



# TRATAMIENTO DE DATOS DE INSTALACIONES DE CONTROL DE TRENES MEDIANTE RAILML.

Trabajo de Fin de Grado

Alberto Burguillo Ruiz

Tutor:

Juan de Dios Sanz Bobi

Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros Industriales

Noviembre 2018

# TRATAMIENTO DE DATOS DE INSTALACIONES DE CONTROL DE TRENES MEDIANTE RAILML

Alberto Burguillo Ruiz 07045

Universidad Politécnica de Madrid - E.T.S.I. Industriales
Noviembre 2018

*Para todas aquellas personas que me han apoyado y ayudado durante esta andadura. Para mis padres y mis hermanas por estar siempre ahí y su ayuda y paciencia, para Inés, que saca lo mejor de mí, y para Dani, por acompañarme en cada uno de los pasos y de los tropiezos durante estos años (y los que quedan), y para LFDC (Ulli, Lara, Juan, Jose, María, Juanan, Jaira y Mawel) por ser tan geniales compañeros de viaje. Gracias a Juan de Dios por facilitar la cosecución de este TFG.*

# Contenido

# 1 Introducción

## 1.1 Panorama actual

El tráfico ferroviario es una de las formas más comunes y extendidas de movimiento de mercancías y personas en el mundo. Si bien el porcentaje de viajes de pasajeros en tren representa un 8% del total[1] en Europa, el uso del ferrocarril para transporte de mercancías puede alcanzar hasta el 25% dependiendo del país.

También es sabido que, el control ferroviario es una tarea con un alto grado de complicación y que necesita de aptitudes y, sobre todo, información a un alto nivel de detalle y con mucha precisión. Sin embargo, a día de hoy, esa información se comparte por medios impresos o en formatos que no permiten ni la edición ni la obtención de dichos datos en un formato de más utilidad de una manera rápida y eficiente. Esto hace preciso el comenzar a pensar en la posibilidad de la digitalización de los datos referentes al control ferroviario, así como sus estructuras y formaciones. Por este motivo, han surgido iniciativas como RailML®, igual que otras iniciativas que intentan hacer posible ese paso hacia la digitalización del dato y su posible tratamiento.

Actualmente, una gran mayoría de los mercados están digitalizados y con ello consiguen ventajas competitivas de bastante calado, como son el análisis de datos para poder mejorar las ofertas dadas a clientes, así como la posibilidad de adecuar el producto o servicio a las necesidades y gustos de los consumidores por medio de análisis masivo de datos como son las tecnologías de Big Data.

## 1.2 Necesidad de unificación

Como bien es sabido, la estructura de la Unión Europea facilita el tránsito de personas y mercancías entre los diferentes países que forman parte de la misma. Además, normas europeas desde la Directiva 1991/440/CEE, hasta la reciente incorporación del cuarto paquete de liberalización ferroviaria, hacen que la comunicación entre trenes, vías y estaciones de control o enclavamientos tenga que ser más clara y sencilla cada día. Esto implica varios problemas, entre los que se encuentra el obvio del lenguaje, así como la dificultad de compaginar sistemas de control diferentes. Así, se intenta apuntar a que, dentro de lo posible, los sistemas de control y guardado de información referente al tráfico ferroviario, sean reconocibles a nivel europeo e incluso mundial. Es decir, del mismo modo que se necesitó en su día un cambiador de ancho para facilitar el tránsito, se necesita un "cambiador de sistemas" para hacer más sencillo el intercambio de datos. De este modo, se puede entender que RailML® viene a hacer la labor de ese cambiador, en cuanto al control y organización se refiere. Para ello hace falta, entre otras cosas, la digitalización de los datos de la que se habla en el punto

[1] https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Passenger_transport_statistics/es#Estad.C3.ADsticas_de_transporte_de_viajeros

anterior. Esto significa que RailML® no es el primer paso, sino un puente para conseguir que dicho primer paso se dé en todos los puntos en la misma dirección.

Como es obvio, cada país, geografía e incluso cada empresa puede expresar de maneras diferentes todo aquello que no esté normalizado. Sin embargo, el código de RailML® es único, y comprensible por cualquier usuario.

## 1.3 Digitalización

Es de sobra conocida la versatilidad y facilidad de uso, transmisión y modificación de los datos en un formato digital. Donde un cambio de una vía (por ejemplo el cierre de una parte, el cruce con una nueva carretera o la creación de una vía paralela) obliga a rehacer todos los datos no digitales, con la consecuente carga de trabajo que ellos conlleva (generación de planos, anexos, y demás documentación). Sin embargo, esa misma modificación sería mucho más ligera si tras modificar los datos (que puede hacerse incluso en un fichero aparte), se pueda generar esos documentos de manera procesal desde un archivo RailML®. Del mismo modo, gracias a las posibilidades que da un código como RailML®, sería posible incluso una automatización sencilla que, tras cambiar diez o veinte líneas de código, sea capaz de generar cualquier tipo de documentación necesaria.

El avance de la tecnología y el manejo de datos en cantidades masivas avanza a una velocidad vertiginosa. Cualquier tipo de dato que tenga un registro digital, puede ser susceptible de ser analizado y estudiado a tiempo real, pudiendo sacar rédito de ello por medio de adaptación de servicios, métodos de venta y mercados, publicidad, y una larga lista de posibilidades que se abren con tecnologías como Big Data. Esto hace posible que se pueda incluir en un futuro en este código valores referentes, por ejemplo, a hábitos de uso y perfiles de usuario, que permitan proveer un mejor servicio, o minimizar los gastos en ciertas ocasiones. Para todo esto, obviamente, el primer paso es la digitalización de la información de las redes ferroviarias y sus usos, así como su material rodante y, todo esto, es posible con RailML®.

De la misma manera, se abre la posibilidad de inclusión de sistemas y servicios como las APIs (Application Programming Interface) que son interfaces que (entre otras cualidades) facilitan enormemente la elaboración de consultas sobre el estado de un archivo en particular. Esto hace que cualquier plataforma pueda obtener información de manera simple y rápida de los archivos digitales que estén expuestos a Internet. Esto sería de especial interés a la hora de compartir infraestructura, horarios e, incluso, venta de tickets con sistemas de terceros.

Por supuesto, como cualquier proceso de este tipo, conlleva una gran inversión inicial y un plazo de desarrollo que es largo y costoso. Sin embargo, cada vez más, estos medios ganan importancia e interés, encaminando estos controles a ser cada vez más digitales. También es de mención que, con el crecimiento del interés que despierta este tipo de datos digitales y su tratamiento, cada vez es más y más sencilla la generación de archivos de este tipo, así como su tratamiento y desarrollo. Actualmente, existe una infinidad de herramientas de tratamiento de lenguajes XML y, este, es solo un ejemplo de la variedad de ayudas que se pueden utilizar cuando se enfrenta la digitalización de un sistema tan complejo como el de control y supervisión ferroviarios.

Es lógico que todo lo que ocurre a nivel empresarial tiene un sentido y un objetivo, y la tendencia a la digitalización no es menos. Esta tendencia es debida principalmente a la ventaja competitiva que ofrece la rapidez del tratamiento de datos digitalizados, así como por el ahorro económico que ofrece, una vez dicha digitalización está completa por su nulo coste de mantenimiento, y su capacidad de adaptación en cortos periodos de tiempo. Es cierto, por otro lado, que esta digitalización tiene un coste, más importante al principio de la vida de ésta, que puede suponer un impedimento lo suficientemente grande como para que una empresa u organización limite su uso o, incluso, lo deseche.

## 1.4 Punto de partida

Para el caso práctico que se ha desarrollado, como primer punto, se tiene una vía existente que, para la evolución del proyecto, carece de importancia. Lo que sí es importante destacar es que esta vía reúne los requisitos mínimos que se han puesto para que la misma sea un ejemplo lo suficientemente amplio para la generalización del modelado de vías a partir del modelado de esta vía.

La vía en cuestión es una vía de la red española situada en la entrada de Asturias desde León. Contiene curvas, cambios de pendiente, señales, balizas, estaciones y desvíos de varios tipos. También, su extensión y recorrido hacen que la vía sea amplia y ofrezca la posibilidad de definir diferentes tipos de modelado durante la misma con el objetivo de ilustrar las diferencias que estos modelados generarían en los resultados.

Como material rodante, se ha elegido el tipo de locomotora por simplicidad y practicidad, respondiendo a criterios de disponibilidad de datos. Así, la locomotora elegida ha sido de la cual se disponía de más información. Los vagones, sin embargo, se han definido como vagones genéricos sin identificaciones específicas más allá de las necesarias para el desarrollo del estudio (como son la capacidad de frenado, o el número de plazas disponibles).

Por otro lado, RailML® es un sistema nuevo (de hecho, aún en desarrollo) lo que supone que cualquier o modelado llevado a cabo con esta herramienta supone un cambio en el sistema actual, de gran profundidad y amplio calado. Este cambio añade una variable de innovación que finalmente, da sentido a este trabajo con un estudio de viabilidad de implantación de este sistema en las redes ferroviarias. Asimismo, con este primer acercamiento (sobre todo su parte práctica) se da pie al desarrollo de proyectos de transformación digital, y de análisis de posibilidades que RailML® ofrece.

# 2 RailML® concepto y definición

RailML® es un formato de intercambio de datos (metalenguaje) de código abierto basado en XML (siglas de eXtensible Markup Lenguage). Esto es, un formato que se utiliza para organizar y etiquetar el contenido de documentos y, por ello, guardar y clasificar información según se requiera en cada momento. También facilita la obtención de datos específicos y el rápido acceso a los mismos, puesto que esos datos siempre van a estar guardados bajo una misma estructura.

Se debe aclarar que esto no es un lenguaje de programación, dado que no sirve para generar una lógica o una ejecución de un programa, sino para guardar y organizar datos; y que, aunque en ocasiones se le llame lenguaje o metalenguaje, tampoco lo es como tal, es simplemente una definición hecha a partir del lenguaje XML.

La intención de la aparición de RailML® surge de la necesidad de unificar la manera de almacenar, distribuir y compartir la información relativa a la organización ferroviaria entre herramientas heterogéneas. Con RailML®, se consigue reducir la cantidad de intermediarios que se necesitan para que se entienda la configuración de una vía, o los tipos de trenes que pasan por ellas, sin ninguna necesidad de traducción o adaptación.

Supuesto que se utilice una aplicación diferente para cada sistema (o red), se necesitará una interfaz entre cada pareja de sistemas. Sin embargo, con la unificación que provee RailML®, se consigue reducir notablemente la cantidad de interfaces que hacen falta para la comprensión entre diferentes sistemas, y allanando el camino hacia una estandarización global.

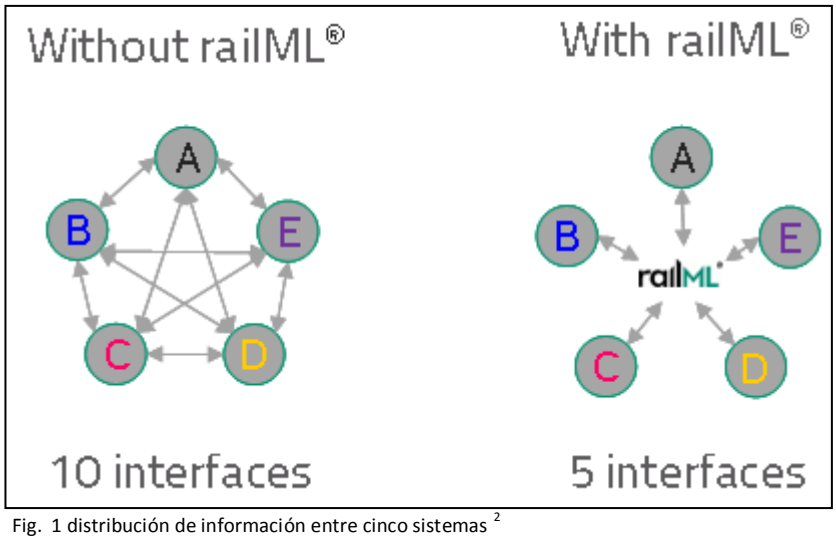


Fig. 1 distribución de información entre cinco sistemas [2]

Este lenguaje está todavía en desarrollo y, por tanto, se debe estudiar con cierta precaución puesto que es susceptible de sufrir cambios. Asimismo es importante destacar que es desarrollado por un conjunto de partícipes entre los que se encuentran empresas importantes del mundo ferroviario.

[2] https://www.railml.org/en/introduction/background.html

# 3 Objetivos

## 3.1 Objetivos del Trabajo

La base de este trabajo de Fin de Grado, es sin duda RailML®. Sin embargo, el estado de este metalenguaje no es estable al completo, y no está en una fase final (aunque es totalmente productiva) dado que, según las necesidades que se van encontrando a medida que se usa, se pueden cambiar algunas de las características inherentes al lenguaje y ello provoca que sea aún un lenguaje en proceso de desarrollo.

Por ello, el objetivo principal de este trabajo es analizar y estudiar el potencial de RailML® en aplicaciones de control ferroviario, así como sus limitaciones y sus necesidades. Esto desembocará, como es lógico, en una aproximación a sus utilidades y su forma de uso.

Además, el segundo objetivo de mayor calado será el de elaborar un modelo práctico de una red ferroviaria utilizando RailML® como herramienta de modelado.

De la misma manera, como último objetivo se tiene validación del modelo práctico en base a la respuesta obtenido del software railVIVID®.

Obviamente, puesto que este trabajo consiste principalmente en la definición de una estructura ferroviaria en RailML®, se necesitará crear o encontrar una definición de las vías que haga posible la consecución satisfactoria del trabajo. Esto pasa por tener una base de concepto y de simbología necesaria para poder comprender una línea ferroviaria analizada. Obviamente, las características de la línea no son de importancia en el estudio de este trabajo, puesto que esta herramienta se debe poder aplicar a cualquier línea sin restricción alguna. Sin embargo, sí se debe aclarar que la línea contiene todos los elemento necesarios para hacer una amplia definición de la misma por medio de RailML®. Se ha intentado seleccionar una línea que abarque la suficiente amplitud para comprender elementos variados y que permita ejemplificar de una manera útil las posibilidades que RailML® brinda para su caracterización.

## 3.2 Objetivos académicos con RailML®

En cuanto al aspecto académico de este estudio, la elaboración de esta memoria constituirá una de las primeras guías de uso y buenas prácticas sobre RailML, cuya difusión académica persigue el objetivo de servir como base inicial a futuros estudios sobre las aplicaciones de la herramienta en el ámbito de la gestión ferroviaria y para permitir una integración más rápida y más homogénea de los próximos modelos de material ferroviario. De este modo se pretende, también generar una base de partida sobre la cual se pueda construir un estudio profundo y una adaptación del lenguaje a la red de uso, y viceversa. Esto incluirá futuros proyectos que irán siendo necesarios a medida que se avance y profundice en el tema.

Es necesario también tener una perspectiva razonable del uso de XML y sus aplicaciones, así como de sus normas de uso, de modo que un objetivo razonable es el análisis del lenguaje RailML® desde un punto de vista informático, con el que se pueda observar la solidez y robustez de los archivos que se puedan generar.

Análisis de RailML®, posibilidades de implantación e impacto.

Asimismo, sobre la base de lo aprendido a lo largo de la especialidad de mecánica, se hace necesario profundizar en la lectura de planos de vías, y otros materiales ferroviarios que son de necesario análisis a la hora de trasladar toda su información a un sistema digital, como es el que se está estudiando.

Puesto que no existe ninguna red con sus sistemas de control completamente digitalizados, se considera que el estudio de estas posibilidades implica una base de innovación en el campo ferroviario que puede dar pie a la ejecución de varias mejoras y más resultados de investigación tales como la aplicación de nuevas tecnologías de tratamiento masivo de datos, intercambio de información a tiempo real, o la inclusión de variantes de esta tecnología a bordo de los vehículos.

## 3.3 Objetivos personales

En el ámbito personal, el principal objetivo es lograr establecer una dinámica efectiva de trabajo de aprendizaje, investigación y desarrollo de trabajo que permita desarrollar este trabajo de manera eficaz y efectiva. Del mismo modo, hay un objetivo primordial que es el entendimiento de cómo se llevan a cabo los proyectos de este tipo.

Al ser esta, una aplicación real de lo aprendido académicamente, un gran objetivo personal es conseguir sentar las bases de una buena gestión de proyecto que posibilite la consecución real, en tiempos, y satisfactoria del mismo. Esto implica desde una planificación inicial, hasta una corrección de riesgos temporales si los hubiese.

Es también muy importante destacar que este proyecto tendrá impacto en el ámbito profesional. Por ello, es vital conseguir una visión empresarial de cómo esta herramienta puede afectar a los intereses de una empresa, así como a la red ferroviaria en general.

# 4 Marco teórico - estudio de RailML®

## 4.1 Generalidades

La estructura de RailML® se divide en cuatro subestructuras principales que se analizarán más adelante. Por debajo de cada subestructura (o subesquema), se encuentran los elementos que son los principales portadores de información y, por último, los atributos. Los elementos pueden estar anidados y/o repetidos tantas veces como sea necesario siempre que se cumpla con lo que indican los esquemas de RailML®. Algunos elementos son obligatorios y otros son opcionales, dependiendo de su importancia en el archivo. Del mismo modo, algunos atributos también son obligatorios mientras que otros serán opcionales. Estos atributos, en cambio no pueden ser repetidos dentro de un mismo elemento (como se verá más adelante, es trivial pensar que un elemento no puede tener dos atributos de la misma clase con valores diferentes). Un ejemplo muy claro de esta última puntualización es la posición de un elemento; este no puede estar en dos lugares a la vez.

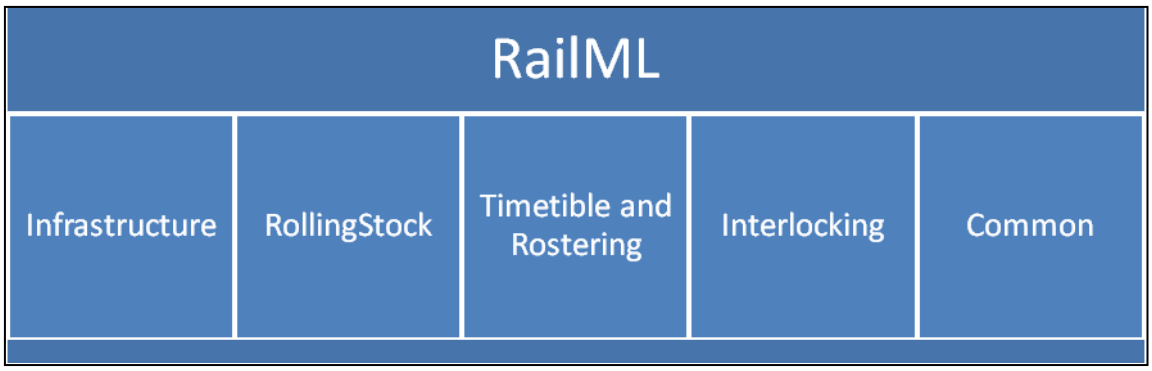


Fig. 2: Esquemas de RailML®

Los cuatro subesquemas principales son los dedicados a Infraestructura, Material Rodante, Horarios, Bloqueos y Elementos comunes. Cada uno de ellos se puede presentar por separado y sin los demás, aunque esta práctica es desaconsejable. Dada la gran cantidad de datos que los subesquemas pueden contener, cada uno de ellos puede llegar a tener una cantidad considerable de líneas de código. Sin embargo, gracias a la simplicidad del lenguaje, estos son analizados rápidamente y pueden ser trabajados prácticamente a tiempo real.

