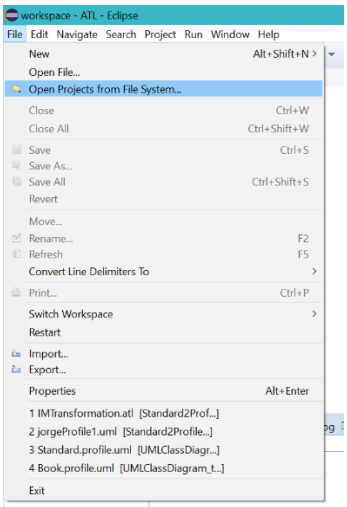

Ilustración 61: Abrir proyecto ATL.

A continuación, aparece una ventana “Import Projects From File System or Archive” y se debe seleccionar el botón “Directory”, luego debe buscar la ruta en donde esta descomprimido el proyecto ATL y presionar aceptar. Finalmente debe seleccionar “Finish” (ver ilustración 62) y el proyecto ATL está listo para ser ejecutado. (ver ilustración 63).

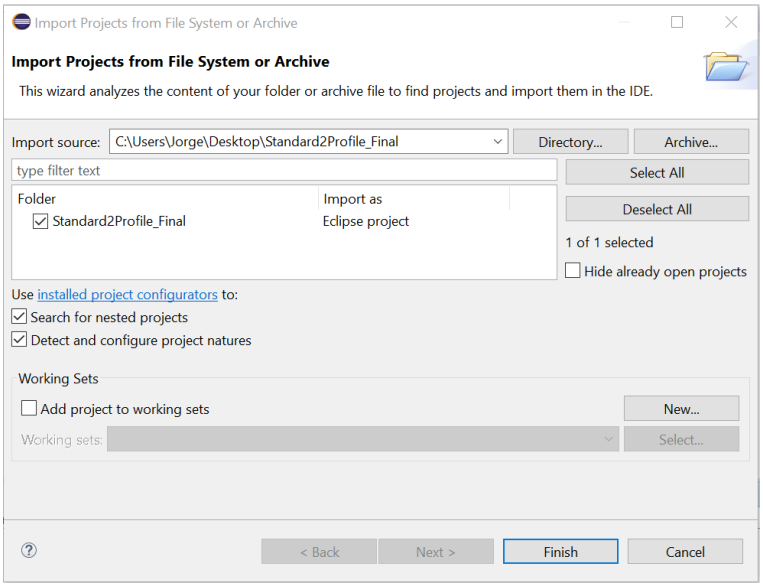

Ilustración 62: Ventana para importar proyectos.

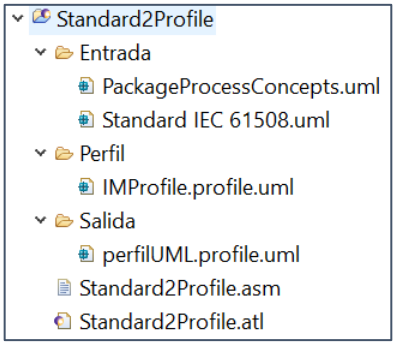

Ilustración 63: Proyecto listo para ser utilizado.

Una vez cargado el proyecto ATL, es necesario aplicar el perfil (IMProfile.profile.uml), para asignar la metaclase correspondiente a las diferentes clases del metamodelo que se esté utilizando como entrada. Este punto no será descrito, ya que fue realizado en secciones anteriores.

Luego de ello es necesario configurar la ejecución de la regla ATL (Standard2Profle). Para este caso es necesario posicionarse con el cursor dentro de la regla ATL, presionar el menú Run y seleccionar Run Configurations (ver ilustración 64).

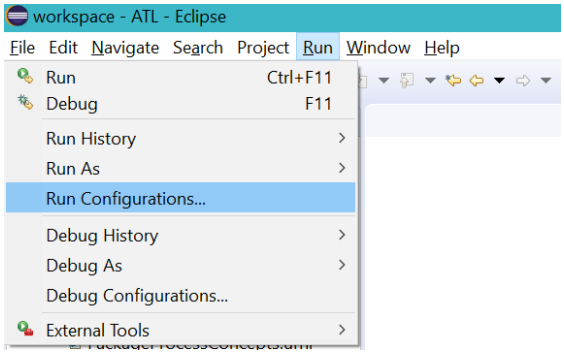

Ilustración 64: Acceso a configuración de ejecución de la regla ATL.