Cada uno de los subesquemas, como es típico de este tipo de lenguajes, debe seguir una estructura predeterminada, que es indicada al comienzo de cada archivo RailML®. Esta estructura cambia según la versión usada. El no cumplimiento de este esquema puede repercutir en fallos a la hora de leer los archivos RailML®.

Para entender un poco mejor cómo funciona este tipo de archivos, es necesario tener un mínimo de conocimientos sobre el lenguaje XML. Todo lo referente a la estructura de este lenguaje es, de hecho, aplicable a la estructura del lenguaje RailML®. Lo más simple para comprender la estructura de XML es saber que esta se genera en forma de árbol, esto es, con elementos anidados unos dentro de otros. El primero de estos elementos es el elemento Raíz (Root element) del que pueden irse añadiendo ramas. Al final de estas ramas, se encontrarán los elementos hijo (child element). Por último, cada uno de los elementos (bien sean elementos hijo o elementos padre) puede tener tantos atributos como se necesite. Estos atributos, junto con el

texto, son los que definen cómo es el elemento. Cada elemento puede contener texto, atributos, más elementos, una mezcla de los anteriores o estar vacíos. Como último elemento notable, en XML se pueden añadir comentarios (tantos como se necesite) para asegurar la correcta comprensión del código, sea quien sea el lector. Esto es, además, de especial utilidad a la hora de escribir dichos códigos.

Como cualquier lenguaje, XML tiene ciertas reglas de sintaxis que deben ser cumplidas. Las más interesantes son las siguientes:

- Cualquier elemento debe estar anidado dentro de un elemento Raíz
- Un archivo de XML puede tener una primera línea (prolog) que se usará para definir la versión del lenguaje utilizada, así como la codificación de caracteres
- XML es sensible a mayúsculas, es decir, <speedChange> no será el mismo elemento que <Speedchange>, ni <speedchange>. Esto es un aspecto muy a tener en cuenta dado que el resultado del modelo de una vía son varios miles de líneas. Por ello es recomendable marcar unas normas de nomenclatura desde el principio
- Los elementos tienen que tener una apertura (<) y un cierre (/>)
- Los atributos deben ser definidos entre comillas dobles o simples

Del mismo modo, los elementos deben cumplir ciertas normas de sintaxis como:

- Comenzar por una letra o un guión bajo
- Contener letras, dígitos, guiones, guiones bajos o puntos
- No contener espacios
- Finalizarse siempre con una barra "/"

De acuerdo con lo anterior, un ejemplo de datos de RailML® podría ser el siguiente:

```
<track id='tr_AB132' code='códigoDeEjemplo' name='nombre ' type='sidingTrack'>
        <trackTopology>
                <connections>
                        <switch id='swi_rob_7' code='swi_rob_7' pos='95'  absPos='25036'>
                                <connection id='' ref='conect_id1' maxSpeed='50' />
                        </switch>
                </connections>
                <trackBegin pos='0'  absPos='12345' id='CM_AB132'><connection
                id='conBeg_10123' ref='con_rob_3'/></trackBegin>
                <trackEnd pos='712'  absPos='25653' id='ten_10123'><connection
                id='conEnd_10123' ref='con_rob_8'/></trackEnd>
        </trackTopology>
        <trackElements> (...)
        </trackElements>
</track>
```

Como se puede observar en este ejemplo, el elemento <connection (..)/> se ha cerrado con una barra dentro de los mismos corchetes. Sin embargo, el elemento <trackTopology> necesita una sentencia de apertura y otra de cierre. Esto se debe únicamente a que los elementos que no contienen subelementos pueden (y es una praxis recomendable) ser cerrados en la misma sentencia.

Lo visto anteriormente se puede resumir y explicar con la siguiente tabla, incluyendo entre paréntesis los atributos y siguiendo una estructura como la que se dibuja a continuación:

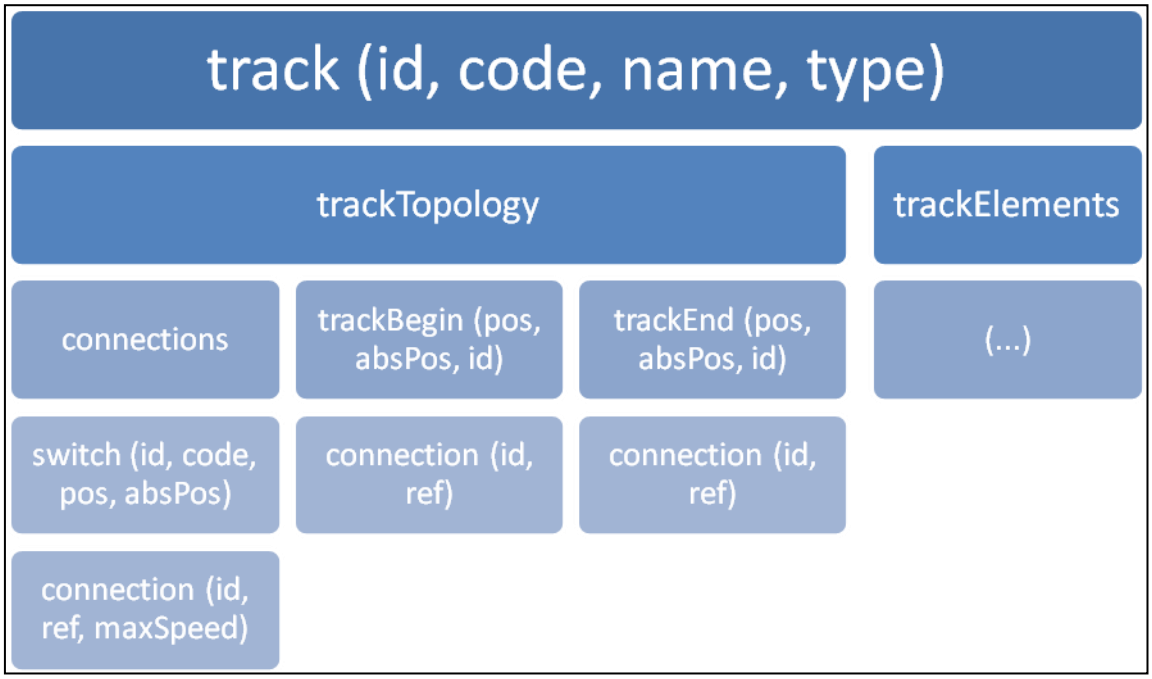


Fig. 3 Distribución de elementos

En ella, se observan de manera más clara que hay relaciones de elementos hermanos, como son <trackElements> y <trackTopology> entre sí; relaciones padre-hijo, como son los elementos <connections> y <switch> y, por último, un elemento raíz que engloba a los demás, que en este ejemplo sería el elemento <track>.

Además, para completar el estudio del esqueleto de un lenguaje XML, falta por definir el valor, que es un dato que queda adherido al elemento sin atributo de por medio (es decir, es el atributo específico del elemento). Este tipo de valor solo se necesita en dos elementos en RailML®, de modo que su importancia no es tan amplia como la de los atributos, sin embargo, es necesario conocerlos. Su posición dentro del código es la misma que la de un subelemento (entre la sentencia de apertura de su elemento y la de cierre). Incluyendo este último punto, la estructura de una sentencia sería la siguiente

<elemento atributo1='definición' atributo2='dato'>**Valor del elemento**</elemento>

## 4.2 Crear un archivo por primera vez

En el momento de comenzar a codificar, la pregunta más obvia es "¿cómo se puede abrir un archivo para crear un '.railml'?". La respuesta es bastante sencilla: xml, y por ello RailML®, son ficheros que se generaron en su día para un intercambio de datos sencillo. Por eso, son generados a partir de ficheros de texto plano o, lo que es lo mismo, cualquier tipo de texto sin importar el formato que se le dé a este. Esto significa que se puede generar un archivo RailML® con cualquier editor de texto de cualquier sistema operativo. De hecho, cualquier archivo de texto plano (como por ejemplo .txt) puede ser transformado en un archivo .xml o .RailML® con solo un cambio de extensión.

Una vez se tenga un archivo de texto abierto y preparado, viene la parte más delicada de toda la elaboración: la correcta definición del archivo, esquemas y versión de RailML® utilizados. Para ello, se deben seguir los pasos enumerados a continuación:

- Rellenar una primera línea con la información relativa al código (versión del código y codificación). Para esta línea puede servir *<?xml versión='**' encoding='**'>* donde los asteriscos deben ser reemplazados por los valores correspondientes (la codificación más común es "utf-8", de formato de transformación Unicode de 8 bits.
- Escribir (o copiar si es posible) una primera estructura vacía. Si se opta por escribir la estructura, la vía más común es primero escribir los subesquemas principales, vacíos de contenido.
- Comenzar a rellenar los elementos según se indica en la parte de Subesquemas.

## 4.3 Reglas y restricciones

RailML®, al igual que XML, definen ciertas normas de obligado cumplimiento. Si estas normas no se cumplen, en la mayoría de casos, se producirá una limitación en la manera de leer el código, e impedirá conseguir los objetivos del código. Por ello, es importante tener presentes los siguientes puntos para definir correctamente un archivo RailML®.

### 4.3.1 Orden en el documento

Como en cualquier lenguaje estructurado, el orden de posicionamiento de cada sentencia del código tiene influencia en la comprensión del mismo por la máquina. Esto significa que, aunque la mayoría de lectores de código XML (o RailML®) son capaces de entender y subsanar estos errores, en ocasiones pueden resultar perjudiciales para la interpretación de resultados en los programas destinados a tal efecto.

Obviamente, es imposible saber de antemano cuál es el orden predeterminado para cada subelemento del código sin embargo, para esto, hay dos posibilidades. Como primera posibilidad está la validación de código según programas como RailVIVID® (o cualquier otro programa de validación de estructura de XML), aunque este método implica completar el código, validar, y cambiar los errores que se muestren, duplicando por ello parte del trabajo. La segunda, siendo esta la más limpia y correcta, es seguir las líneas que se dan en los esquemas. Estos esquemas son descargables de la página de RailML® como se indica en el pie de página del siguiente punto (subesquemas). Estos esquemas, en sí, son complicados de leer dado que no se parecen en casi nada al propio código que se generará como digitalización de la información de las vías.

La estructura de estos esquemas difiere un poco entre los elementos que tengan hijos y los que no, pero siempre seguirá unos patrones:

- Todos y cada uno de los elementos permitidos en el código estarán reflejados en este esquema.
- Cada uno de ellos estará reflejado por un elemento *<xs:complexType>* que tendrá como nombre el nombre del elemento RailML®.

- Cada elemento tendrá un apartado llamado *<xs:sequence>* o *<xs:complexContent>* que contendrá lo que buscamos, el orden de los subelementos que se incluyan en el elemento en cuestión. Obviamente, si no hay subelementos, puede no aparecer este contendor.
- Asimismo, también tendrá información necesaria para saber utilizar el código como son el número mínimo y máximo (si lo hay) de veces que debe aparecer con "minOccurs" y "maxOccurs", y los formatos de dato en *<xs:extension>* bajo el atributo "base" o en el atributo "type".
- El formato de dato que sigue, se encuentra normalmente en otro archivo también descargable desde la misma página que indica los diferentes formatos. Este se puede encontrar (si es un tipo string o cadena de caracteres, un tipo integer o entero, o algún tipo específicamente definido como el tipo tBrakeType que comprende los valores "none", "compressedAir", "vacuum", "handBrake", "parkingBrake" o "cableBrake") bajo el elemento *<complexType>* y que indica dentro del mismo, los diferentes valores que puede tomar, así como una explicación de cada valor si esta es necesaria (por ejemplo, en el caso de que se usen abreviaturas).

### 4.3.2 Validaciones y comprobaciones

Como norma general, todo código que vaya a ser usado como codificación de RailML®, debe pasar dos pruebas que marcarán la aptitud de este código para ser representado por medio de alguno de los programas de interpretación, como RailVIVID®. La primera comprobación será una validación de estructura, para la cual se comprueba que el código esté bien formado y siga los cánones del lenguaje XML. Entre estas comprobaciones están: que todo elemento abierto sea cerrado en el mismo nivel que está abierto, que los atributos no aparezcan repetidos, o que el esquema (xmlns) que se va a utilizar esté definido y pueda ser encontrado. El éxito de esta constatación implica simplemente que el archivo puede ser abierto por el software que se utilice para ello. Sin embargo, en este punto, el código es una caja negra, que puede tener o no información, y esta puede estar bien formada o no. Esto es lo que se busca en la segunda comprobación (llamada validación en RailVIVID®), en la que se busca que los elementos obligatorios estén presentes, que el orden sea el correcto, que no haya referencias ciegas (como se llama a una referencia a algo que no existe, esto es, cuando un elemento se llama ordenador, y al referenciarlo lo llamamos computadora, la máquina no será capaz de encontrarlo; este ejemplo es obvio pero es un error que pasa con alta frecuencia al escribir, dado que si, en lugar de ordenador se busca *Ordenador*, u *ordeador,* tampoco serán encontrados), o que los elementos que tienen un número máximo o mínimo de llamadas, cumplan este número.

A la hora de programar, es altamente complicado ser tan preciso durante los miles de líneas que se generan en este tipo de códigos. Para ello es tremendamente recomendable fijar desde el principio unos estándares de nomenclatura que se puedan seguir durante todo el desarrollo de la codificación. Una vez que se termine el código, es también recomendable hacerlo pasar por uno de los muchos comprobadores estructurales de XML (sirve de la misma manera) que existen en internet, puesto que RailVIVID® hace esa comprobación pero no da la información referente a dónde se encuentra el fallo que pueda desencadenar la mala construcción, o cuál es el fallo que se ha producido.

### 4.3.3 Tipos de datos

Como se ha mencionado anteriormente, toda la redacción de código que se haga en RailML® debe seguir unas normas. Una de estas normas es, como en cualquier lenguaje, la formación de los tipos de dato a usar en la definición de los atributos. Esto impide que se indiquen datos equivalentes de diferentes maneras (por ejemplo, no se puede señalar el punto kilométrico de un elemento con letras, siempre debe ser con un dato numérico).

#### Datos comunes

Las nociones básicas de cualquier lenguaje de programación o de datos, comienzan por conocimiento básico de los tipos de datos a utilizar. Por eso, y a causa de la inmensa cantidad de lenguajes diferentes, se tendió a homogeneizar los tipos de datos usados (lo que no es otra cosa que dar nombre a los diferentes datos que son conocidos por todo usuario), por ejemplo, a la cadena de caracteres se le llamó tipo de dato "string" (de cadena en inglés), o al dato de los números enteros, "integer". Siguiendo esta lógica, existen otros muchos tipos de dato que son ampliamente conocidos y pueden ser encontrados en cualquier texto acerca de programación. Los más importantes en RailML® son "boolean" (tipo booleano, verdadero o falso), "string" e "integer".

#### Listas de datos, y datos específicos

En determinadas ocasiones, no todos los tipos comunes cumplen las especificaciones características de un atributo, o no son adecuadamente específicos o concretos. Para esto, XML da la posibilidad de generar, para atributos en concreto, tipos de datos personalizados, donde, por ejemplo, se puede dar al usuario la posibilidad de elegir entre valores diferentes de una lista. Un buen ejemplo de esto son los sentidos de circulación, estos son diferentes dependiendo de si dicho sentido se aplica al sentido prioritario de una vía, o al sentido al que se asigne una señal, es decir, una vía puede tener un sentido de circulación preferencial, o no tenerlo, por ello ese dato será elegido de entre los valores de la lista [up, down, none] (términos en inglés para arriba, abajo o ninguno). Sin embargo, una señal siempre está orientada a un sentido, por ello, el atributo de dirección, dentro de una señal, tendrá una limitación por tipo de dato según la lista [up, down]. De ahora en adelante, todos los atributos que tengan tipos de datos de una enumeración, se especificarán incluyen sus posibles valores entre corchetes, es decir, siempre que el lector encuentre una lista de valores entre corchetes , esta se referirá a los posibles valores del atributo junto al que se encuentre.

Esto es también, lo que pasa en RailML® con la mayoría de datos numéricos racionales; para no extender demasiado la longitud de datos que no necesitan demasiada precisión, se restringen algunos valores a solo dos, tres o cuatro cifras decimales según convenga. Este es el caso del atributo posición, que está limitado a seis cifras decimales, dado que es una medida expresada en metros, y no tiene sentido afinar más allá del milímetro aunque, en este caso se dé libertad de hacerlo hasta la micra.

### 4.3.4 Tablas y gráficos

Para muchos de los datos que se utilizan en Ingeniería en general, los gráficos son de una inmensa utilidad. Sin embargo, son complicados de digitalizar dado que para ello hay que discretizar valores continuos. Para ello, en RailML®, se pueden incluir tablas de datos para varios datos, como son las resistencias al avance en función de la velocidad. Asimismo, hay tablas que dependen de una tercera variable, como es el caso de la inclinación en el ejemplo anterior. Por esto, es importante la capacidad del lenguaje para modelar estas tablas de datos. Aunque el código puede parecer complicado y engorroso, una vez se entiende la estructura de estas tablas, el método de confección de las mismas es sencillo y rápido, incluso pudiendo automatizarse fácilmente.

Un ejemplo de estructura de las tablas es la siguiente:

```
<valueTable xValueName='velocidad' xValueName='km/h' yValueName='resistencia'
yValueName='N' zValueName='inclinación' zValueName='%'>
        <columnHeader zValue='0'/>
        <columnHeader zValue='0.1'/>
        <valueLine xValue='50'>
                <values  yValue='30'/>
                <values  yValue='35'/>
        < valueLine/>
        <valueLine xValue='100'>
                <values  yValue='60'/>
                <values  yValue='75'/>
        < valueLine/>
< valueTable/>
```

Esto representará la siguiente tabla.

| Resistencia (N) | velocidad | |
|---|---|---|
| Inclinación | 50 km/h | 100 km/h |
| 0% | 40 | 70 |
| 0,10% | 145 | 195 |

Tabla 1: valores de resistencia al avance en función de velocidad e inclinación

NOTA: véase que los valores de la variable y se incluyen de n en n dentro de la variable x; siendo n la cantidad de tablas que se definen (es decir, cantidad de valores z) habiendo una relación entre el orden en que están definidas las variables z y las variables y.

### 4.3.5 Dublin Core Metadata Initiative

Existe una iniciativa que merece mención en este trabajo, dada su importancia en el esquema common, así como en algunos subelementos <metadata>. Esta iniciativa es un intento de

compromiso para el uso de metadatos en lenguajes estructurados como XML. Consiste en la definición de unas reglas (quince actualmente) para estandarizar la semántica y la sintaxis de elaboración de datos metadatos (es decir, recursos que sirven para definir recursos) en la informática en general. De este modo, se consiguió generar una norma ISO que contiene normativa de organización de datos con respecto si los datos tratan del contenido de los mismos, su propiedad intelectual, o la instanciación de dichos datos.

Hoy en día, prácticamente cualquier página web basada en HTML, sigue las directrices del Dublin Core para la generación, estructuración y uso de metadatos; y esto permite que dichos metadatos puedan ser interpretables a escala global de una manera sencilla.

Como curiosidad, el nombre de esta iniciativa, se debe a que la primera reunión que se tuvo al respecto de este tema, fue en Dublín, Ohio, en Estados Unidos.

## 4.4 Subesquemas

Antes de seguir con este estudio, se debe dejar claro que el uso de la palabra esquema en este tipo de lenguajes, se ha ampliado, refiriendo como esquema cualquier conjunto de elementos que se definen bajo unas mismas normas, cuando el esquema propiamente dicho es el archivo (.xsd o .xsdl, de lenguaje de definición de esquema XML) que guarda y establece estas normas. Así, el *elemento* Infrastructure (llamado habitualmente esquema), se define por el esquema Infrastructure.xsd.

La continuación lógica de este análisis lleva inmediatamente a hablar sobre cuáles son y de qué están formados los cinco subesquemas[3] de este lenguaje. Como se ha mencionado anteriormente, estos subesquemas son Infrastructure (IS), Rollingstock (RS), Timetables and Rostering (TT), Interlocking (IL)y Common (CO). Ellos son, en orden, los subesquemas dedicados a Infraestructura, Material Rodante, Horarios, Bloqueos y Elementos comunes.

### 4.4.1 Esquemas de definición

Para ello, como en cualquier archivo XML, se pueden conseguir de manera sencilla la definición de los esquemas que usa RailML®, pero es necesario saber interpretarlos, por eso es necesario tener una visión sobre cómo leer e interpretar un esquema que defina RailML®. En estos esquemas, se presentan elementos para cada uno de los posibles elementos que aparecerán en el esquema final. Dentro de cada uno de esos elementos aparecen tanto atributos que definen cómo deben estar representados, como subelementos que darán información varia, desde qué elementos se pueden anidar en él, hasta qué tipos de atributos y datos debe contener. A continuación se dan definen los elementos más importantes de los esquemas para comprenderlos correctamente:

<xs:schema> nos dará información sobre dónde se aloja el esquema (su URL) y algunas notas.
<xs:element name="formations" type="rail:eFormations" minOccurs="0"> define un elemento y, todo lo que esté dentro de él será parte de su definición. Aquí también se define cuántas

[3] https://www.railml.org/en/download/schemes.html y https://wiki.railml.org

veces puede aparecer como máximo o mínimo, de qué tipo es este elemento, y otras restricciones.
<xs:complexType> o <xs:simpleType> definirán si el elemento es de tipo complejo o simple.
<xs:complexContent> incluye todo el contenido de el elemento (atributos y/o subelementos).
<xs:extension base="rail:tRollingstock"> define el tipo de datos que alberga.
<xs:sequence> definirá la secuencia de los elementos incluidos en él. La secuencia que se muestre dentro de <xs:sequence> es la que se recomienda usar en RailML® para un correcto funcionamiento.
<xs:attribute> define un tipo de atributo.

De este modo, si tenemos un elemento que contenga en su esquema el dato minoccurs='1' ya sabemos que este elemento tiene que estar presente una vez al menos.

En RailML® se utilizan dos tipos de esquemas, uno para la estructura de los elementos y otro para los atributos. Asimismo, se definen estos dos esquemas para cada uno de los cuatro (son cuatro en la actualidad pero serán cinco posteriormente) subesquemas que se definen a continuación.

### 4.4.2 Infrastructure

Este subesquema guarda y organiza toda la información referente a la estructura de la red, como son las vías, estaciones, operadores, etc. También utiliza referencias cruzadas de los otros subesquemas, de manera que puede relacionar un tramo de vía con un horario específico, o la vía de una estación, con un tipo de tren.

Como es obvio, el punto principal, y en el que más información hay que tratar, es la vía. Para ello, existen los elementos <track>, en los que se agrupan circuitos de vía relativos a una misma vía, y se aloja toda su información. No existe una regla dentro de RailML® que obligue a agrupar las vías de una manera determinada. Esta distinción debe ser elegida por los operadores de la red de la manera que más les convenga. Una buena manera es generando un elemento <track> por cada circuito de vía puesto que, aunque esta manera genere más código, se mantiene más orden y se facilita la comprensión de las vías y su estructura.

En el caso práctico, sin embargo, por motivos de tiempo y sencillez de presentación, se ha optado por presentar las vías haciendo una similitud entre el elemento <track> y el tramo de una vía que es continua y se encuentra bajo el control de un mismo enclavamiento. Así, se obtienen elementos <track> más densos, pero se evita repetir datos que no tienen demasiada utilidad.

Estos elementos <track> se encuentran alojados dentro de un elemento contenedor llamado <tracks>. Esto ocurre con asiduidad en RailML®, por lo que es conveniente prestar atención a la definición del elemento para no cometer errores (recordemos que, si asignamos un atributo a un elemento que no tendría que estar asignado a él, recibiremos un error de lectura cuando queramos abrir el archivo).

Los elementos que son directamente descendientes de este subesquema son los siguientes:

| | | |
|---|---|---|
| <additionalName> | <any> | <infraAttrGroups> |

| | | |
|---|---|---|
| <metadata> | <operationControlPoints> | <trackGroups> |
| <tracks> | <controllers> | <speedProfiles> |
| <states> | | |

Tabla 2: subelementos del esquema Infraestructure

(el elemento <states> es introducido a partir de la versión v2.4, por lo que tampoco aparecerá en el ejemplo práctico). Dentro de cada uno de estos elementos, podremos ver elementos anidados, así como atributos sin embargo es bastante extraña la presencia de textos en los elementos de RailML®.

El subesquema de la infraestructura es, sin duda, el que más información suele contener, dado que hay mucho más que definir en ello (señales, cambios de vía, estaciones, velocidades, etc.) que en los otros esquemas. También se debe en parte a que, como es lógico, hay mayor cantidad de kilómetros de vía que de tren.

### 4.4.3 RollingStock

Este apartado estructura y almacena toda la información relativa al material rodante. Este subesquema tiene dos elementos descendientes directos, que son

| | |
|---|---|
| <formations> | <vehicles> |

Tabla 3: Subelementos del esquema rollingstock

En ellos, se hace un análisis de cada vagón por separado (en el elemento <vehicles>) con atributos inherentes a dichos vagones, como son su peso intrínseco, su capacidad de frenado, sus motores, su peso máximo autorizado, su aceleración, o la capacidad de paso por curva que tienen, entre muchos otros.

Por otro lado, el elemento <Formations> es el encargado de dar forma a un tren, uniendo elementos descendientes del elemento <vehicles>, y adjudicándole unas características específicas tales como el orden de los vagones, o el perfil de velocidades.

Como se ha mencionado anteriormente, en este subesquema se utilizan elementos de <vehicles> en <formations>, lo que es posible gracias a las referencias (ref [idref]) que son maneras de identificar que un elemento hace mención o está relacionado de alguna manera con otro elemento que puede no estar en el mismo subesquema, o en el mismo elemento padre.

### 4.4.4 Timetables and Rostering

Este es el esquema RailML® encargado de almacenar información referente a la organización temporal de la red, bien sea de horarios, períodos operativos, asignaciones entre otros. Los elementos de primer orden de este subesquema son los siguientes:

| | | |
|---|---|---|
| <additionalName> | <metadata> | <timetablePeriods> |
| <operatingPeriods> | <categories> | <annotations> |
| <trainParts> | <trains> | <trainGroups> |
| <rosterings> | <xs:any> | |

Tabla 4: Subelementos del esquema rostering

Estos elementos sirven para explicar el funcionamiento de la red, en términos de frecuencias, días de servicio especial, horarios y demás elementos necesarios para la correcta explicación del servicio. Es de mención que el elemento <trains> así como el elemento <trainParts> y <trainGroups> parecen tener más cabida en el subesquema RollingStock que en este. Esto se debe a que estos elementos se utilizan como identificación para unir o relacionar partes de distintos subesquemas entre sí. Ello significa que, se puede relacionar qué tipo de trenes pasa por la vía. Del mismo modo, en los demás subesquemas se ofrecen posibilidades para conectar elementos de diferentes subesquemas entre sí. Más adelante, con la explicación en detalle de cada subesquema, se verán ejemplos que clarificarán esta cuestión. Es lógico pensar que esto también ocurre entre diferentes elementos de un mismo subesquema.

### 4.4.5 Interlocking

Es necesario subrayar que este subesquema estará disponible a partir de la versión 3 de RailML®, por lo que no ha sido objeto de estudio profundo en este trabajo dado que el ejercicio práctico se desarrolla sobre RailML® v2.3. Sin embargo, dado que el objetivo del mismo es analizar y explicar la validez y uso de este lenguaje, se debe añadir que éste es el esquema referente a la coordinación de bloqueos. Es decir, este esquema es el que se encargará de estandarizar el funcionamiento de las señales según el estado de la vía.

### 4.4.6 Common / Metadata

El subesquema Common se puede utilizar en las primeras versiones de RailML® para hacer referencia a todos aquellos elementos que son, por definición, comunes a varios de los subesquemas anteriores, o a todos ellos.

Sin embargo, se introdujo la posibilidad de utilizar metadatos como un elemento más. Estos metadatos permiten agrupar dos tipos de datos: por un lado los datos que, por sus características, no son aptos para ser englobados dentro de ningún otro subesquema o, por el contrario, datos que deben estar presentes en todos ellos. Estos datos, se rigen por elementos que siguen la estructura propuesta por la iniciativa Dublin Core Metadata[4], referentes a una estandarización de buenas prácticas, y de reglas sintácticas que hacen los datos comprensibles para cualquier tipo de lector que se enfrente a ellos.

Como se puede ver en la siguiente tabla, referente al subesquema Metadata, este se compone de diecisiete elementos principales que, a su vez, se componen de varios subelementos diferentes como viene siendo común en este lenguaje.

| | | |
|---|---|---|
| <any> | <dc:contributor> | <dc:coverage> |
| <dc:creator> | <dc:date> | <dc:description> |
| <dc:format> | <dc:identifier> | <dc:language> |
| <dc:publisher> | <dc:relation> | <dc:rights> |
| <dc:source> | <dc:subject> | <dc:title> |
| <dc:type> | <organizationalUnits> | |

Tabla 5: Subelementos del esquema common

[4] http://dublincore.org

La utilización de este subesquema nos permite, por ejemplo, introducir en el código datos referentes al creador del mismo, que pueden ser útiles en caso de dudas o correcciones; datos referentes a derechos de copia o uso; o datos de organizacionales.

Es de destacar que, además, de este subesquema, existen elementos *<metadata>* dentro de cada subesquema para introducir estos datos, según el mismo funcionamiento, es decir: datos que influyan en todos los elementos de un subesquema o que no influyan en ninguno de estos elementos pero tenga importancia para el subesquema del mismo modo.

# 5 Marco de desarrollo de un modelo en RailML®- Buenas prácticas

La primera buena práctica es esencial para cualquier trabajo con un archivo tipo XML, y es utilizar un editor de textos adecuado. Si bien es cierto que cualquier editor de texto plano puede servir, utilizando un editor con formatos se obtienen varias ventajas, como la posibilidad de agrupar elementos, o diferenciar por colores de texto entre elementos, atributos, comentarios y valores de manera automática. Asimismo, se tienen comandos dedicados para según qué funcionalidad, como por ejemplo transformar una frase en un comentario con un atajo de teclado sencillo, o la contracción del texto por niveles.

Una vez elegida la aplicación a usar para editar el código, se proporcionan unos consejos que pueden facilitar formidablemente el trabajo a llevar a cabo.

## 5.1 ID

Este es, posiblemente el atributo con más importancia de todos los presentes en el código. Esto es así, como se explica en la parte de diagramas y sus elementos, porque sirve para unir y referenciar elementos que están en estructuras diferentes, o en elementos diferentes de una misma estructura. Por esto, como se ha dicho, es vital que estos ID sean únicos y, a la vez, sean reconocibles por el usuario que pueda modificar dicho código. Para eso, la propuesta usada en el desarrollo de este trabajo y que ha resultado bastante útil es componer el ID en tres o cuatro partes (según sea necesario) en que la primera parte indique qué elemento está definiendo el ID, el segundo elemento indique a qué grupo de elementos pertenece este ID dado que ese elemento se puede repetir en varios puntos del código (como es el ejemplo de una señal llamada "E2" que exista en varias vías), como tercer elemento se tendría una indicación al nombre o identificación propia del elemento que se va a usar y, si es necesario, un número para elementos que se repitan o se necesiten ordenar.

Con el ejemplo expuesto de una señal "E2" que, supongamos, está en una vía que se ha llamado "vía 1425", un ID organizado de esta manera quedaría así ID='sñl_1425_E2_01'.

## 5.2 Nomenclatura

Obviamente, acompañando el punto anterior, se necesita una nomenclatura consistente. Es decir, es una buena práctica elegir un tipo de nomenclatura y seguirla durante todo el desarrollo de esta. Para eso, se debe tener claro desde un primer momento, cuáles serán los elementos que se van a definir, y cuáles serán los nombres que se quieran dar a dichos elementos. Por supuesto, habrá elementos cuyos nombres vengan impuestos de un principio por el nombre del elemento real que se modela; como en el caso de las vías, no tiene sentido llamar a una vía con un nombre o identificador diferente al que tiene en la vida real, por mantener una consistencia en el código, puesto que esa consistencia la da la red de la infraestructura.

Dentro de la nomenclatura, hay que hacer una mención especial a que este código es sensible a mayúsculas, lo que significa que el código no entenderá el elemento <Train> o <TRAIN> si la definición en los esquemas señala que el elemento es <train>. Asimismo, se obtendrán errores si llamamos por referencia al ID *'tun_132_Ujo_01'* si el ID se ha definido como *'tun_132_ujo_01'*. En una definición de un código que muy fácilmente supera las diez mil líneas de código, es necesario mantener un criterio firme en este aspecto, para evitar tener que invertir largo tiempo en resolver errores más adelante.

## 5.3 Planificación

RailML® contiene una cantidad enorme de datos, esto implica que la creación del código va a ser larga. Por otro lado, a consecuencia de las relaciones que se pueden establecer entre varios elementos, es un trabajo de desarrollo que va a necesitar de volver sobre sí mismo para comprobar datos, saber cómo se ha denominado cierto elemento, o comprobar si dicho elemento tiene los atributos que se requieren.

Igualmente, con mucha seguridad, se necesiten documentos de soporte sobre los que apoyarse para consultar esas dudas. Para esto, una hoja de cálculo puede cumplir las expectativas, pero es vital saber qué datos se van a necesitar.

Por estos dos motivos, es muy recomendable, antes de comenzar a generar código, tener una visión general de lo que el código nos va a requerir, y los datos que vamos a necesitar, puesto que si esto no se hace cabe la posibilidad de que se den por innecesarios datos (atributos) que posteriormente se necesitarán, o incluso que se comience a modelar teniendo que dejar vacíos que requieren haber rellenado otros esquemas anteriormente.

Si por algún motivo, esta planificación no puede hacerse correctamente, o no puede completarse, otra posibilidad es definir una palabra clave que no vaya a ser usada en el resto del código de manera que, una vez el código esté terminado, se pueda buscar esa palabra en el editor de textos, sabiendo que en cada punto en que se ha puesto esa palabra, falta información por completar, o hay algo que debe ser revisado.

Por los mismos motivos, y dado que los elementos track son una de las partes más importantes del código, se da que una manera preferencial de actual es la de tener clara la subdivisión de vías que se va a llevar a cabo en el modelado de la red. Como se vio en su definición, la red se puede dividir en elementos <track> de varias maneras y, debido a esto, es importante tener clara cuál va a ser esa división mucho antes de comenzar a escribir código referente a la infraestructura.

### 5.3.1 Reconocimiento de esquemas

En la misma dimensión que la planificación, se encuentra un punto vital al a hora evitar errores que, a posteriori, se traducirán en tiempo gastado en recuperar errores de formato. Este punto es el orden que los elementos deben llevar en el código. Según qué dato esté en un puesto incorrecto, hará que los datos sean irreconocibles, o que solo tengamos un aviso sobre la incoherencia de la estructura.

Igualmente, se debe evitar ambos tipos de errores. Para ello, la mejor forma es, antes de comenzar el modelado, haber tenido una revisión de la parte que se va a modelar en los esquemas de definición.

### 5.3.2 Esqueleto del RailML® vacío

Como más adelante se dice, hay la posibilidad de automatizar la creación del código según los datos que se tengan y ciertas reglas. Sin embargo, si el código se va a generar de manera manual, es de mucha ayuda crear un esqueleto vacío, o lo que es lo mismo, una estructura completa de RailML® pero sin valores en sus atributos, de manera que, una vez que se vaya a pasar el contenido al código, no haya que preocuparse por la estructura y solo haya que completar los valores.

Otro modo de llevar a cabo esta metodología es conservar (al menos) dos archivos, uno con el esqueleto vacío y otro de trabajo, de modo que cada vez que se quiera agregar un elemento cualquiera, se pueda copiar y pegar su esqueleto desde el archivo vacío.

También es muy útil intentar compilar el código cada vez que se haya terminado un elemento, para poder acotar errores en el código, si es que los hay.

### 5.3.3 Recolección de datos

Puesto que RailML® es un lenguaje basado en estructuración y organización de datos; dichos datos son una parte necesaria de la generación de archivos RailML®. Obviamente, estos datos tienen una procedencia y es importante, antes de comenzar a modelar, haber asegurado que los datos de que se dispone cubren el espectro de información que se necesita para completar el código satisfactoriamente.

Una de las maneras óptimas para proceder es, tras haber entendido cuál es la estructura del código, dividir la infraestructura en las partes que se vayan a indicar en el archivo, es decir, si se va a establecer un elemento <track> por cada vía al completo, por cada circuito de vía, o cualquier otro tipo de división. Una vez esto esté hecho, se identificarían todos los elementos que están presentes en dicha vía (como señales, aparatos, puentes, y demás elementos presentes en el subesquema), y pasar esos datos a un modelo organizativo como se puede ver en el punto siguiente.

### 5.3.4 Organización de datos

Seguidamente, es obvio que es mucho más sencillo trabajar con datos fáciles de leer, estructurados y bien organizados. Por eso es una buena idea gastar un poco de tiempo en organizar los datos que se van a utilizar según las necesidades del código. Puede sonar obvio, pero es una tarea que puede librar de más de un quebradero de cabeza puesto que, a la hora de escribir código, no hay comparación entre obtener los datos de una tabla, o una lista, que de los mapas o planos de vías.

Este paso es imprescindible si se quiere ahorrar tiempo y, sobre todo, si se pretende definir algún tipo de automatización.