Aparece una ventana donde se debe configurar los diferentes parámetros para la ejecución de la regla ATL, en la pestaña ATL Configuration. Los parámetros son los siguientes:

- ATL Module: En este campo se debe indicar el archivo ATL que contiene la regla ATL.
- Metamodels UML: En este campo se debe indicar la versión UML que se utiliza para definir los diagramas de entrada y el perfil de salida. Para el caso de este proyecto corresponde a la versión 5.0.0.
- PRO. En este campo se debe indicar el perfil generado de forma manual para realizar el maping correspondiente.
- IN: En este campo se debe indicar el metamodelo de entrada que se desea generar el perfil de forma automática. En el proyecto corresponde a los modelos que están en la carpeta entrada, del proyecto ATL.
- OUT: En este campo se crea un archivo UML de salida, dentro de la carpeta salida y con la siguiente nomenclatura: nombre.profile.uml.

En las siguientes imágenes se muestra lo descrito anteriormente.

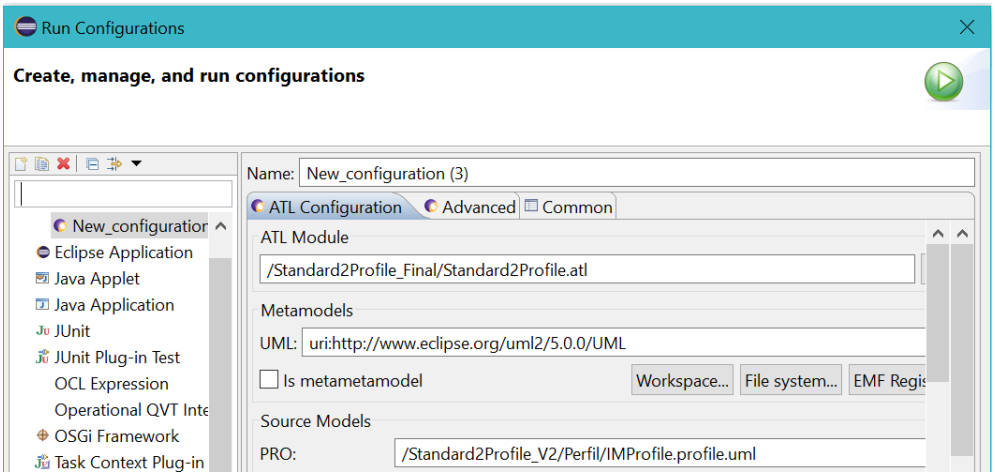

Ilustración 65: Run Configurations parte 1.

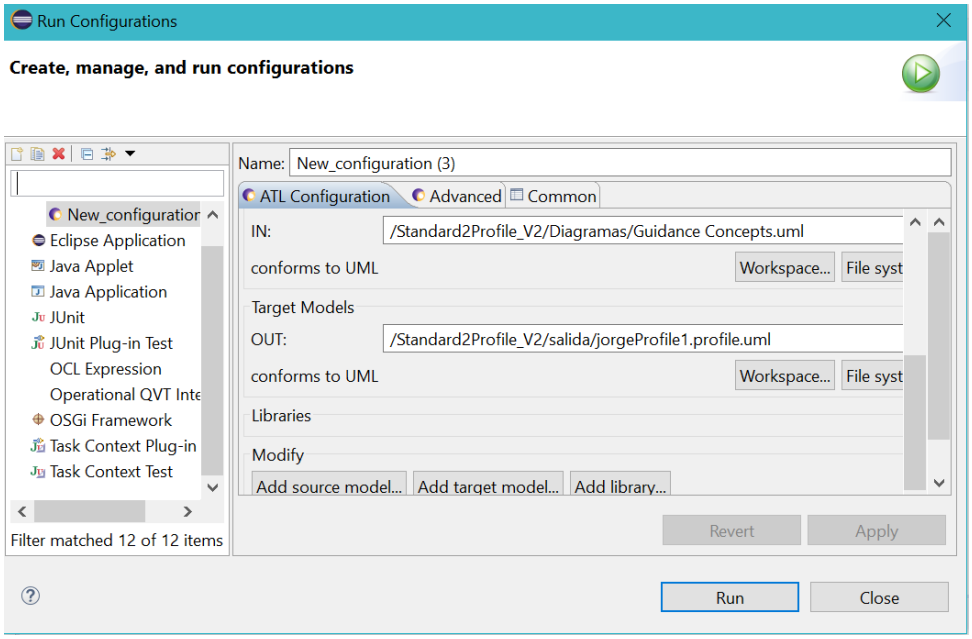

Ilustración 66: Run Configurations parte 2.

Una vez realizado la configuración, seleccionamos el botón Run y se ejecutan las reglas ATL. Una vez finalizado se genera el archivo de salida, con el perfil UML de acuerdo a los elementos que posee el diagrama de entrada seleccionado. El perfil se presenta de la siguiente manera.

Procedimiento para aplicar el perfil IMProfile

Para aplicar el perfil es necesario posicionarse en el nodo raíz del metamodelo de Safety, presionar mouse derecho, seleccionar y presionar *Load Resource*. Ver ilustración 67.

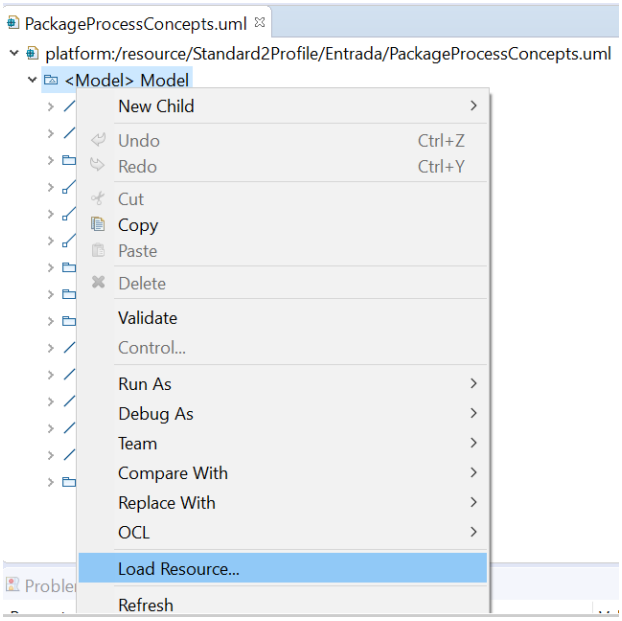

Ilustración 67: Procedimiento para cargar modelo.

Luego se selecciona el archivo fuente del perfil *IMProfile* que se desea aplicar, el cual está contenido en la carpeta que contiene el presente proyecto (Standard2Profile), y presionar OK para finalizar. Ver ilustración 68.

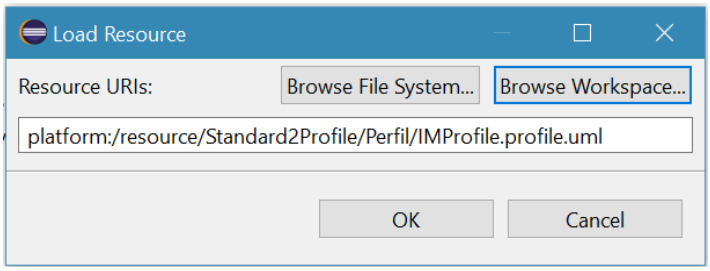

Ilustración 68: Cuadro que se carga la ruta del archivo que contiene al perfil.

Una vez realizado el procedimiento anterior, volver a posicionarse con el cursor en el nodo raíz del metamodelo de Safety y seleccionar el menú *UML Editor*, luego *package* y finalmente *Apply profile*. Ver ilustración 69.

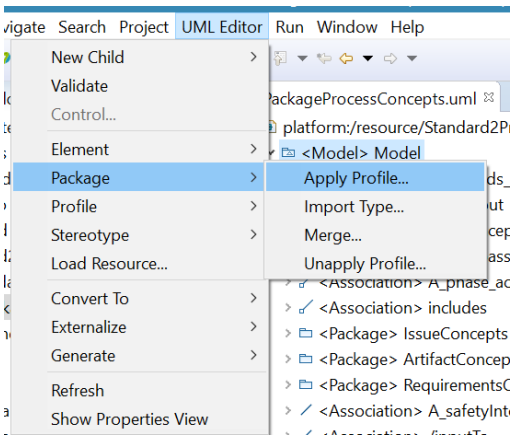

Ilustración 69: Ejemplo de procedimiento.