### 5.3.5 Automatización

Una vez que se hayan organizado los datos en un formato digital adecuado, es muy ventajoso tratar de automatizar la generación del código. Puesto que todas las líneas siguen una estructura conocida, y los elementos y subelementos son también conocidos, se puede crear una herramienta simple que ayude a crear el código de manera automática a partir de los datos guardados. Dependiendo de la sencillez con la que esto se haga, se puede crear una herramienta que luego precise una reestructuración del código ( o al menos una supervisión) o se puede generar una herramienta que genere el código completo con toda su estructura. Sin embargo, esta última posibilidad requiere una inversión de tiempo y esfuerzo más elevada en un principio, además de una mejor ordenación de datos, con la necesidad de presentar las relaciones y referencias en las tablas de datos que se usen.

Sin embargo, la forma que, a priori, es más sencilla, es utilizar las propias fórmulas de un editor de hojas de cálculo, en las que se concatenen los elementos con sus atributos, según la estructura que defina el usuario.

## 5.4 Xpath

A la hora de desarrollar el código, así como para el desarrollo de herramientas que puedan interactuar con un código de tipo XML, es de una excepcional ayuda conocer qué es y cómo se utiliza XPath. XPath es un lenguaje que recorre el archivo y lo modela como un árbol de nodos. Esto le permite destacar elementos según se desee encontrar ciertas estructuras dentro del archivo XML (o RailML®). La utilidad de este lenguaje es inmensa, puesto que permite recibir datos de forma discreta.

XPath responde a las siglas de XML Path lenguage (o séase, lenguaje de camino XML). Es así porque, Xpath se basa en definir un elemento según el camino que llega a él (bien sea desde el principio del documento, o desde un elemento que sea único y fácilmente reconocible). Para eso, establece ciertas reglas sintácticas, y acepta algunas funciones lógicas que facilitan la tarea de la examinar elementos según sus propiedades. Así, Xpath da la posibilidad de encontrar datos como "el primer elemento de todos los <formation>", o "los elementos <track> que no contengan elementos <connection>". Esto permite hallar elementos, o contarlos, según se haya definido.

Queda claro que XML es un lenguaje que se destina a organizar los datos, y a que éstos puedan ser consultados de manera directa, sin tener que recorrer todo el documento; pero para eso hace falta identificar qué elemento se quiere encontrar (o qué elementos) y aquí es donde entra XPath. La mayor utilidad de esto es su uso como expresiones regulares[5] dentro de programas de interpretación, solo que buscando datos en un XML en lugar de cadenas en un texto.

En algunos editores de texto es posible utilizar XPath, aunque la mayoría requieren la instalación de un plugin; y esto se puede utilizar para la detección rápida de cierto tipo de errores (sobre todo los errores por omisión).

[5] Las expresiones regulares o racionales son formas de generar un patrón de búsqueda de cadenas de caracteres según una secuencia de tipos de caracteres.

En el texto siguiente se dan dos ejemplos de cómo actuaría un Xpath en parte de un fragmento del caso práctico usado.

Se han creado tres sentencias XPath para señalar los ejemplos. Cada uno de los Xpath señalará un tipo de dato diferente:

| | |
|---|---|
| //braking[@brakeType] | ejemplo (1) en *cursiva* |
| //speedProfile[ 1] | ejemplo (2) en subrayado |
| /speedProfiles/speedProfile[2]/braking | ejemplo (3) en **negrita** |

```
<speedProfiles>
    <speedProfile id='veloc_estac' code='veloc_estac' influence='other:none'
    name='velocidad de estacion' xml:lang='ES'>
        <tilting maxTiltingAngle='5.83' actuation='none' tiltingSpeed='30' ></tilting>
        <braking brakeType='compressedAir' airBrakeApplicationPosition='P'
        minimumBrakePercentage='15' / >
    </speedProfile>
    <speedProfile name='velocidad de tramo de via' xml:lang='ES' maxAxleLoad='22.5'
    maxMeterLoad='7.2'  trainProtectionSystem='ASFA'>
        <braking minimumBrakePercentage='120'></braking>
    </speedProfile>
    <speedProfile id='veloc_cambio' code='veloc_cambio' >
        <tilting maxTiltingAngle='5.83' actuation='none' />
        <braking brakeType='compressedAir' airBrakeApplicationPosition='P'/ >
    </speedProfile>
</speedProfiles>
```

# 6 Caso práctico - Modelo en RailML

Como es normal en este tipo de metalenguajes, la anidación en forma de árbol se utiliza para tener agrupaciones (mayores o menores) de datos. Para hacer más gráfica la explicación, siguiendo con el símil de la estructura de "árbol", los datos están representados por las hojas, mientras que los elementos serían las ramas. Estas ramas pueden salir directamente del tronco o de otra rama mayor y, del mismo modo, cualquiera de estas ramas puede tener hojas, contener otras ramas que tengan dichas hojas, o ambas cosas a la vez. La comprensión de este concepto es de alta importancia dado que, en RailML®, los atributos (hojas) se pueden hallar en cualquiera de los elementos (ramas) tengan o no más elementos anidados. Así, si una vía tiene unas características intrínsecas (por ejemplo el nombre o el código de la misma, o la dirección preferencial de la vía) cuyos datos irán adscritos al elemento vía; pero esta vía también tendrá subelementos, como los pasos a nivel, que también tendrán sus características y, por tanto, sus datos adjuntos.

Del mismo modo, los datos representados (llamados atributos) son los que guardan la cualidad del dato. Esto significa que si el usuario necesitase guardar la información relativa a que hay un coche con sistema de frenado hidráulico, dentro del subelemento <trainBrakes>, se incluirá un atributo llamado "brakeType" con valor "compressedAir", de modo que el elemento quedaría definido así: (...)<trainBrakes brakeType='compressedAir'>(...)</trainBrakes> en el caso en que este elemento albergase más subelementos, o de la siguiente manera si no los tuviera: (...)<trainBrakes brakeType='compressedAir'/>(...).

Todos los atributos, a su vez, deben seguir una sintaxis específica para que, al traducir el código con los programas de interpretación (se verán más adelante), los datos sean coherentes. Técnicamente, esto se consigue por medio de los esquemas utilizados, de manera que un campo que esté configurado para introducir una fecha solo podrá contener datos de formato fecha. El incumplimiento de este último punto supone la imposibilidad de lectura del archivo y, por tanto, la anulación de su utilidad. Muchos de estos formatos se rigen por estándares utilizados anteriormente en *XML* por dos motivos: obviamente porque, al ser un lenguaje heredado de este, utiliza muchas de sus características; y porque los formatos que usa XML tienen una amplia utilización en el mundo de la informática y han sido aceptados por toda la comunidad informática como estándares a utilizar.

A continuación se va a desglosar una lista de los subelementos que se pueden utilizar, y cuáles son su significado y forma de uso. Como se ha escrito anteriormente, la estructura de cualquier elemento de RailML® será del tipo:

```
<elemento1 atributo1='valor' atributo2='valor' atributo3='valor'>
        <subelemento1 atributo1='valor' atributo2='valor'/>
</elemento1>
<elemento2 atributo1='valor' atributo2='valor' atributo3='valor'>
```

### 6.1.1 Elementos de <infrastructure>

El elemento infrastructure, a pesar de ser uno de los esquemas principales, no queda libre de tener atributos que la definen. Dichos atributos son: *id*, *code*, *name*, *description*, *xml:lang*, *version*, *xml:base*, *rollingstockRef*, *timetableRef*. Los primeros cinco, por ser referentes a la identificación del elemento, se repiten ampliamente durante todo el código (por este motivo, y teniendo en cuenta que su uso es idéntico en cada uno de los elementos, se explicarán solo una vez para evitar redundancias). Algunos, como id o code son identificadores para el código en sí mismo, mientras que name y description son atributos que se mostrarán al usuario para ser reconocidos. Es muy importante entender que el atributo id tiene que ser único, dado que es la identidad interpretable por la máquina a la hora de leer el código (podría decirse que es el DNI de cada elemento). ID será obligatorio en todos los elementos y relleno con un elemento tipo xs:ID (es decir un tipo string comenzando con una letra, y conteniendo solo letras, números o subrayados). Es necesario tener en cuenta que el ID será muy posiblemente utilizado en otros elementos para referenciar, esto es, si se tiene una elemento A (entiéndase el ejemplo de un tipo de freno) que va a ser utilizado dentro de otro elemento B (un vagón), el elemento B tendrá una referencia al elemento ID del freno. Esto será específicamente en próximos elementos. A razón de esta importancia, el ID tiene una restricción importantísima, que es que debe ser único, y reconocible por la máquina, es decir, un código alfanumérico de elementos ASCII (ver glosario).

Para ejemplificar esto, se da el siguiente código en el que el elemento <modelo> queda unívocamente unido al elemento <marca> por medio de su ID:

```
<marca id='marca01' atributo1='valor1'/>
<modelo id='modelo01' marcaRef='marca01'/>
```

Los otros cuatro atributos de denominación son opcionales, y todos se rellenarán con tipos string salvo xml:lang[6] que contendrá uno de los identificadores de idioma (ES para castellano, por ejemplo) en el que están expresados el resto de nombres y descripciones de este elemento. Este es uno de los ejemplos que se describen anteriormente por el formato específico dado que solo acepta formatos URI incluidos en la norma ISO 639, que son de la forma "es" para representar el español o "de" para representar el alemán. También se pueden añadir matices para la región de que proviene dicho idioma (por ejemplo "en-US" para el inglés de Estados Unidos o "en-UK" para el inglés británico, esta segunda sílaba se rige por la norma ISO 3166-1).

Por último, los elementos rollingstockRef y timetableRef son referencias a los subesquemas Rollingstock y Timetable respectivamente que son definidos de manera adjunta a esta infraestructura y tendrán el ID de dichos esquemas. El atributo xml:base contendrá una referencia a un elemento infrastructure externo (no utilizado en nuestro caso).

Dentro del esquema que contiene los datos referentes a la infraestructura, encontramos los siguientes elementos:

(en la enumeración siguiente, se ha adjuntado un código numérico que representa la relación entre elementos, de modo que cualquier elemento 1.5.n es sucesor directo del elemento 1.5, y

[6] https://www.ietf.org/rfc/rfc4646.txt

el elemento 8.6 es un elemento al mismo nivel que el 8.2; sin embargo, esta numeración no guarda relación alguna con el orden en que dichos elementos deban aparecer en la codificación; para ello habrá que consultar los esquemas proporcionados)

### 1 - additionalName:

Este es un elemento que se utiliza para dar diferentes nombres a cualquiera de los elementos donde está presente (este elemento está presente en varios elementos que pueden ser nombrados de más de una manera). Una de las utilidades más interesantes de este elemento es la nomenclatura en diferentes idiomas para, por ejemplo, partes de vía que vayan a ser transitadas por trenes provenientes de diferentes países (sobre todo en zonas cercanas a fronteras entre países), o de elementos que históricamente se han llamado de una manera y ahora se llaman de otra.

Sus atributos son *name*, *description* y *xml:language*.

Para el caso que contemplamos, la mayoría de estos elementos *additionalName* serán omitidos. Sin embargo, se han incluido algunos con el simple propósito de ilustrar cuál es su funcionamiento y cuáles son las posibilidades que ofrece.

### 2 - any:

Este es un elemento de bastante potencial en general pero, paradójicamente, poco útil a la hora modelar un elemento dado que los datos que normalmente son utilizados para la comprensión gráfica de una vía han sido ya añadidos a los elementos en modo de otros atributos. Por ello, en el ejemplo analizado, no hay ningún elemento <any> usado. Si se quisiera utilizar, se debe antes ampliar el esquema utilizado como referencia y definir un tipo de dato y estructura, con su consiguiente complicación. Por ello, su estudio queda fuera de los límites de este análisis.

### 3 - infraAttrGroups:

Dentro de RailML®, como se vio anteriormente, existen elementos cuya única función es ser un contenedor de elementos con propiedades comunes. El elemento infraAttrGroups es uno de ellos. Este elemento no tiene atributos y contiene a los elementos que vienen a continuación.

Dado que éste es el primer elemento contenedor que aparece, es útil en este punto recordar que los elementos que actúan como contenedor pueden tener también valores y atributos propios (caso del elemento <infrastructure>).

### 3.1 - infraAttributes:

El primer y único descendiente directo de 3 - infraAttrGroups. Puede estar repetido tantas veces como sea necesario y contiene los atributos *id*, *code*, *name*, *description*, y *xml:lan*. Id y Code son atributos muy importantes que se verán repetidamente en la vasta mayoría de los elementos del código. Ambos sirven para identificar el elemento en el que se encuentran, pero con una diferencia: ID se utiliza para referenciar los elementos internamente mientras que Code deberá utilizar un código alfanumérico que sea comprensible por cualquier colaborador que tenga acceso al código. En un ejemplo más claro, si este análisis contuviera ambos elementos, el ID podría ser "*tfg7045*" mientras que code sería "*analiz_RailML*".

En este caso, cualquier elemento <track> contendrá un elemento <infraAttrGroupRef> que hará referencia al ID de uno de los <infraAttributes> que se definan anteriormente. Con esto, se asociará automáticamente que dicha vía utiliza las características incluidas en el <infraAttributes> que se llama desde <infraAttrGroupRef>, de modo que no hay que repetir en cada vía qué electrificación tiene, o cuáles son las características de su radio.

### 3.1.1 - axleWeight:

Un indicador del peso máximo admisible en una vía. Contiene dos atributos que son *value* y *meterload* que indican respectivamente el peso máximo por eje en toneladas y el peso máximo por metro lineal en toneladas por metro. El primero de ellos es obligatorio, mientras que el segundo es opcional (especialmente interesante en puentes, viaductos, etc.), sin embargo es muy recomendable el uso de ambos atributos.

Para toda la ejecución del caso práctico, se ha tomado, por simplicidad, el peso máximo igual al peso estimado de circulación. Este se ha cogido como veintidós toneladas y media por eje, contando con cuatro ejes por vagón. Como caso especial, se ha obtenido de las tablas de valores, que la locomotora pesa en su totalidad ochenta y ocho toneladas métricas.

### 3.1.2 - electrification:

Definición del sistema eléctrico que se utiliza en la vía. Para ello, se usan los atributos *type* indicador del tipo de suministro de corriente, *voltage* y *frequency*. Los posibles valores de type son none (ningún sistema de transmisión), overhead (transmisión por catenaria), 3rdRail (tercer rail), sideRail (tercer rail lateral) y other:(...). En cuanto a voltage y frequency, sus valores serán siempre positivos y medidos en voltios y hercios. Los tres atributos son opcionales y tienen valores por defecto "none", "0" y "0".

En el ejercicio práctico, se entendió que solamente había un tipo de electrificación por motivos de sencillez de código, aunque esto no sea del todo correcto dado que hay zonas muertas en las vías. Por ello habría que definir al menos tres tipos diferentes de elementos <electrification>: uno para cada zona muerta, y otro para cada zona de electrificación (antes y después de la zona muerta).

Dada la amplia presencia de el atributo valor "other:anything" en muchos de los atributos presentes en RailML®, se debe aclarar el funcionamiento de este valor. Para que sea reconocible por la máquina, este debe formarse por la cadena de caracteres "other:" seguida de, al menos, dos caracteres más.

### 3.1.3 - epsgCode:

Ciertos elementos en este lenguaje pueden estar documentados por su posición geográfica exacta. Para ello, se necesita saber qué sistema de coordenadas se utiliza como referencia. Para esto se utiliza el elemento epsgCode que tiene dos atributos *default* y *extraHeight*. Estos indican respectivamente el sistema utilizado como referencia, y el sistema usado para la altura en caso de que sea un sistema aparte.

### 3.1.4 - gauge:

El elemento gauge muestra cuál es la distancia entre raíles de las vías. Su único atributo es el valor de la distancia, almacenado en *value*. Este debe ser un valor decimal medido en milímetros.

### 3.1.5 - clearanceGauge:

Muy parecido al anterior, pero esta vez referido a la distancia que existe entre las vías paralelas. En lugar de un valor medido en milímetros, se expresa el código [7] de los estándares que se han tomado en cada país. Los utilizados en España son GHE16 GEB16 y GEC16; y estos códigos se almacenan en el atributo *code*. El seleccionado para el caso hecho es GEB16.

### 3.1.6 - generalInfraAttributes:

De nuevo, un elemento contenedor de todos los <generalInfraAttribute>. Como la mayoría de elementos contenedor, no tiene atributos.

#### 3.1.6.1 - generalInfraAttribute:

Elemento contenedor de <attributes> también sin atributos.

##### 3.1.6.1.1 - attributes:

Elemento contenedor de <attribute> sin atributos.

###### 3.1.6.1.1.1 - attribute:

Este es un elemento donde se pueden almacenar los atributos globales de la estructura considerada. Sus atributos son name y value, ambos obligatorios y que albergan el nombre y el valor de los atributos. Es un elemento que, por sí mismo, no es necesario pero sirve para ampliar la cantidad de atributos que afectan a una vía. En el caso que nos afecta, se ha llenado con valores arbitrarios que no tienen sentido funcional, sino que tiene la única función de dar un ejemplo de cómo debería ser.

### 3.1.7 - operationMode:

Una misma vía puede regirse por uno o varios modos operativos. Para eso, este elemento contiene la información necesaria respectiva a su funcionamiento. Igual que en todos los elementos contenidos en <generalInfraAttribute>, estos modos serán aplicables y válidos en todos y cada uno de los elementos que tengan una referencia a estos atributos de infraestructura. Los atributos de este elemento son *modeLegislative modeExecutive clearanceManaging*. Los dos primeros son obligatorios y deben llevar datos de tipo string (cadenas de caracteres) mientras que el tercero es opcional y puede ser cualquiera de los siguientes valores: sight, time, blocking, LZB-blocking, absBrakeDist u otros. También fue relleno como ejemplo.

[7] https://wiki.railml.org/index.php?title=Dev:TrainClearanceGauges

### 3.1.8 - owner:

Elemento que establece los datos referentes al gestor de las vías a las que se referencia. Se compone de tres atributos sin ningún otro elemento anidado. Sus atributos son *uic-no* (opcional) que alberga el código UIC[8] del dueño y *infrastructureManagerRef* que indica la referencia al gestor que se haya definido en el subesquema metadata. Al analizar una línea española se ha cogido ADIF como gestor y Renfe como dueño, obteniendo el código UIC de una lista provista por UIC (la referencia a dicha lista está en el pie de página). El código para ADIF es 0071 y el de Renfe es 1071.

### 3.1.9 - powerTransmission:

El elemento powerTransmission es, como su nombre indica, un elemento que define cómo se hace la entrega de potencia entre vagón y vía. Para ello utiliza dos atributos que son *type* y *style.* Ambos son opcionales, el segundo se rellena con un tipo string y el primero tiene una lista de posibles valores que es: [adhesion, gearrack, cable o other:anything] para hacer referencia a la transmisión de potencia por rozamiento (ruedas), cremallera, cable u otros (por ejemplo, un tren de levitación magnética). Con rellenar el atributo type, el elemento queda suficientemente definido.