Luego aparece una ventana, donde se visualizan las alternativas de perfiles UML que se pueden aplicar, y dentro de ellos debemos identificar el perfil *IMProfile* seleccionándolo y presionar la tecla *apply*, trasladando el perfil a una nueva lista y para finalizar presionar OK. Ver ilustración 70.

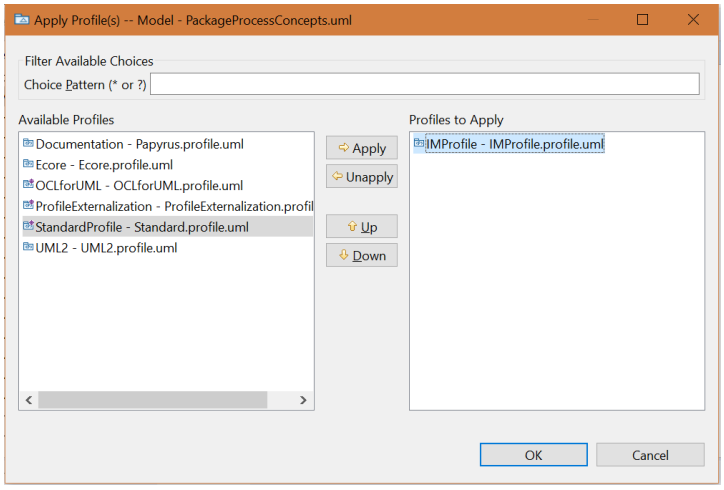

Ilustración 70: Ventana Apply Prrofile, para aplicar el perfil IMProfile.

Una vez aplicado el perfil al metamodelo de Safety, se agrega el estereotipo *MappingInfo*, permitiendo realizar el mapeo entre las clases del metamodelo de Safety y la *Metaclase Class* de UML, como muestra la siguiente ilustración.

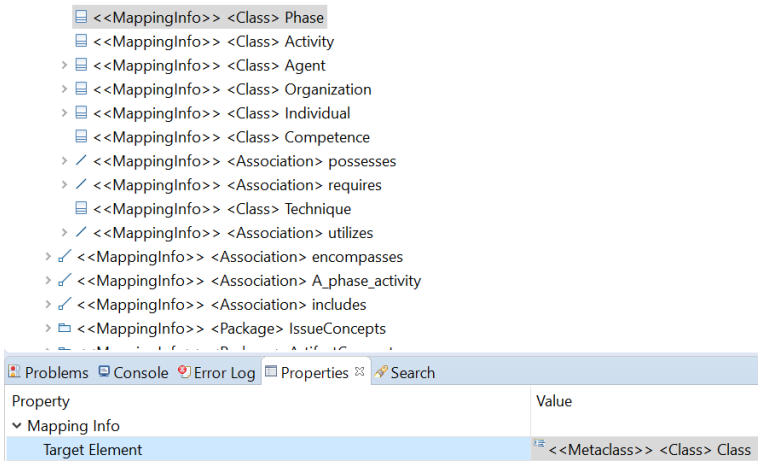

Ilustración 71: Metamodelo de entrada, una vez aplicado el perfil.

Ya finalizada la tarea de mapeo de todas las clases del metamodelo de Safety, se puede ejecutar la trasformación ATL, este procedimiento fue descrito en la sección anterior.

Procedimiento para aplicar el perfil UML al modelo de dominio

Para aplicar el perfil UML generado de forma automática al modelo de dominio es necesario antes definir el perfil UML. Se debe posicionarse en el nodo raíz, seleccionar el menú *UML Editor*, y finalmente seleccionar *profile* y define (ver ilustración 72). Se abre una nueva ventana donde se selecciona *OK* (ver ilustración 73). Con ello el perfil UML puede ser aplicado al modelo de dominio.