### 3.1.10 - speeds:

Elemento con mucha utilidad, contenedor de elementos <speed>.

#### 3.1.10.1 - speed:

<speed> permite definir perfiles de velocidad especiales (que difieren de los enumerados más adelante en speedProfile).

Como se ha dicho anteriormente, la configuración de este elemento otorga (junto con el elemento speedProfile que se verá más adelante) mucha utilidad y permite, entre otras cosas, definir en un primer momento cuáles serán los perfiles de velocidad que se usarán en las vías que estén referidas a estos valores. Esto implica una importante reducción de trabajo a la hora de codificar diferentes velocidades en la vía. Pongamos el ejemplo de dos posibles velocidades máximas para una estación con seis vías, una de esas velocidades afectaría a los trenes de pasajeros y la otra a los trenes de mercancías. Estos significaría que habría que codificar doce perfiles diferentes de velocidad (dos por cada vía) y la especificación en cada una de ellas de la categoría de trenes a los que esta limitación afecta (por ejemplo, la dicha antes de trenes de mercancías y trenes de pasajeros). Sin embargo, con esta configuración, solo son necesarias dos definiciones de velocidad, y asignarlas doce veces por medio de sus identificadores (o atributos ID).

Sus atributos son los siguientes: *etcsTrainCategory*, *profileRef*, *status* y *vMax.* En él, los trenes del tipo ectsTrainCategory (siguiendo las normas de identificación del documento

[8] https://www.cit-rail.org/media/files/public/Freight/Circular_letter_23-2014_Appendix_1_List_of_codes.pdf

ERA_ERTMS_040001[9]) deben seguir la velocidad máxima vMax mientras estén en los perfiles de velocidad marcados por profileRef. El elemento Status da información sobre el estado de esta velocidad. Esta velocidad se debe expresar en kilómetros por hora.

Los únicos atributos opcionales de este elemento son status y etcsTrainCategory. Si estos se omiten, se entenderá que la restricción de velocidad máxima es válida para cualquier tren y cualquier estado. Asimismo, se podrá repetir este elemento con el mismo perfil varias veces, si este afecta a trenes diferentes. Para nuestro caso, solo se ha modelado un tipo de tren de pasajeros que cumpliría las mismas características, de modo que solo se han rellenado los elementos profileRef y vMax.

### 3.1.11 - trainRadio:

Toda infraestructura de ferrocarril lleva aparejada una comunicación por radio. La definición de las características de esta radio es vital para el buen funcionamiento del ferrocarril, así como para la seguridad del mismo. En este elemento se utilizan los siguientes atributos para dicha definición: *radioSystem, networkSelection, publicEmergency, broadcastCalls, textMessageService, directMode* y *publicNetworkRoaming.* Los valores para dichos atributos deben ser, cadenas de caracteres para los dos primeros, referentes al sistema de radio utilizados (como GSM-R) y el procedimiento de selección de red de la radio. Los demás atributos se rellenan con valores booleanos (es decir, "true" o "false", términos ingleses para verdadero o falso), para expresar si la radio tiene disponibles: llamadas de emergencia (*publicEmergency*), llamadas telefónicas (*broadcastCalls*), servicio de mensajes de texto SMS (*textMessageService*), modo directo de radio (*directMode*) o servicios de Roaming o servicio de itinerancia de datos (*publicNetworkRoaming*). Es recomendable rellenar todos los atributos si es posible.

### 3.1.12 - trainProtection:

Aquí se definirá una característica tan importante como son los medios de protección de trenes que se utiliza en las vías que utilizan la <infraAttributes> definida por medio de referencia de su ID. Esta protección se define por medio de dos atributos: la manera por la que se efectúa la comunicación entre tren y vía, almacenado en *medium* y la forma de supervisión que se lleva a cabo, con *monitoring*. Ambos atributos tienen una lista de posibles valores:

Para el atributo opcional medium: m*echanical* para tipo mecánico, *electric* para tipo eléctrico, *inductive* para inductivos, *magnetic* si es magnético, *optical* para sistemas ópticos, *radio* para sistemas por radiofrecuencia, *rail* para sistemas que se comuniquen por el carril, *cable* si la comunicación es por cable, o *none* si no hay comunicación para este tipo de protección.

Para monitoring por su parte, los posibles valores son: *intermittent* si es supervisión intermitente, *continuous* cuando la supervisión sea continua, o *none* si no hay protección de tren. Este atributo es opcional y, cuando no está se toma por defecto el valor *none.*

[9] https://www.era.europa.eu/sites/default/files/activities/docs/ertms_040001_etcs_variables_values_en.pdf
-Agencia para ferrocarriles de la Unión Europea, Mayo 2018-

### 4 - metadata:

Sobre metadata ya se hizo una descripción en el punto de explicación de los subesquemas, ya que este elemento funciona del mismo modo: el elemento "metadata" puede ser utilizado una vez, o ser omitido en el código. No tiene atributos ni elementos descendientes obligatorios aunque sí existen algunos con amplia utilización, que son <dc:creator>, <dc:date>, <dc:format>, <dc:identifier>, <dc:language>, <dc:source> y <dc:title>. Todos estos elementos, como se puede comprobar, comienzan por dc: dado que son elementos provistos por el Dublin Core Metadata[10]. Este elemento al completo es omitido en el ejercicio dado que la información que brinda no es necesaria ni útil en nuestro caso.

### 5 - operationControlPoints:

Este es un elemento contenedor de elementos <ocp> que referirán los puntos de control operacional. No contiene atributos. Es importante dado que éste albergará todos los elementos que controlan la circulación y serán necesarios para designar, por ejemplo, las estaciones, o los horarios más adelante.

### 5.1 - ocp:

Elemento identificador de Puntos o lugares de Control u Operación es decir, enclavamientos. Para su identificación, tiene varios elementos que se detallarán a continuación, y los siguientes atributos: *ID*, *code*, *name*, *description*, *xml:lang* (ya vistos), *timezone* guardando la zona horaria del lugar donde está la vía, *type* para indicar qué tipo nombre se ha dado a este OCP (sus valores son *operationalName*, *trafficName*, *localName* y *other:anything* según el nombre sea según aspectos operacionales, de trafico, nombre en la lengua local, u otros) y *parentOcpRef* guardando el enclavamiento padre de este enclavamiento.

Para la correcta consecución del proyecto, se entiende que cada una de las estaciones presentes en el tramo estudiado. De este modo es lógico hacerse una pregunta, ¿implica esto que cada estación tenga que ser un enclavamiento y viceversa? La respuesta es que no, obviamente el código ofrece libertad para modelar una estación como enclavamiento o no. Para esto se utiliza el atributo "parentOcpRef". El método a seguir es, definir un <ocp> para el enclavamiento y, después, definir tantos <ocp> como se necesiten, como dependiente del ocp que representa el enclavamiento. Dado que esto es complicado de visualizar, se da un ejemplo clarificante (omitidos los subelementos por optimización de espacio), donde el ocp_1 será el enclavamiento, y los otros dos serán apeaderos sin enclavamiento propio:

```
<ocp id='ocp_001' code='MGMS3728' name='ocp_1' description='Punto Control Operativo 1'
xml:lang='ES' timezone='Europe/Madrid' type='localName' />
<ocp id='ocp_002' code='MGXJ9505' name='ocp_2'  description='Punto Control Operativo 2'
xml:lang='ES' timezone='Europe/Madrid' type='localName' parentOcpRef ='ocp_001'/>
<ocp id='ocp_003' code='BLTR9425' name='ocp_3'  description='Punto Control Operativo 3'
xml:lang='ES' timezone='Europe/Madrid' type='localName' parentOcpRef ='ocp_001'/>
```

De este modo, queda indicado por la parte subrayada que los ocp 2 y 3 son dependientes del ocp 1.

10 http://dublincore.org

Asimismo, también es notable que cada uno de los ocp que se añadan en esta parte serán representados gráficamente en la vista de perfil de la vía, como se puede ver en la imagen de debajo. Así, se deduce que cualquier elemento que se necesite representar en esta vista, necesitará un elemento OCP.

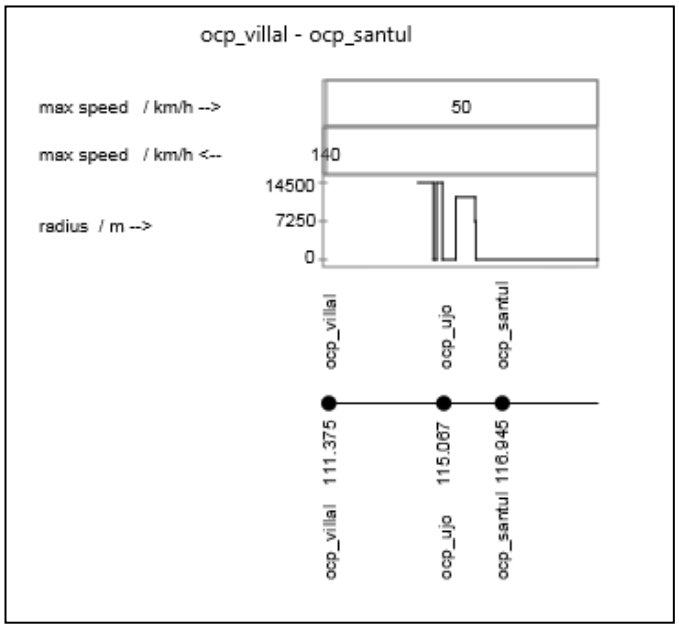


Fig. 4 Representación de OCP en vista gráfica.

### 5.1.1 - additionalName y 5.1.2 - any:

Ambos elementos funcionan exactamente de la misma manera que los expuestos anteriormente. Para más información, ver los puntos 1 - additionalName y 2 - any.

### 5.1.3 - area:

Referente a la zona geográfica en que el enclavamiento es responsable. Este elemento es bastante simple, sin descendientes y conteniendo solo tres atributos, que son *name*, *number* y *zip*. Estos atributos representan respectivamente el nombre de la región el número de la región y el código postal de la misma. Todos ellos son opcionales y serán enteros positivos en el caso de *number* y cadenas de caracteres en los otros dos. (Para versiones anteriores a la 2.2, el atributo zip será un entero positivo y obligatorio). En cada ocp del caso práctico, se ha utilizado una herramienta digital de internet para hallar el código postal a la que la estación pertenece. Del mismo modo, y con un objetivo meramente informativo, se ha rellenado el atributo number con las primeras tres cifras del código postal.

### 5.1.4 - geoCoord:

Este elemento es de extraordinaria utilidad para todo elemento en que esté incluido ya que posibilita la confección de mapas geográficos de las vías y sus elementos. Como este elemento se repite en varias ocasiones (como es el caso de additionalName, por ejemplo), se definirá en este punto y en los consiguientes se hará referencia a este punto, pues su definición es válida para todos los casos en los que se usa. Los atributos que se deben rellenar en este elemento son cuatro: *coord* se rellenará con los valores de las coordenadas, *extraHeight* contendrá la altura

geográfica del punto que representa, *epsgCode* es un URI () como referencia del sistemas de coordenadas utilizado en coord con un EPSG-Code y *heightEpsgCode* contendrá la misma información que epsgCode pero en referencia a la altura.

Este elemento está presente en multitud de elementos, de manera que se pueda especificar la posición de dichos elementos en la vista de mapa. Si esta información no se especifica, se el programa usado para la interpretación del código hace una aproximación mediante el atributo *pos* del elemento, haciendo una interpolación sencilla.

#### 5.1.5 - propEquipment:

El elemento propEquipment es un contenedor de datos de uno de los tipos que vienen a continuación. Estos datos serán principalmente dos: o bien una lista de banderas binarias que darán valor de verdadero o falso a ciertas propiedades o, como segunda posibilidad, una lista de referencias a los elementos track (vías) que pertenezcan al OCP. Si se elige este segunda manera, cada uno de los elementos definidos en las vías pertenecerán directamente al ocp que tenga dichas vías referenciadas.

##### 5.1.5.1 - summary:

Aquí estarán contenidos los elementos (representados por banderas, como se ha dicho antes verdaderas o falsas) que representan aproximadamente el equipamiento técnico del ocp. Sus atributos son *signalBox*, con valores de entre la lista [none, mechanical, electro-mechanical, electrical o electronical], *powerStation* será verdadero si el ocp tiene una central eléctrica, *hasHomeSignals* será verdadero si el ocp tiene señales de llegada a la estación, hasStarterSignals dirá si el ocp tiene señales de salida de estación, y hasSwitches definirá si hay aparatos o no. Este elemento no es compatible con el siguiente <trackRef>.

##### 5.1.5.2 - trackRef:

Esta es la segunda posibilidad de definición de elementos que equipan técnicamente a un OCP. Se deberá incluir un elemento por cada vía que esté incluida en el ocp. Para definir estos elementos, se tendrán que rellenar dos atributos: *sequence* (opcional) que establece un orden numérico para las vías y *ref* para el ID de la vía que se quiera incluir en este ocp. Así, cuando se incluya una sentencia tipo *<trackRef sequence='1' ref='vía_01'/><trackRef sequence='2' ref='vía_02'/>*, la vía que se nombró por el ID 'vía_01' será la primera vía de este ocp, seguida de la vía 'vía_02'.

Es importante volver a recalcar que, dentro de un ocp, se deberá elegir una de las dos modalidades (bien summary, bien trackRef), aunque es lógico que, si las vías están bien definidas, trackRef ya contiene toda la información necesaria para estos elementos, y la manera más lógica de proceder es referenciar los ocp y sus vías, por medio de trackRef. También hay que tener en cuenta que para la consecución de esta metodología, hay que definir bien las señales con el atributo function para determinar si dicha señal es de entrada o salida a una estación.

### 5.1.6 - propOperational:

Elemento con mucha importancia para recalcar el tipo de elemento ocp. Para esta definición, se necesitan cuatro atributos que son *operationalType* representando el tipo de punto de control que es (bien sea una estación "station", un Stop "stoppingPoint", un depósito de trenes [depot], una conexión entre dos vías paralelas [crossover], una unión o separación entre dos líneas [junction], un enclavamiento, normalmente con operarios, que asegure la distancia entre trenes de la misma vía [blockPost], un enclavamiento, normalmente sin operarios, que asegure la distancia entre trenes de la misma vía [blockSignal], un ocp que recoge y entrega carga a los trenes de mercancías pero sin control sobre operaciones [siding], o cualquier otro [other:anything]), seguido de el atributo *trafficType* con los valores de la siguiente lista [passenger, freight, shunting, other:anything] según si el tráfico será prevalentemente de pasajeros, mercancías, u otros; además se tendrán que rellenar los atributos *orderChangeable* con un verdadero o falso según si se puede cambiar la secuencia de los trenes, y *ensuresTrainSequence* como verdadero o falso según si se controla la secuencia del tren desde este ocp.

#### 5.1.6.1 - uptime:

<uptime> define las restricciones de circulación provenientes de las horas del día. Se pueden añadir tantos elementos como sean necesarios, y se deberán rellenar con los siguientes atributos: *from* para indicar el inicio del periodo con un dato de tipo xs:time[11], *until* para indicar el final del periodo con el mismo formato xs:time, y *mode* con uno de los siguientes valores [manned, unmanned, off, other:anything] dependiendo de si el ocp está atendido por operarios, no atendido, o sin funcionamiento, u otro modo. De este modo, si el ocp está sin funcionamiento hasta las 6 de la mañana, atendido de 6 a 18 y sin atender de 18 a 21, y sin funcionamiento de 21 a 24 horas, se deben rellenar cuatro elementos con los valores siguientes:

```
<uptime from='00:00:00' until='05:59:59' mode='off'/ >
<uptime from='06:00:00' until='17:59:59' mode='manned'/ >
<uptime from='18:00:00' until='20:59:59' mode='unmanned'/ >
<uptime from='21:00:00' until='23:59:59' mode='off'/ >
```

### 5.1.7 - propOther:

Elemento para propiedades adicionales. Sus atributos son *frontier*, *chargeFrontier* y *status*. Los dos primeros son booleanos, denotando si el ocp hace de frontera, o si contiene una frontera. El atributo status se refiere a la disponibilidad y sus valores son [conceptual, planned, operational, disabled, closed, other:anything], dependiendo de si el ocp es según su estado conceptual, planeado, operacional, si está no utilizable, o si está cerrado.

#### 5.1.7.1 - additionalName:

Nombre adicional que se le puede aplicar, ya visto en el punto 1

[11] Este tipo será horas minutos y segundos, en el formato hh:mm:ss

### 5.1.8 - propService:

Elemento que especifica los servicios que están disponibles en este ocp. Todos los atributos son booleanos y opcionales. En el ejemplo práctico se han establecido de manera aleatoria. Los atributos son: *passenger*, *service*, *ship*, *bus*, *airport*, *tariffpoint*, *goodsLoading*, *goodsSiding*, *goodsIntermodal*, o *goodsMarshalling*; según si el ocp tiene servicio para: pasajeros, servicios adicionales, intercambios con barcos, autobuses, aeropuertos, tickets, carga y descarga de mercancías, vía secundaria para intercambio de bienes, intercambio de bienes entre tren y camión, o reordenación de trenes.

### 5.1.9 - designator:

Para cualquier abreviatura, designación o nombre secundario que tenga el elemento ocp, se puede utilizar en el elemento additionalName, asimismo, en este elemento designator se puede detallar claves externas que llamen a este ocp. Este elemento no es de extremada utilidad para la codificación de una red simple, pero cobrará algo más de sentido cuando pueda haber referencias entre las redes. Sin embargo, es un elemento que requiere codificación externa de otros elementos en un archivo XML y, por su complejidad no ha sido usado en el caso práctico. Sus atributos son *register* para definir un índice de los registros desde los que el ocp puede ser llamado, o a los que el ocp puede llamar; *entry* albergando la clave externa del OCP perteneciente a dicho registro; *startDate* y *endDate* para señalar la validez temporal de este elemento con un tipo de dato xs:date. Los dos primeros atributos son requeridos para este elemento, mientras que los últimos son opcionales.

### 6 - trackGroups:

Después de la definición de las vías, se pueden agrupar las mismas en líneas, para esto, el elemento trackgroups contiene elementos tipo <line>, que se verá más adelante. Este elemento no tiene atributos. Las vías, sin embargo, se pueden agrupar por vías pertenecientes : una línea, un área controlada localmente, una estación o un plan de prioridades.

### 6.1 - line:

Elemento que modela una línea de ferrocarril. Se puede orientar de varias maneras, aunque la más obvia y lógica es agrupar las vías por líneas, y sentidos de circulación (si están definidos). Así, los atributos que tiene este elemento son: *id*, *code*, *name*, *description*, *xml:lang*, *type*, *infrastructureManagerRef*, *lineCategory*. Type y lineCategory hacen referencia a qué tipo de vía es. El primero contiene un valor de [mainLine, branchLine, secondaryLine, other:anything] para referirse a líneas principales, ramificaciones de línea (uniones entre líneas por ejemplo), líneas secundarias, y otros respectivamente. El segundo de ellos es uno valor de acuerdo con la norma EN-15528 [A, B1, B2, C2, C3, C4, D2, D3, D4, D4xL, E4, E5] infrastructureManagerRef llevará el ID del gestor de la infrastructura, declarado previamente en el subesquema Metadata (common).