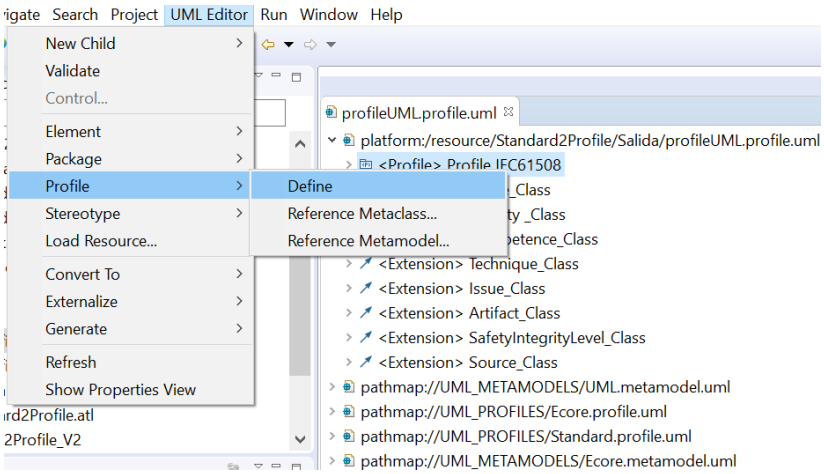

Ilustración 72: Definición del perfil UML.

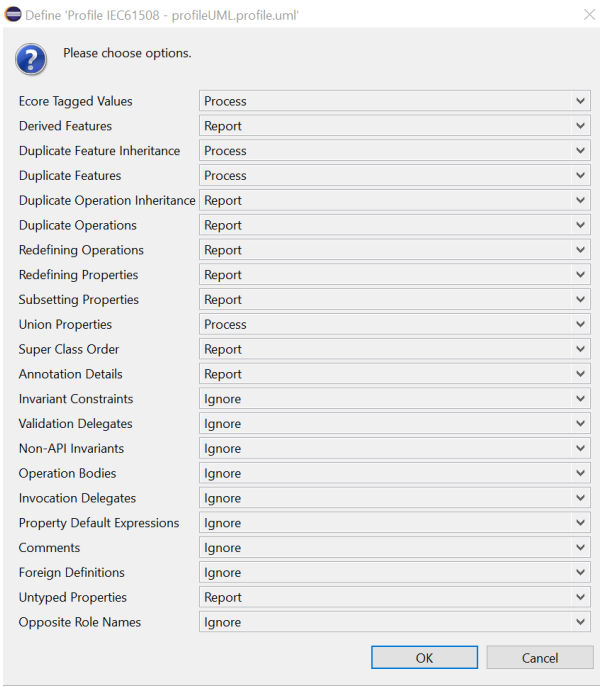

Ilustración 73: Definición del perfil UML.

El siguiente paso es posicionarse en el nodo raíz del modelo de dominio, para cargar el perfil UML definido en el paso anterior. Seleccionamos el menú *UML Editor* y luego *Load Resource* (ver ilustración 74), donde se abre nueva ventana donde debemos cargar el perfil UML. En la ventana *Load Resource* indicamos la ruta del perfil UML (ver ilustración 75).

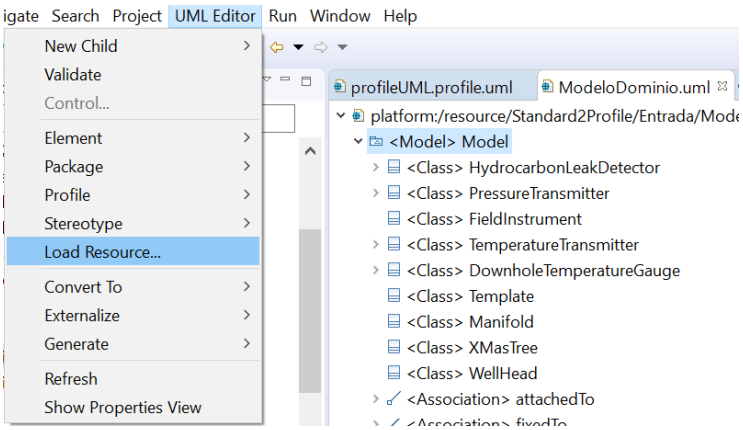

Ilustración 74: Procedimiento para cargar el perfil UML.

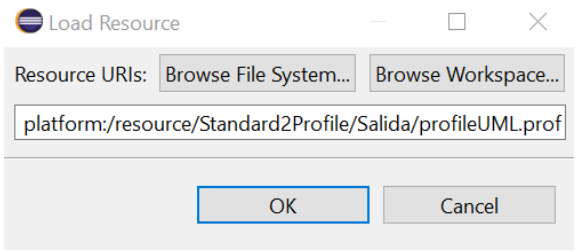

Ilustración 75: Ventana Load Resource, para cargar el perfil UML.

Un tercer paso, es nuevamente posicionarse en el nodo raíz del modelo de dominio para aplicar el perfil UML generado de forma automática. Seleccionamos el menú *UML Editor,* luego *Package* y *Apply Profile* (ver ilustración 76). Se abre la ventana *Apply Profile* (ilustración 77), donde se seleccionará el *Profile IEC 61508* y será aplicado presionando *OK*.

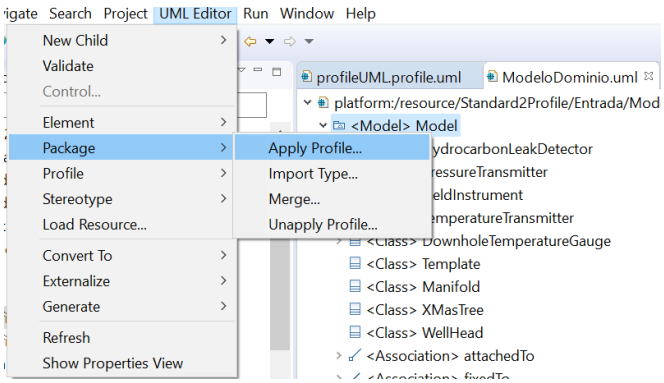

Ilustración 76: Procedimiento para aplicar el perfil UML al modelo de dominio.

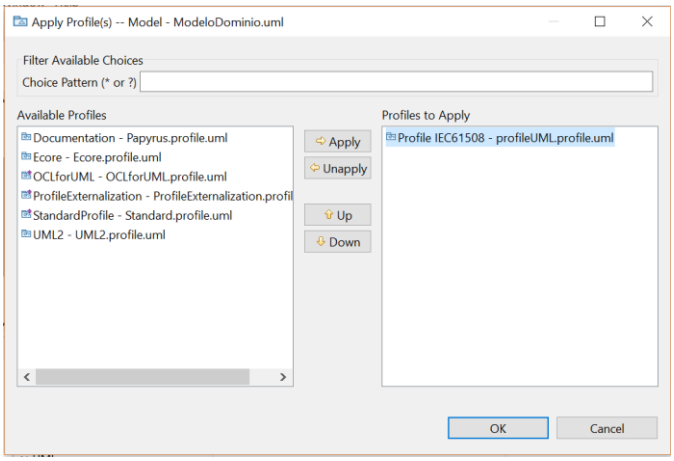

Ilustración 77: Procedimiento para aplicar el perfil UML al modelo de dominio.

Finalmente, el cuarto paso y ultimo corresponde aplicar los estereotipos definidos en el perfil UML en las clases definidas en el modelo de dominio según corresponda. Para ello debemos seleccionar un elemento *class* en el modelo de dominio, luego seleccionar el menú *UML editor*, *Element* y *Apply Stereotype (ver ilustración 78).* Se abre la venta *Apply Stereotype* (ilustración 79), donde se debe seleccionar el estereotip que corresponda a la clase seleccionada y para finalizar seleccionar *OK.*

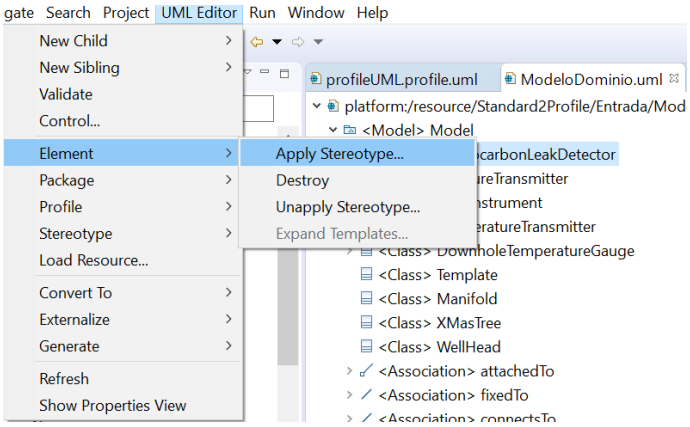

Ilustración 78:Procedimiento para aplicar los estereotipos en las clases del modelo de dominio.