### 6.1.1 - additionalName, 6.1.2 - any:

Elementos ya vistos en elementos anteriores.

### 6.1.3 - states:

Contenedor de elementos <state> sin atributos. Sirve para definir partes de la infraestructura como habilitadas o deshabilitadas.

#### 6.1.3.1 - state:

Cada elemento state representa un estado de un parte específica de la infraestructura. Para ello, se tienen los siguientes atributos: *disabled* (booleano que marca verdadero si está deshabilitado) , *operatingPeriodRef* (referencia al ID del periodo operativo que define las fechas de este estado), *remarks* (variable string que permite una más larga definición de states).

Si no hay ningún elemento state definido, se entenderá que los valores de la infraestructura son válidos sin ninguna restricción.

#### 6.1.3.1.1 - from, 6.1.3.1.1.1 - geoCoord, 6.1.3.1.2 - to, 6.1.3.1.2.1 - geoCoord:

<from> y <to> permiten limitaciones geográficas al elemento donde se encuentren (útil por ejemplo, para partes que estén en mantenimiento). Sus atributos son *ocpRef*, *pos* y *absPos.* ocpRef tendrá el ID del OCP al que se refiere, y pos y absPos serán valores que marcan, respectivamente, el kilómetro en el que se encuentra, contando desde el principio de la vía, y el kilómetro global desde el punto de kilometraje 0. Estos dos atributos están incluidos en muchos de los elementos siguientes, pero se darán ya por explicados en este punto. Si tenemos una vía que empieza en el km. 120, y una señal que está en el km. 150 los valores de pos y absPos para esa señal serán pos='30' absPos='150'.

### 6.1.4 -trackRef:

Este elemento tendrá el mismo funcionamiento que el elemento 5.1.5.2 - trackRef. Referirá a cada una de las vías que se incluyen en la línea, y su secuencia.

### 7 - tracks:

Este es, posiblemente, el elemento más importante de toda la definición de la infraestructura, dado que contiene elementos track, que definirán las vías en sí mismas. No tiene atributos.

### 7.1 - track:

El elemento track es la representación de una vía. esta vía se puede delimitar de varias maneras (es decir, hacer corresponder un elemento Track a un circuito de vía, o a la parte de vía contenida en un OCP, etc.) sin embargo, se ha entendido que lo más lógico es definir un elemento Track por cada parte de vía contenida en un OCP.

El elemento Track, contiene los atributos *id*, *code*, *name*, *description*, *xml:lang*, *type* y *mainDir*. Type se refiere a un elemento de [mainTrack, secondaryTrack, connectingTrack, sidingTrack, stationTrack, other:anything] correspondientes a una vía principal, secundaria, de conexión, lateral, de estación, u otro tipo. mainDir, a su vez, definirá la dirección en la que su uso será preferible.

<track> contiene muchos elementos, que son:

#### 7.1.1 - additionalName, 7.1.2 - any y 7.1.3 - states

Elementos ya estudiados.

#### 7.1.4 - trackDescr

Elemento no utilizado ya que va a ser marcado como obsoleto

#### 7.1.5 - trackTopology:

Elemento que marcará los principios de la forma topológica de la vía. Contenedor, sin atributos.

##### 7.1.5.1 - borders:

Contenedor de elementos border, sin atributos.

###### 7.1.5.1.1 - border:

Elemento que define una frontera, por ejemplo entre zonas de tarificación, o entre países o comunidades autónomas. Sus atributos son: *id*, *code*, *name*, *description*, *xml:lang*, *pos*, *absPos*, *dir* (definición de la validez de la frontera, siendo sus valores none si no hay restricción, up si el sentido de validez de la frontera es desde <trackBegin> hacia <trackEnd>, down para el sentido contrario, both para los dos sentidos, y unknown si hay un sentido, pero no se sabe cuál es). Por último el atributo *type* (obligatorio), denota si el elemento border es tarifario, de área, de estado, de país, o de estación ([tarif, area, state, country, station, other:anything]).

Este elemento contiene los subelementos additionalName, any, geoCoord y states que ya han sido explicados anteriormente.

##### 7.1.5.2 - connections:

Elemento que contendrá los subelementos que definirán las conexiones entre diferentes vías. Tendrá dos tipos de conexiones posibles: switch para un aparato de vía normal, o crossing para los cruces entre vías (cruces tipo rombo o diamante).

###### 7.1.5.2.1 - crossing:

Elemento que sirve para definir un cruce entre vías de tipo diamante, con los atributos: *id*, *code*, *name*, *description*, *xml:lang*, *pos*, *absPos*, *ocpStationRef* para designar el ocp que controla este cruce, *controllerRef* para referenciar al controlador de la parte de vía donde se encuentra este cruce, *trackContinueCourse* definiendo la posición del cruce en la que continúa la vía normalmente, *trackContinueRadius* indicando el radio de la vía en ese punto, *normalPosition* mostrando la posición normal (cerrada) del cruce(entre recto, a la izquierda, a la derecha, u otros [straight, left, right]) *model* denotando el modelo de cruce que es, *length* con la logitud del mismo en metros, y *type* seleccionado entre [simpleCrossing, simpleSwitchCrossing o doubleSwitchCrossing] según si es un cruce simple, un cruce simple con aparato o un cruce doble con aparato.

Este elemento contiene los subelementos additionalName, any, geoCoord y states que ya han sido explicados anteriormente[12].

#### 7.1.5.2.1.1 - connection:

Elemento que señala qué vías están conectadas con este elemento <crossing> además de la misma en la que se está describiendo el elemento. Se podría considerar que este elemento modela un desvío.

Sus atributos son: *id*, *ref* con la referencia del otro elemento connection al que se une, *orientation* indica si el elemento es entrante, saliente o perpendicular según el sentido creciente de kilometraje con los valores [incoming,outgoind,rightAngled, unknown y other:anything], *course* atributo que marca si el desvío se encuentra recto, a la derecha o a la izquierda según [straight, right, left, other:anything] *radius* con el valor en metros de la curvatura del elemento que se desvía, *maxSpeed* definiendo la velocidad máxima posible en el desvío, *passable* (atributo muy importante, puesto que define si el cruce se puede rebasar, por defecto se tomará que sí se puede si este elemento no se define).

#### 7.1.5.2.2 - switch:

Elemento que modela un aparato simple de desvío. Sus atributos son exactamente los mismos que en el elemento crossing y se rellenan de la misma manera. Asimismo, también contiene un elemento <connection> que también tiene las mismas propiedades que el anterior elemento connection.

#### 7.1.5.2.2.5 - connection:

Elemento ya definido.

#### 7.1.5.3 - crossSections:

Contenedor de elementos <crossSection> sin atributos

#### 7.1.5.3.1 - crossSection:

Este elemento se utiliza para dar una unión entre la vía y los diferentes OCP que coexisten a lo largo de ella. Sin embargo, su utilidad queda reducida a los casos en los que haya OCPs que no hayan sido definidos dentro del resto del código de la vía. Sus atributos son: *id*, *code*, *name*, *description*, *xml:lang*, *pos*, *absPos*, *type* que tendrá uno de los valores [station, block, autoblock u othre:anything] según si el tipo de OCP sea una estación, un bloqueo o un bloqueo automático; *ocpRef* para definie el OCP al que se refiere (se necesitará un elemento crossSection por cada OCP que se quiera añadir) y el atributo *dir* señalando si este OCP afecta en ambas direcciones, en la dirección de kilometraje creciente, o decreciente, o en ninguna [both, up, down, none].

[12] Dado que no han sido especialmente utilizados en el caso práctico y para evitar ser repetitivos en esta explicación, se van a pasar por alto estos cuatro elementos salvo mención específica en todos los próximos elementos. Para obtener información sobre su existencia o no en un elemento, basta con buscar información en los esquemas que definen dichos elementos, o en las páginas de apoyo de RailML(R).

#### 7.1.5.4 - mileageChanges:

Elemento contenedor de mileageChange sin atributos.

##### 7.1.5.4.1 - mileageChange:

Se utiliza para modelar cambios de kilometraje (por ejemplo, cuando dos vías confluyen, una deja su kilometraje de lado para adoptar el de la otra). Para ello se definirán los atributos *id*, *code*, *name*, *description*, *xml:lang*, *dir*, *absPosIn*, *type*, *absPos*, *pos*, *absDir*.

absPosIn define en metros la posición del punto en que cambia el kilometraje, según la vía que "pierde" su kilometraje; pos hace lo propio, pero según la vía que conserva el kilometraje. absPos, a diferencia de otros elementos en que está presente, sólo denota cuál es el kilómetro oficial de cambio de este nuevo kilometraje.

#### 7.1.5.5 - trackBegin y 7.1.5.6 - trackEnd:

Elementos imprescindibles sin los que ningún elemento en la definición de vías estaría referenciado ni definido. Estos elementos definen el punto donde comienza y acaba la vía y, lo que es más importante, definen un kilometraje que servirá a posteriori para posicionar cualquier elemento en ellas. Su definición es bastante sencilla por obvia, ambos elementos tienen los ya conocidos atributos *id*, *pos*, *absPos*, mientras que tackBegin contiene también el atributo *absDir* que marcará la dirección del kilometraje en referencia a la vía.

Es destacable que, como se ha dicho al principio de este tema, el orden en el que se presentan estos elementos no es necesariamente el orden en el que deben aparecer en el archivo. De hecho, en este caso, trackBegin debe ser siempre el primer elemento dentro de <trackTopology> mientras que trackEnd debe ser el segundo.

Ambos elementos tienen el elemento descendiente que es muy importante en estos casos ya que da al código la capacidad de entender que dos vías están conectadas, o que, por ejemplo, una vía tiene una terminación muerta. Este elemento será uno y solo uno de los siguientes posibles: <connection>, <bufferStop>, <openEnd>, <macroscopicNode>. A continuación se procede a explicar cada uno de ellos.

##### 7.1.5.6.1 - connection:

En este elemento se define el principio o final de vía como una conexión con otro punto (a) de la red (ya sea el final o el principio de otra vía, o un aparato de vía -*switch*- definido). Siendo así, el elemento queda completamente definido con dos atributos: *id*, y *ref.* El atributo ref tendrá el ID del otro elemento <connection> con el que está enlazada la vía, en este caso, el ID del punto (a).

##### 7.1.5.6.2 - bufferStop:

Este elemento expone que la vía comienza o termina en una parada de final de vía. Para definir este elemento sólo hace falta rellenar el *id*, aunque también se pueden rellenar de manera opcional *code, name, description* y *xml:lang.*

#### 7.1.5.6.3 - openEnd:

Si este elemento se incluye como descendiente de un elemento comienzo o final de vía, se estará definiendo dicho final o principio como algo desconocido por el programador, es decir, no se sabe qué tipo de final de vía es. Sus atributos son los mismos que en el caso de bufferStop

#### 7.1.5.6.4 - macroscopicNode :

Esta es una manera bastante útil de definir una conexión entre vías sin tener ningún respeto ni referencia a la disposición física de las mismas. Esto puede ser útil en ciertas ocasiones, como en grandes estaciones en las que no se quiera entrar al detalle de cómo se organizan las vías de la propia estación, si no solo las entradas y salidas. Tiene dos atributos que son *ocpRef* y *flowDirection* que será [in, out, both, unknown] si el tráfico entra en el nodo, sale, entra y sale según el caso, o no se sabe.

### 7.1.6 - trackElements:

Aquí se definirán los elementos físicos de la vía, desde puentes, hasta radios de curvatura. Es un elemento contenedor sin atributos. En los elementos de contenidos por éste, es muy común que haya un elemento contenedor y varios elementos del tipo concreto dentro. Esto es rápidamente reconocible porque el elemento contenedor va siempre en plural. Para evitar reiteraciones que no aportan valor, se entiende que ninguno de estos elementos contiene atributos salvo que se indique lo contrario. En todos ellos, los elementos 'en singular' se pueden repetir tantas veces como sea necesario, por lo que si hay cuatro túneles en una vía, la forma correcta de expresarlo será:

```
<tunnels>
        <tunnel id='1'/>
        <tunnel id='2'/>
        <tunnel id='3'/>
        <tunnel id='4'/>
</tunnels>
```

Además, muchos de estos elementos representan cambios. Esto es así porque, si no se dice nada al respecto, se toma siempre el valor definido en los atributos del ocp, o de los atributos de la vía según el caso. Esto significa que, si no hay ningún elemento " clearanceGaugeChange" se dará por hecho que la distancia libre alrededor del tren será siempre la misma, y siempre igual a la definida en el los atributos de la infraestructura.

### 7.1.6.1 - axleWeightChanges:

#### 7.1.6.1.1 - axleWeightChange:

Estos son los puntos de la vía donde el peso máximo permitido cambia. Se definen mediante *id*, *code*, *name*, *description*, *xml:lang*, *pos*, *absPos*, *dir*, *value* y *meterload.* Todos los atributos se han visto anteriormente salvo value, que representa el valor en toneladas métricas del nuevo límite de peso por eje, y meterload que define el mismo límite pero por metro lineal de vehículo en toneladas partido por metro.

En todos los elementos que estén presentes en <trackElements>, como es el caso, serán obligatorios los atributos *id*, y *pos*, como mínimo.

#### 7.1.6.2 - bridges:

#### 7.1.6.2.1 - brigde:

*NOTA: debido a un error propagado al a hora de definir los esquemas, el elemento que por lógica debería ser bridge (puente en inglés) es, sin embargo, brigde (intercambiando g y d). Esto no se modifica para evitar problemas de compatibilidad con códigos generados anteriormente.*

Representa un puente de cualquier tipo (puente, acueducto, etcétera) por el que pasan las vías. Para su definición, se pueden rellenar los atributos: *id*, *code*, *name*, *description*, *xml:lang*, *pos*, *absPos*, *dir*, *meterload, kind* y *length.* kind representa el tipo de puente por medio de un tipo string; y length representa un número real de seis cifras decimales, con la longitud del puente en metros. Si este puente existe a cause de algún elemento que cruza las vías (como un río, o una carretera), este elemento se puede modelar con los elementos siguientes:

#### 7.1.6.2.1.5 - crossedElements:

#### 7.1.6.2.1.5.1 - crossedElement:

Como se ha dicho anteriormente, este elemento resulta útil a la hora de definir y modelar elementos que crucen la vía por debajo del puente definido (importante, solo los elementos que crucen por dicho puente, puesto que habrá otro elemento para los cruces a nivel).

Sus atributos son *id*, *code*, *name*, *description*, *xml:lang*, *type* y *pos*. Como única novedad, el atributo type señala qué tipo de elemento cruza las vías, con un tipo de dato "other:anything" que se genera con la cadena "other:" seguida de al menos dos caractéres.

#### 7.1.6.3 - clearanceGaugeChanges:

#### 7.1.6.3.1 - clearanceGaugeChange:

Elemento que define la posición en la que hay un cambio de la distancia libre alrededor del tren. Sus atributos son *id*, *code*, *name*, *description*, *xml:lang*, *pos*, *absPos* y *dir* [none, up, down, both o unknown]. Si este elemento se utiliza, es elemental (aunque no obligatorio) rellenar el siguiente elemento descendiente de este, ya que da información sobre el estándar que rige estas medidas.

#### 7.1.6.3.1.5 - clearanceGauge:

Ya explicado anteriormente. Su atributo es *code* y es obligatorio. Su valor será el código que rige estas distancias libres. En el ejercicio práctico se entiende que toda la vía se mantiene con el código "GEB16" y por ello no hay ningún elemento de tipo clearanceGaugeChanges en él.

#### 7.1.6.4 - electrificationChanges:

##### 7.1.6.4.1 - electrificationChange:

Como su nombre indica, esta será la representación en código de los puntos de la vía donde la electrificación de la misma cambie de alguna manera. Además de los atributos comunes en este tipo de elementos (*id*, *code*, *name*, *description*, *xml:lang*, *pos*, *absPos* y *dir*) también podemos encontrar los atributos *type*, *voltage*, *frequency*, *vMax*, e *isolatedSection*, que contendrán respectivamente: el tipo de entrega de electricidad ([none, overhead, 3rdRail, sideRail, other:anything] según sea ninguno, con catenaria, con tercer raíl, con raíl lateral u otros tipos), el voltaje en voltios (si no se da, se toma por defecto 0), la frecuencia en herzios (por defecto, 0), la velocidad máxima en km/h, y un valor booleano que indica si hay una sección aislada o zona muerta.

###### 7.1.6.4.1.1 - maxTrainCurrent:

Se dará en este elemento la máxima corriente que se permite. Sus atributos son *maxCurrent* (máxima corriente permitida en amperios), *type* uno de los elementos de la lista, dependiendo de si se refiere a corriente estática o dinámica[*standstill* o *driving*]y *validFor* [*train* o *pantograph*] según si la limitación se refiere al tren o al pantógrafo.

#### 7.1.6.5 - gaugeChanges:

##### 7.1.6.5.1 - gaugeChange:

En él se puede definir el punto donde la distancia entre vías cambia. Para definir este elemento, además de los atributos comunes a los anteriores (*id*, *code*, *name*, *description*, *xml:lang*, *pos*, *absPos* y *dir*), se debe definir el atributo *value* con un dato decimal que representa la distancia entre vías en milímetros.

#### 7.1.6.6 - geoMappings:

##### 7.1.6.6.1 - geoMapping:

Este elemento da la posibilidad de mapear una posición de la vía según sus coordenadas geográficas. Para esto es vital tener por un lado la posición en la vía del punto que se quiera definir (por su kilometraje), y sus coordenadas geográficas. Para las coordenadas, se utilizará el subelemento <geoCoord> que será obligatorio.

Sus atributos son los que se han visto anteriormente que están presentes en prácticamente todos los elementos de <trackElements>: *id*, *code*, *name*, *description*, *xml:lang*, *pos*, *absPos* y *dir.*

#### 7.1.6.7 - gradientChanges:

##### 7.1.6.7.1 - gradientChange:

Este elemento modela la información necesaria para tener claros los puntos en los que hay cambio de pendiente. Estos elementos son de alta importancia porque son los que definirán el perfil de inclinaciones de la vía. Para su definición son necesarios los atributos *id*, *code*, *name*, *description*, *xml:lang*, *pos*, *absPos*, *dir,slope, transitionLength* y *transitionRadius.* Estos tres

últimos elementos se completarán con: un número de con tres decimales con el valor de la pendiente en pulgadas por milla (obligatorio); número con seis decimales que da la longitud en metros de la transición; y un número de seis decimales que da el valor en metros del radio de la curvatura que tiene dicha transición.