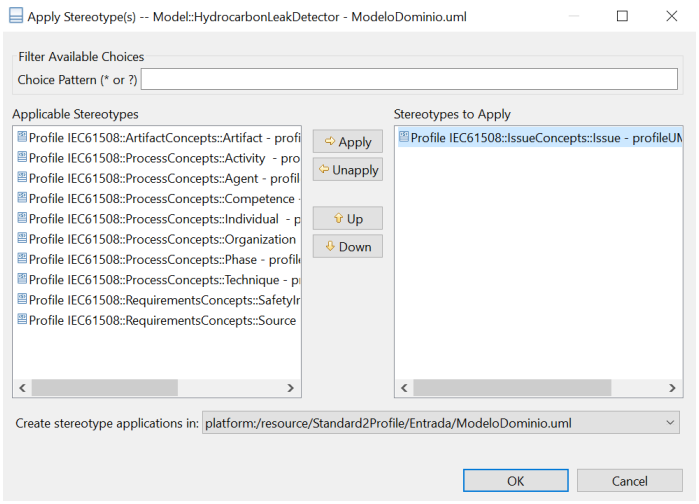

Ilustración 79: Procedimiento para aplicar los estereotipos en las clases del modelo de dominio.

Una vez aplicado los estereotipos en la totalidad de las clases presentes en el modelo de dominio, se deben guardar los cambios y se finalizara el proceso.

Pruebas de Rendimiento

De acuerdo a los objetivos definidos en un comienzo la herramienta desarrollada debe generar el perfil de forma correcta en su totalidad de acuerdo al modelo conceptual utilizado como entrada. Por lo tanto, la eficiencia de la herramienta es algo que está en segundo plano.

La herramienta de generación automática de perfiles se ejecutará cuando el usuario lo estime conveniente, o bien, cuando sea necesario obtener un perfil nuevo a partir del modelo conceptual del estándar aplicado a otro dominio en particular.

Las pruebas se ejecutaron en un equipo con las siguientes características:

- Procesador Intel i5, con frecuencia de reloj 1.50 Ghz
- Ram: 4 GB
- Tipo de sistema: 64 bits

El tiempo que tarda en generarse el perfil dependerá del tamaño del modelo conceptual utilizado como entrada.

Capítulo 6: Conclusiones y Trabajo Futuro

En el presente trabajo se ha presentado una propuesta para la generación automática de perfiles UML orientados a modelos de Safety. Esta propuesta utiliza una serie de reglas de trasformación aplicadas sobre un metamodelo de Safety de entrada más información de mapeo que permite determinar las metaclases UML que deben ser extendidas, así como las extensiones que debe incorporar cada estereotipo del perfil generado. Tanto la definición del metamodelo de entrada como el mapeo con UML se han implementado mediante modelos de clases de UML y perfiles UML. La utilización de UML para la definición del metamodelo de Safety y definición de información de mapeo facilita la aplicación de la propuesta ya que no requiere la utilización de editores específicos, nuevas herramientas para mapeo de modelos, o conocer nuevos lenguajes de modelado. La propuesta se ha ejemplificado en la trasformación de un metamodelo de Safety y perfil UML bastante conocido en la literatura. Al comparar resultados se pudo apreciar que existen algunos problemas de definición conceptual en el perfil generado manualmente. En cambio, las reglas de transformación encapsulan decisiones de definición alineadas con el estándar UML que garantizan la validez sintáctica del perfil generado, al mismo tiempo que aseguran la completitud del perfil en relación al metamodelo UML de entrada. Otra contribución relevante es que el mapeo para la identificación de extensiones no requiere conocimientos sobre el diseño de perfiles UML. Esto es debido a que la identificación de las extensiones se basa en el reconocimiento de diferencias entre el metamodelo de Safety y el metamodelo de UML. Al ser el mapeo explícito e independiente de la definición del perfil UML, su validación por parte de terceros se simplifica, permitiendo un mejor análisis de las equivalencias entre los conceptos del metamodelo de Safety con el metamodelo de UML. Como trabajo futuro consideramos realizar evaluaciones con nuevos metamodelos de Safety

de entrada para refinar la implementación de las reglas de transformación. Además, se planea el desarrollo de mecanismos que faciliten el mapeo entre metamodelos, y su validación para garantizar la correcta generación del perfil final.

Capítulo 7: Bibliografía

Biggs, G., Sakamoto, T., & Kotoku, T. (2016). A profile and tool for modelling safety information with design information in SysML. *Software & Systems Modeling, 15*(1), 147-178.

de la Vara, J. L., Ruiz, A., Attwood, K., Espinoza, H., Panesar-Walawege, R. K., López, Á., . . . Kelly, T. (2016). Model-based specification of safety compliance needs for critical systems: A holistic generic metamodel. *Information and Software Technology, 72*, 16-30.

Giachetti, G., Marín, B., & Pastor, O. (2009). *Using uml as a domain-specific modeling language: A proposal for automatic generation of uml profiles.* Paper presented at the International Conference on Advanced Information Systems Engineering.

Group, O. M. (1997-2016). OMG Formal Versions of UML. Retrieved from http://www.omg.org/spec/UML/

International Electrotechnical, C. (2000). Functional safety of electrical/electronic/programmable electronic safety related systems. *IEC 61508*.

Knight, J. C. (2002). *Safety critical systems: challenges and directions.* Paper presented at the Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on.

Kuschnerus, D., Bruns, F., Bilgic, A., & Musch, T. (2012). *A UML profile for the development of IEC 61508 compliant embedded software.* Paper presented at the Proceedings of the 6th International Congress and Exhibition—Embedded Real Time Software and Systems, ERTS2 2012.

Leitner-Fischer, F., & Leue, S. (2011). Quantitative analysis of UML models. *Proceedings of Modellbasierte Entwicklung eingebetteter Systeme (MBEES 2011). Dagstuhl, Germany, 27*.

Lin, C.-L., Shen, W., & Kountanis, D. (2013). *Using UML profile and OCL to impose regulatory requirements on safety-critical system.* Paper

presented at the Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2013 14th ACIS International Conference on.

Lionel Briand, T. C., Shiva Nejati, Rajwinder Panesar-Walawege, Mehrdad Sabetzadeh. (2009). Characterizing the Chain of Evidence for Software Safety Cases: A Conceptual Model Based on the IEC 61508 Standard (Technical Report). *Simula Research Laboratory and Det Norske Veritas*, 46.

Panesar-Walawege, R. K., Sabetzadeh, M., & Briand, L. (2011a). *A model-driven engineering approach to support the verification of compliance to safety standards.* Paper presented at the 2011 IEEE 22nd International Symposium on Software Reliability Engineering.

Panesar-Walawege, R. K., Sabetzadeh, M., & Briand, L. (2011b). *Using UML profiles for sector-specific tailoring of safety evidence information.* Paper presented at the International Conference on Conceptual Modeling.

Panesar-Walawege, R. K., Sabetzadeh, M., & Briand, L. (2013). Supporting the verification of compliance to safety standards via model-driven engineering: Approach, tool-support and empirical validation. *Information and Software Technology, 55*(5), 836-864.

Panesar-Walawege, R. K., Sabetzadeh, M., Briand, L., & Coq, T. (2010). *Characterizing the chain of evidence for software safety cases: A conceptual model based on the IEC 61508 standard.* Paper presented at the 2010 Third International Conference on Software Testing, Verification and Validation.

Ritala, T., & Kuikka, S. (2007). *UML automation profile: enhancing the efficiency of software development in the automation industry.* Paper presented at the Industrial Informatics, 2007 5th IEEE International Conference on.

Schäfer, W., & Wehrheim, H. (2010). Model-driven development with mechatronic uml *Graph transformations and model-driven engineering* (pp. 533-554): Springer.

Smith, D. J., & Simpson, K. G. (2016). *The Safety Critical Systems Handbook: A Straightforward Guide to Functional Safety: IEC 61508 (2010 Edition), IEC 61511 (2015 Edition) and Related Guidance*: Butterworth-Heinemann.

Zoughbi, G., Briand, L., & Labiche, Y. (2011). Modeling safety and airworthiness (RTCA DO-178B) information: conceptual model and UML profile. *Software & Systems Modeling, 10*(3), 337-367.