#### 7.1.6.8 - levelCrossings:

#### 7.1.6.8.1 - levelCrossing:

Como se dijo anteriormente, existe un elemento que da la posibilidad de modelar cualquier cruce a nivel que esté presente en la vía. Este elemento se utilizará prioritariamente para pasos a nivel, pero también puede ser utilizado para otros, pero siempre será utilizado para elementos que no sean vías de ferrocarril (para eso existe el elemento <crossing>. Para su correcta definición, este elemento alberga los atributos siguientes: *id*, *code*, *name*, *description*, *xml:lang*, *pos*, *absPos*, *dir, ocpStationRef* (id del elemento <ocp> asociado), *controllerRef* (id del elemento <controller> asociado), *length* (longitud en metros de la longitud de la sección que cruza la vía según la dirección de la vía), *angle* (valor con tres decimales que representa el ángulo en grados que forman la vía con el elemento que cruza, teniendo que ser un valor entre 0 y 90), *protection* (string que permite informar sobre el equipamiento de protección que hay en el cruce). Al igual que en el elemento <brigde>, este elemento se completan con los <crossedElements> para definir

#### 7.1.6.8.1.5 - crossedElements:

#### 7.1.6.8.1.5.1 - crossedElement:

Visto anteriormente.

#### 7.1.6.9 - operationModeChanges:

#### 7.1.6.9.1 - operationModeChange:

El punto en el que cambian los modos de operación (ejecutivo o legislativo) de un OCP que esté presente en la vía que se está definiendo. Este elemento puede estar incluido tantas veces como sea necesario, y se define con los atributos *id*, *code*, *name*, *description*, *xml:lang*, *pos*, *absPos*, *dir, modeLegislative* (obligatorio)*, modeExecutive* (obligatorio) y *clearanceManaging.* Los tres últimos atributos contendrán el modo legislativo (string), le modo ejecutivo (string) y la manera en la que se controla la distancia entre vías.

#### 7.1.6.10 - ownerChanges:

#### 7.1.6.10 - ownerChange:

Este elemento se utilizará para definir el punto en el cual el dueño de la infraestructura cambia. En él se incluyen los valores del nombre del dueño en un dato string en el atributo *ownerName* (obligatorio), el código UIC de este dueño con un tipo entero positivo en el atributo *uic-no*, y la referencia al gestor de la infraestructura en esta nueva parte (cambie o no) por medio de su ID en el atributo *infrastructureManagerRef*. Además de estos, también se encuentran presentes los atributos ya conocidos *id*, *code*, *name*, *description*, *xml:lang*, *pos*, *absPos* y *dir.*

En el caso práctico que se ha estudiado, se entiende que el dueño de la infraestructura no cambia durante todo el recorrido de la parte estudiada.

#### 7.1.6.11 - platformEdges:

#### 7.1.6.11.1 - platformEdge:

Elemento muy importante que se utiliza para definir las características físicas de los andenes de una estación, o apeadero. Este elemento, junto con el elemento <ocp> pueden definir por completo una estación en una primera instancia. Sus atributos son atributos *id*, *code*, *name*, *description*, *xml:lang*, *pos*, *absPos*, *dir, ocpRef, length* (para la longitud en metros del andén en el sentido de la vía), *height* (definiendo la altura del andén en milímetros), *side* (con los valores [right, left] en función de si el andén está a la derecha o a la izquierda de la vía tomando como referencia el sentido creciente del kilometraje), *parentPlatformEdgeRef* (atributo en que se puede referenciar a otro andén al que este pertenece, esto se puede utilizar por ejemplo para modelar partes diferentes del andén en que, por ejemplo, haya diferentes alturas).

#### 7.1.6.12 - powerTransmissionChanges:

#### 7.1.6.12.1 - powerTransmissionChange:

Este elemento da la posibilidad de definir más de un tipo de transmisión de potencia del tren a la vía. Este elemento se define exactamente de la misma manera que el elemento 3.1.9 - powerTransmission, con la adición de los atributos típicos de estos elementos, de modo que los atributos totales son: *id*, *code*, *name*, *description*, *xml:lang*, *pos*, *absPos*, *dir, type* y *style.*

#### 7.1.6.13 - radiusChanges:

#### 7.1.6.13.1 - radiusChange:

Otro elemento vital para la correcta definición de la vía, puesto que esta es la información referente a las curvas que haya en el trazado de la vía en estudio. Dada la importancia de este elemento, se ha incluido en todas las vías del trabajo práctico.

Para su correcto funcionamiento, son necesarios varios atributos, como son: *id*, *code*, *name*, *description*, *xml:lang*, *pos*, *absPos*, *dir, radius, superelevation* y *geometryElementDescription.*

El atributo radius da información sobre el radio exacto de la curvatura de la vía en metros. Este debe ser positivo para las curvas a derechas según el sentido de kilometraje creciente, y negativo para las curvas a izquierdas. Aunque técnicamente lo correcto sería dar un valor infinito al radio de las partes rectas, para facilidad de interpretación y cálculo por la máquina, se toma radio igual a cero como el estándar que define las partes rectas del trazado.

Por su parte, superelevation da información sobre el peralte en milímetros, mientras que el atributo geometryElementDescription nos da información sobre la geometría de la curva, con los siguientes valores posibles:

- TS_cubicParabola transición a una curva de tipo parábola de tercer grado (cúbica)
- TS_parabola4 transición a una curva de tipo parábola de cuarto grado

- TS_clothoide transición a una curva de tipo clotoide
- TS_WienerBogen transición a curva de tipo Wiener Bogen
- TS_BlossBogen transición a curva de tipo Bloss Bogen
- TS_Sinusoide principio de transición a curva de tipo sinusoide
- TS_Cosinusoide Comienzo de transición cosinusoide
- SC para definir el final de una curva o su transición (recomendada para las partes rectas)

De este modo, la curva existente en la vía de la imagen a continuación tendría que quedar modelada de la siguiente manera:

```
(...)<trackElements>
        <radiusChange id="curva1" pos="123.45" radius="1000"
        geometryElementDescription="TS_clothoide" />
        <radiusChange id="curva2" pos="234.56" radius="0"/>
</trackElements>(...)
```

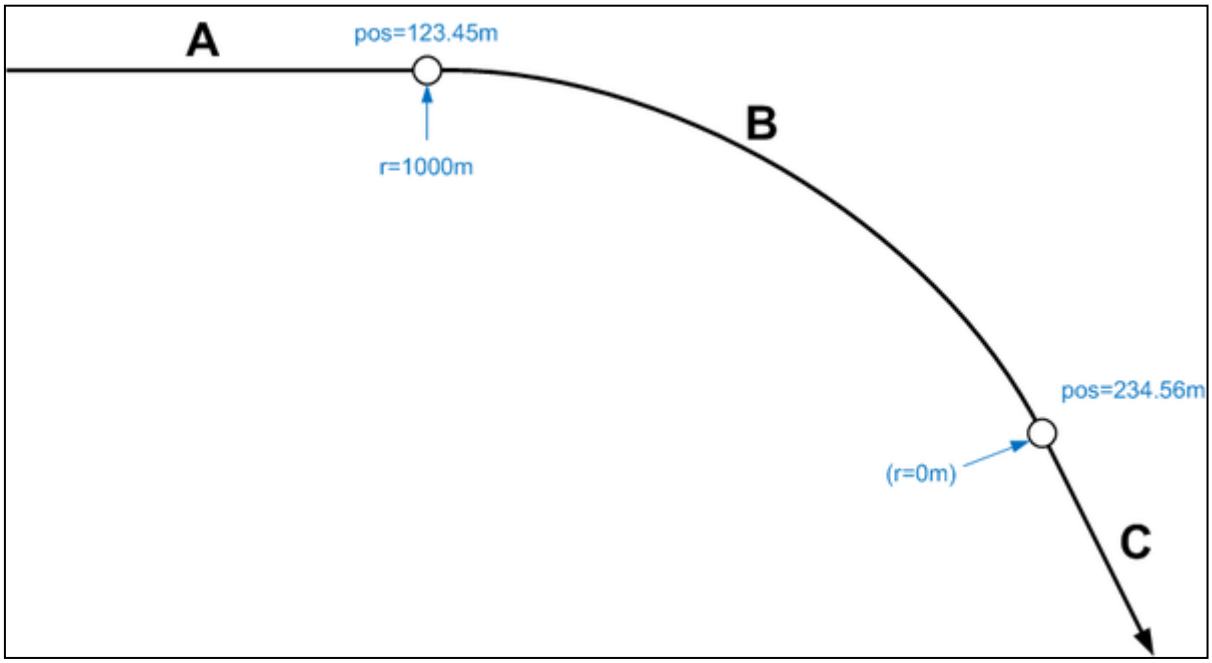


Fig. 5 Planta de una curva genérica definida por el código presentado anteriormente [13]

### 7.1.6.14 - serviceSections:

### 7.1.6.14.1 - serviceSection:

Elementos creados para poder definir la conexión entre una vía de ferrocarriles y una vía de servicio (por ejemplo, en estaciones).

Sus atributos son *id*, *code*, *name*, *description*, *xml:lang*, *pos*, *absPos*, *dir,ocpRef, length, height, side* definiendo el lado al que se encuentra según kilometraje ascendente,

[13] https://wiki.railml.org/index.php?title=IS:radiusChange

*parentServiceSectionRef* para relacionar el elemento serviceSection que agrupa a varios elementos de este tipo; y varios atributos booleanos: *ramp, maintenance, loadingFacility, cleaning, fueling, parking* y *preheating* que definen si el servicio está equipado con rampa, servicios de mantenimiento, equipamiento para carga, sistemas de limpieza, servicio de repostaje, aparcamiento, o sistemas de precalentamiento.

#### 7.1.6.15 - speedChanges:

#### 7.1.6.15.1 - speedChange:

Otro elemento importante, que modela los cambios de velocidad máxima que se experimentan en una vía. Este elemento es necesario incluso aunque se modelen los perfiles de velocidad, porque con este elemento se asignan dichos perfiles. También, se puede definir una velocidad que no esté contemplada en los perfiles de velocidad, pero que tenga las mismas características que uno de los perfiles. Así, *id*, *code*, *name*, *description*, *xml:lang*, *pos*, *absPos*, *dir, etcsTrainCategory* (código que define el tipo de tren a que afecta esta velocidad máxima si no afecta a todos), *profileRef* (ID del elemento profile que define las propiedades del campo de velocidades), *status* (string con el estado de la vía a la que afecta esta velocidad), *vMax* (velocidad máxima a la que se refiere este speedChange), *trainRelation* (definición que establece si el cambio de velocidad es efectivo desde el comienzo del tren, la mitad del mismo, o el final de este [headOfTrain, midOfTrain, endOfTrain]), *mandatoryStop* (verdadero o falso según haya una parada obligatoria o STOP [true, false]), *signalised* ([true o false] dependiendo de si el cambio de velocidad está señalizado físicamente de manera específica junto a la vía )

#### 7.1.6.16 - trackConditions:

#### 7.1.6.16.1 - trackCondition:

Elemento que contiene la información relativa al cambio de condiciones de la vía. Bastante útil para modelar trabajos de mantenimiento, o clausuras de la vía por cualquier otro motivo. Para completar ese modelado, se necesitan los atributos que se detallan a continuación: *id*, *code*, *name*, *description*, *xml:lang*, *pos*, *absPos*, *dir, length* y *type.* Para el atributo type, se puede elegir un valor de la lista siguiente:

- *nonStoppingArea* para un área en la que no se puede parar
- *lowerPantograph* para designar una zona en la que haya que bajar el pantógrafo
- *mainPowerSwitchOff* si se quiere definir una zona con la funete principal de energía sin suministro
- *radioHole* en una zona en que haya un agujero en el terreno
- *airTightness* si se debe indicar algo referente a la hermeticidad del tren
- *noRegenerativeBrake* para indicar que no se debe usar el freno regenerativo
- *noEddyCurrentBrake* cuando se deba anular el freno de corrientes parásitas
- *noMagneticShoeBrake* si se debe anular el freno de zapata magnética
- *other:anything* para cualquier otra indicación

#### 7.1.6.17 - trainProtectionChanges:

#### 7.1.6.17.1 - trainProtectionChange:

Este elemento será usado en caso de que los sistemas de protección del tren cambien. La forma de modelado es simple, se da un elemento <trainProtectionChanges> por cada punto en el que cambie algo. Si, por otro lado, cambia más de una cosa en un punto, será necesario un solo elemento de este tipo, indicando con él cuáles son los nuevos parámetros. Tiene los atributos que se detallan: *id*, *code*, *name*, *description*, *xml:lang*, *pos*, *absPos*, *dir, medium* y *monitoring.* Los dos últimos atributos se modelan de la misma manera que se vio en el punto 3.1.12 trainProtection.

#### 7.1.6.18 - tunnels:

#### 7.1.6.18.1 - tunnel:

Elemento que define la existencia de túneles en el recorrido de la vía. Igual que los anteriores, puede ser utilizado tantas veces como sea necesario.

En muchos casos, se puede dar que un mismo túnel contenga una pareja de vías (o más). Este caso se modela con un elemento túnel en cada vía, teniendo en cuenta que, aunque se trate del mismo túnel, los atributos ID de ambos deben ser diferentes aunque los nombres, códigos y descripciones pueden ser las mismas. Una posibilidad obvia es nombrar los ID de ambos completándolos con el número de vía al final. Es decir, si tenemos dos vías (vía 0132 y vía 7045) dentro de un túnel (túnel de Etsii), una vía posible y con fácil reconocimiento para formar los ID sería ID='tun_etsii_0132' y ID='tun_etsii_7045'. Como ya se explicará en el capítulo de buenas prácticas, este es un método excelente para evitar organizar los ID y evitar duplicarlos (lo que daría problemas a la hora de obtener la interpretación del código).

#### 7.1.6.18.1.5 - crossedElements:

#### 7.1.6.18.1.5.1 - crossedElement:

Al igual que el elemento brigde, se puede completar el elemento tunnel con <crossedElements> de manera que se conozca el elemento que cruza (si lo hay) por encima del túnel. Este elemento ya se ha estudiado anteriormente.

#### 7.1.7 - ocsElements:

Otro elemento contenedor sin atributos. En este caso, diseñado para contener los elementos de la vía que están dirigidos a sistemas de operación y control de vías. Sus elementos se detallan a continuación.

#### 7.1.7.1 - signals:

Elemento contenedor de todo lo referido a señalización. Este se divide en dos subelementos:

#### 7.1.7.1.1 - signal:

Cualquier señal, sea luminosa o no, se representará por medio de este elemento. Esto significa que se utilizará un elemento <signal> por cada una de las señales que haya en la vía que se está

definiendo. Dado que es posible que varias señales actúen de una forma concreta, dependiendo las unas de las otras aunque no estén en la misma vía, existe un elemento al mismo nivel, que se verá después (<signalGroups>) que agrupará todas las señales de un mismo tipo.

El elemento <signal> tiene atributos, y son los siguientes: *id*, *code*, *name*, *description*, *xml:lang*, *pos*, *absPos*, *dir*, *ocpStationRef*, *controllerRef*, *type* [main, distant, repeater, combined, shunting, other:anything], *function* [exit, home, intermediate, other:anything], *maskableRoute*, *maskableATC*, *ruleCode*, *virtual*, *sight*, *sigSystem*, *distNearestDangerPoint*, *trackDist* y *height*.

Height define la altura de la señal en metros (seis decimales), trackDist contiene la distancia entre la mitad de la vía y la mitad de la señal en metros, ruleCode define en una string el código de la regla que rige esta señal, virtual define con un tipo booleano si esta señal es únicamente virtual y por ello no es visible físicamente (solo visible en la cabina). Los atributos maskableRoute, maskableATC, , sight, sigSystem y distNearestDangerPoint no son recomendados por motivos de compatibilidad dado que estos serán eliminados en la versión 2.4.

Como se ha dicho, las señales pueden ser de varios tipos diferentes y todas ellas deben estar cubiertas por este elemento, por lo que existen varios subelementos aplicables a dichas señales, que se enumeran a en los siguientes puntos. Estos elementos son opcionales, pero de alta importancia.

En el caso práctico se han tomado los valores indicados por el Ministerio de Fomento[14] como normativos para la definición de señales ferroviarias. Tal vez los dos puntos más importantes en la definición de estos elementos sean los atributos *type, funcion* y *ruleCode*.

#### 7.1.7.1.1.5 - speed:

Define a la señal que lo contiene como señal de velocidad. A su vez, completa la definición de <speed> por medio de tres atributos que serán inherentes a la señal: *switchable* siendo verdadera o falsa según si se puede apagar y encender o si por el contrario es fija; *kind* [distant, repeater, combined, shunting][15]; trainRelation (definido igual que en el elemento 7.1.6.15.1 <speedChange>).

#### 7.1.7.1.1.5 - IS:speedChangeRef:

Este elemento contenido en el anterior, da una referencia para unir la señal de velocidad con el cambio de velocidad máxima al que obedece. Para eso, como siempre, se tiene un atributo *ref*.

#### 7.1.7.1.1.6 - etcs:

Este elemento define la señal como un panel de ETCS. Sus atributos son *switchable, srsVersion, level_0, level_1, level_2* y *level_3,* que ya se han visto en este texto.

[14] https://www.boe.es/buscar/act.php?id=BOE-A-2017-556

[15] Esta lista se define en el esquema de tipos de datos de infraestructura bajo el tipo <xs:simpleType name="tSignalType">. Para más información, acudir al punto Esquemas de Definición.

#### 7.1.7.1.1.7 - levelCrossing:

Elemento importante que precisa que la señal es referente a un paso a nivel. Este elemento da información sobre el tipo de señal de paso a nivel que es. Sus atributos son tres: *switchable, type* [bell, whistle, announcing, activating, supervision] para decir si la señal tiene aviso por campana, silbato, señales o barreras con anuncio del estado de seguridad técnica, panel indicativo del estado de las señales o barreras, o punto de supervisión inmediatamente antes del paso a nivel; y *ref* con el ID del <levelCrossing> al que pertenece.

#### 7.1.7.1.1.8 - trainRadio:

Señala que el elemento <signal> indica información referente a la radio del tren. Se completa con *switchable*, *trackConditionRef* con una referencia al ID del elemento <trackCondition> al que obedece.

#### 7.1.7.1.1.9 - catenary:

Indicación de que la señal en que se incluye es referente a un cambio en la catenaria. Se rellena igual que el anterior <trainRadio>

#### 7.1.7.1.1.10 - line:

Para paneles o señales de línea. Sus atributos son *switchable* y *ref* (ID de la línea respectiva)

#### 7.1.7.1.1.11 - milepost:

Significa que la señal es un poste de kilometraje. Tiene cinco atributos que lo definen: *switchable, shownValue* decimal que indica el valor exacto que se muestra en la señal o poste*, shownRemark* para cualquier nota que se encuentre en la señal*, mileageChangeRef* referencia al ID del elemento <mileageChange>*, lineRef* referencia al ID de una línea.

#### 7.1.7.1.1.12 - braking:

Indicador de que la señal es una señal de frenado. Se define por medio de los atributos *switchable* y *trackConditionRef*.

#### 7.1.7.1.1.13 - trainProtectionElementGroupRef:

Referencia que completa a la señal nombrando qué elemento <trainProtectionElementGroup> está conectado al aspecto de la señal. Especialmente válido para semáforos y señales luminosas. Solo tiene un atributo ref para indicar el ID del elemento <trainProtectionElementGroup> referido.

#### 7.1.7.1.1.14 - baliseGroupRef:

Igual que el elemento anterior, pero esta vez refiriendo a las balizas que estén conectadas a esta señal.

#### 7.1.7.1.2 - signalGroups:

##### 7.1.7.1.2.1 - signalGroup:

Elemento (y su previo contenedor sin atributos) que marca la relación entre señales que tienen alguna relación entre sí (la mayoría de las veces, relación de posición geográfica, o porque afecten o son afectadas por un mismo elemento). Su definición es simple, por medio de atributos y subelementos obligatorios <signalRef>, con uno de ellos por cada señal que se incluya en el grupo. Sus atributos son *id*, *code*, *name*, *description*, *xml:lang*, *type* [distant-main, station, other:anything] indicando si el grupo de señales es un grupo de señales distantes, de una estación, u otros.

###### 7.1.7.1.2.1.3 - signalRef:

Relación de señales (por el atributo ref que contiene el ID de la señal <signal>) que se incluyen en el grupo y su orden secuencial (atributo opcional *sequence*).

#### 7.1.7.2 - trainDetectionElements:

Elemento contenedor que forma parte de <ocsElements> y contiene cualquier elemento que se incluya en la vía con un uso determinado: el de la ayuda a la detección de trenes. Este elemento no tiene atributos pero sí dos, por un lado <trackCircuitBorder> y por otro <trainDetector>.

##### 7.1.7.2.1 - trackCircuitBorder:

Elemento que incluye los puntos donde el circuito de vía termina y empieza uno nuevo. Sus atributos son *id*, *code*, *name*, *description*, *xml:lang*, *pos*, *absPos*, *dir*, *ocpStationRef* y *controllerRef*.

##### 7.1.7.2.2 - trainDetector:

Este elemento es la definición de cualquier detector tipo contador de ejes, detector de eje caliente, etcétera. Sus atributos son *id*, *code*, *name*, *description*, *xml:lang*, *pos*, *absPos*, *dir*, *ocpStationRef*, *controllerRef*, *detectionObject* [wheel, axle, train, endOfTrain, obstacle, other:anything] (para detectores de ruedas, ejes, trenes, final de tren, obstáculos en la vía y otros) , *medium* [mechanical, hydraulic, pneumatic, inductive, optical, radio, other:anything] (mecánico, hidráulico, neumático, inductivo, óptico, radio y otros), *posInTrack* [center, leftRail, leftRailInside, leftRailOutside, rightRail , rightRailInside, rightRailOutside, outside, outsideLeft, outsideRight] (para su posición en las vías en el centro, en el carril izquierdo, dentro del carril izquierdo, fuera del carril izquierdo, en el carril derecho, dentro del carril derecho, fuera del carril derecho), *model, directionDetection* y *axleCounting* ambos booleanos que serán verdaderos si la detección es direccional, y si hay recuento de ejes.

#### 7.1.7.3 - balises:

Al igual que las señales, este contenedor de balizas contiene las balizas en sí mismas y sus posibles agrupaciones.

#### 7.1.7.3.1 - balise:

Elemento que modela las balizas de una vía. Sus atributos son *id*, *code*, *name*, *description*, *xml:lang*, *pos*, *absPos*, *dir*, *countryID* como referencia al ID del país al que pertenece la baliza, *groupID* para unir una baliza con su elemento baliseGroup , *linkingAccuracy, linkReactionAscending, linkReactionDescending, staticTelegram.*

#### 7.1.7.3.1.4 - baliseGroup:

Elemento que agrupa las balizas. El modo en que se utilizan es el mismo que en <signalGroup>, aunque aquí la referencia también la tiene la baliza en el atributo groupID. De este modo, se tendrán que definir en este elemento los atributos ya conocidos, siendo estos *id*, *code*, *name*, *description*, *xml:lang y type* que indica el uso funcional de este grupo con [infill, signal, technicalFixed, technicalSwitchable]. Para definir la referencia que se ha comentado, se tiene el subelemento baliseRef.

#### 7.1.7.3.1.4.3 - baliseRef:

Mismo funcionamiento que <signalRef>

#### 7.1.7.4 - trainProtectionElements:

Contenedor, sin atributos, de elementos que sirven para la protección del material rodante.

#### 7.1.7.4.1 - trainProtectionElement:

Definición de un elemento que sirve a la protección del tren. Pueden definirse tantos como se necesite. Sus atributos son: *id*, *code*, *name*, *description*, *xml:lang*, *pos, absPos, dir, medium* (misma definición que en 3.1.12 trainProtection), *trainProtectionSystem* (definición del sistema nacional de protección instalado en la vía), y *model.*

#### 7.1.7.4.2 - trainProtectionElementGroup:

Agrupación de elementos anteriores. Se forma con los atributos: *id*, *name*, *description*, *xml:lang* y con sus subelementos <trainProtectionElementRef> (tantos como elementos se agrupen)

#### 7.1.7.4.2.1 - trainProtectionElementRef:

Elemento que solo contiene la referencia de uno de los elementos que se unen según el grupo anterior por medio de su atributo *ref*.

#### 7.1.7.5 - stopPosts:

#### 7.1.7.5.1 - stopPost:

Elemento que define la posición de una señal de parada en la vía. Su correcta definición viene de la mano de sus atributos *id*, *code*, *name*, *description*, *xml:lang*, *pos, absPos, dir, ruleCode, trainRelation* (definido previamente, elemento que relaciona desde qué parte del tren la señal es válida), *platformEdgeRef* (referencia al ID del elemento <platformEdge> al que se asocia), *traiLength* (valor que se puede establecer si este poste es solo válido para trenes de longitud igual o inferior a la definida, en metros), *axleCount* (valor que se puede establecer si este poste

es solo válido para trenes que tengan un número igual o inferior de ejes al definida), *wagonCount* (la misma mecánica que el atributo anterior axleCount, pero para los vagones), *verbalConstraints* (valor de texto que se puede establecer si este poste es solo válido para cumplan una cierta restricción indicada por este valor), *virtual* y *ocpRef.*

En el ejemplo que acompaña a este texto no se ha incluido ningún elemento stop por no ser relevantes para la consecución del trabajo.

#### 7.1.7.5.1.5 - validForMovements:

Especificación de los trenes a los que afecta el STOP. Se definen por el atributo *kind* que es del tipo other:anything con la especificación del tren.

#### 7.1.7.5.1.6 - signalRef:

Este subelemento permite referenciar a varias señales que tengan el mismo comportamiento. Dicha referencia es por medio de *sequence* y *ref.* Estos atributos se han visto con anterioridad.

Se utilizará este elemento tantas veces como se necesite.

### 7.1.7.6 - derailers:

#### 7.1.7.6.1 - derailer:

Elemento que permite la definición de aparatos descarriladores. En el ejercicio práctico no se han incluido por falta de utilidad. Sus atributos son *id*, *code*, *name*, *description*, *xml:lang*, *pos, absPos, dir, ruleCode, model, kind* [tipo de descarrilador en un dato tipo string]*, derailerSide* [lado al que el tren descarrilaría en función del sentido ascendente de kilometraje].

### 7.1.7.7 - trainRadioChanges:

#### 7.1.7.7.1 - trainRadioChange:

Permite indicar los puntos de la vía a partir de los que hay un cambio del sistema de radio instalado en la infraestructura. Su utilidad es obvia, siempre dependiendo de la configuración de la radio en las vías. Su configuración reside en los siguientes atributos: *id*, *code*, *name*, *description*, *xml:lang*, *pos, absPos, dir, radioSystem, networkSelection, publicEmergency, broadcastCalls, textMessageService, directMode* y *publicNetworkRoaming*. Todos estos atributos se definen como se ha visto anteriormente (los primeros cinco aparecen en la mayoría de los elementos de <ocsElements>, y los demás en 3.1.11 - trainRadio>.

### 7.1.8 - infraAttrGroupRefs:

#### 7.1.8.1 - infraAttrGroupRef:

Último elemento que se utiliza para definir cada una de las vías en sí mismas. A pesar de la simpleza de este elemento (que solo tiene un atributo *ref* para indicar el ID del elemento <infraAttributes> que relaciona), es un elemento con mucha importancia y altísima utilidad dado que permite relacionar una vía con todo un conjunto de condiciones definidas previamente de forma global con solo una referencia. Esto significa que si uno de los elementos de la vía (por

ejemplo la radio) es común para varias vías (supongamos diez vías), no es necesario definir la radio diez veces con la misma información, sino que se puede definir una sola vez en el elemento <infraAttributes> y, más tarde, referir cada una de las diez vías con el elemento <infraAttrGroupRef>. Así visto, puede parecer irrelevante utilizar un modo u otro, pero si al ejemplo de la radio añadimos todas las posibles definiciones que el elemento <infraAttributes> permite (y que se pueden repetir fácilmente en muchas de las vías), son trece elementos que dejamos de repetir en cada una de las diez vías, ahorrando bastante tiempo con ello.

### 8 - controllers:

Elemento contenedor sin atributos para definir todos los sistemas de control y operación que no son tangibles.

### 8.1 - controller:

En él se pueden definir todos aquellos controladores de las instalaciones que estén a pie de vía. Habitualmente esto se hace por uno o pocos controladores. En el trabajo se ha optado por simplificar esto a un controlador por enclavamiento.

Los atributos de este elemento son los siguientes: *id*, *code*, *name*, *description* y *xml:lang.* También existen los siguientes atributos introducidos a partir de la versión 2.4 (por lo que quedan fuera de los límites de este estudio) *parentControllRef, model, type, technologyType, swVersion.* También se incluyen con la versión 2.4 el subelemento <ocpRef> que relaciona el controlador con el ocp referente.

### 9 - speedProfiles:

### 9.1 - speedProfile:

Como se indicó en el elemento 3.1.10.1 <speed> y en 7.1.6.15.1 <speedChange>, este elemento es muy importante tanto para el modelado de las velocidades máximas en las vías. Este elemento sirve para generar unos perfiles comunes de velocidad que establezcan características compartidas entre varias vías de la infraestructura.

Se modela por medio de los atributos *id*, *code*, *name*, *description, xml:lang, influence* para permitir establecer la superposición de perfiles de velocidad (útil en el caso de que un tramo tenga un perfil de velocidad, con pequeños tramos en que la velocidad sea menor para algunos trenes específicos), *maxAxleLoad* para señalar el peso máximo por eje de los trenes en este perfil*, maxMeterLoad* indicando el peso máximo por metro lineal de los trenes en este perfil*,* operatingPeriodRef relacionando el elemento <operatingPeriod> por su ID, startingTime, endTime, endDayOffset, trainProtectionSystems, verbalConstraint.

### 9.1.3 - tilting:

Correspondencia con la máxima capacidad de inclinación del tren que es necesaria para el tramo con estas características. Este elemento se modelará con tres atributos, como son: *maxTiltingAngle* que da el valor máximo que el tren necesita soportar en grados (valor de cero a noventa), *actuation* definiendo el modo de inclinación que tendrá el tren [active, pasive, rollCompensation o none] para un modo de funcionamiento activo, pasivo, de compensación de

giro, o ninguno; y el atributo *tiltingSpeed* que da información sobre la velocidad de cambio de inclinación del tren en grados por segundo.

#### 9.1.4 - braking:

Elemento muy importante para la seguridad del tren, que permite el modelado de las configuraciones necesarias del tren en el perfil de velocidad definido. Para ello usa tres atributos que son *brakeType* para indicar el tipo de freno con un tipo other:anything, *airBrakeApplicationPosition* dando la información sobre la posición del freno base en uno de los valores [G, P, R o N/A] representando respectivamente a mercancías, pasajeros, tren rápido, o información no disponible; y *minimumBrakePercentage* siendo este un entero entre 6 y 225 que representa el porcentaje de frenado según rendimiento de frenado en relación al peso del vehículo

#### 9.1.5 - path:

Este elemento indica que el perfil de velocidad pasa de un ocp a otro ocp vecino. Para esto, no tiene ningún atributo, completando su definición por medio del subelemento obligatorio que se expone a continuación.

#### 9.1.5.1 - ocpRef:

Llamada a cada uno de los ocp por los que pasa el perfil de velocidad. Es necesario rellenar uno por cada ocp que se incluya, y es muy conveniente rellenar qué orden siguen estos ocp. Sus atributos son *ref* y *sequence*, que ya son conocidos por el lector.

### 6.1.2 Elementos de Rolling Stock

Del mismo modo que con el esquema de infrastrucure, el de rollingStock tiene que ser definido antes de comenzar a definir sus elementos. Como ya es conocido, en este esquema tendrán cabida todos los elementos referentes a definiciones que se requieran para el material rodante. Como pasa en los demás esquemas, necesita una identificación y unos atributos a rellenar. Asimismo, también requiere unir este esquema con los demás esquemas que se hayan confeccionado. Por esto, el elemento rollingStock tiene los atributos son *id*, *code*, *name*, *description*, *xml:lang*, *version*, *xml:base*, *infrastructureRef* (referencia de la infraestructura por su ID) y *timetableRef*; y los siguientes subelementos que dan sentido al esquema:

#### 1 - vehicles:

El primer elemento de la parte que define el material rodante define los vagones que se van a poder encontrar en cualquier formación, uno a uno. Se trata de un elemento contenedor, en el que no se define ningún atributo.

#### 1.1 - vehicle:

Este elemento es muy importante para la correcta consecución del modelado de  tiene unos atributos a definir que son

- *id*, *code*, *name*, *description*, *xml:lang* (ya conocidos)

- *vehicleFamilyRef* para referenciar otros vehículos de modo que se pueden utilizar las características de uno para los demás
- *vehicleCategory* como catergorización del vagón en función de si es un vagón remolcado, para pasajeros con motor, con cabina de control, dedicado a tracción sin pasajeros, remolcado para materiales, y otros [coach, motorCoach, controlCabCoach, motorVehicles, freightWagon, other:anything]
- *axleSequence* definiendo el orden de los ejes en el vagón
- *numberDrivenAxles* para señalar cuántos ejes tienen entrega de potencia con un etero
- *numberNonDrivenAxles* con el número de ejes que son libres con un dato entero
- *trackGauge* para definir la distancia entre vías para la que está definido el vagón
- *trackGaugeAlternative* distancia alternativa en caso de que la anterior no se cumpl
- *adjustableWheelSet* booleano que , define si la anchura del eje es ajustable (si es verdadero, los atributos *trackGauge* y *trackGaugeAlternative* deben estar completos)
- *length* definiendo la longitud del vagón entre ambos acoples en metros
- *speed* como la velocidad máxima permitida para el vagón, en km/h
- *towingSpeed* con la velocidad máxima que el vagón permite si es remolcado en km/h
- *bruttoWeight* peso total del vagón en toneladas con carga
- *nettoWeight* peso total de la carga del vagón en toneladas
- *tareWeight* peso total del vagón vacío en toneladas
- *bruttoAdhesionWeight* fracción del peso total con carga que descansa sobre ejes tractores
- *tareAdhesionWeight* peso total en vacío que descansa sobre ejes tractores
- *axleLoad* carga total por eje en toneladas
- *resistanceFactor* factor de resistencia del vagón, en N/kN (para vehículos lentos)
- *onTrainHead* booleano que será verdadero si el vagón va en cabeza normalmente
- *onTrainTailOnly* booleano que define si el vagón debe ir únicamente en la cola del tren

Todo lo que a medidas físicas del vagón se refiere, se ha obtenido de documentación de la locomotora que se ha utilizado como ejemplo, así como de una estimación de peso de veintidós toneladas y media por eje, y cuatro ejes por vagón.

### 1.1.1 - classification:

Elemento contenedor sin atributos que da datos informativos o con carácter organizacional sobre el vagón.

#### 1.1.1.1 - manufacturer:

Muestra notando cuál es el fabricante del vagón y la información relativa a él que sea necesaria para su correcta identificación. Contiene tres atributos que son *vehicleManufacturerRef* como una referencia al código que el fabricante tenga en una lista de uso habitual (como la del UIC), *manufacturerType* como una descripción del tipo del vagón puesta por su fabricante a la hora de su fabricación, *serialNumber* para señalar el número de serie del vagón en su fabricación.

#### 1.1.1.2 - operator:

Definición de la información que el operador del vagón necesita habitualmente para su funcionamiento. Es posible tener más de un elemento <operator> para cada vagón, puesto que es posible que este elemento sea utilizado por más de un operador. El elemento quedará perfectamente definido por medio de cuatro atributos que son: *operatorClass* con el número o indetificador de la clase que el operador ha dado al vagón; *vehicleOperatorRef* referido al identificador del operador en las listas comunes de operadores; *startDate* y *endDate.*

#### 1. 1.1.2.1 - vehicleIdentification:

Subelemento usado para identificar el vagó según indica la UIC. Se forma por tres atributos que se detallan a continuación:

*uicIdentNumber* contiene el string de doce caracteres exactamente, que define el vehículo según los estándares de la UIC (o similar), *countryCode* informando del código del país registrado, para identificación, *vehicleKeeperMarking* marcando cuál es el dueño del vehículo por medio de una cadena string de uno a cinco caracteres

### 1.1.2 - engine:

Los motores son partes esenciales de un tren para que este funcione. Por ello, es vital tener información sobre las características del mismo. Esto implica que haya un elemento dentro de la definición del tren específico para modelar los motores y su comportamiento. Solamente contiene dos atributos, pero muchos subelementos.

Sus dos atributos son *haulageLoad,* que indica el peso máximo de carga permisible en toneladas que puede ser remolcado; y *axleDriveType* que contiene la definición de la manera en que el par es entregado al eje. Los posibles valores de este último son [cardanShaft, tubularAxle, noseSuspensionDrive, helicalSpringGear , rubberRingResilientDrive, buchliDrive, inclinedRodDrive, sideRodDrive, chain u other:anything] según si la entrega hace por medio de eje con junta cardan, eje tubular, engranaje directo, engranaje helicoidal, junta de anillo de resiliencia, transmisión de Buchli, barra inclinada, barra lateral, cadena u otros tipos.

#### 1.1.2.1 - monitoring:

Subelemento de <engine> que contiene los datos relativos a los sistemas de control y protección instalados en el tren. Es un elemento contenedor que se define por medio de sus elementos descendientes.

#### 1.1.2.1.1 - etcs:

Este elemento representa las capacidades del vehículo con respecto a la normativa europea. Aunque su funcionamiento y definición no iguales, la definición de sus atributos es la misma que en el elemento 7.1.7.1.1.6 - <etcs>.

#### 1.1.2.1.1.1 - specificTransmissionModule:

Contiene datos acerca de cualquier interfaz de sistema ETCS instalada en el vagón para control y operación. Se modela con los atributos *id*, *code*, *name*, *description*, *xml:lang, softwareVersion* conteniendo un string que contiene la versión del software instalado, *onBoardUnitID* contiene el ID único de una unidad a bordo particular que será usado para su comunicación con el centro de control, y *nationalSystemRef* como referencia al ID del elemento <nationalSystem> que va adjunto a este módulo.

#### 1.1.2.1.2 - nationalSystem:

Elemento que define los detalles para conocer cada uno de los sistemas de control y protección instalado. En caso de que estos elementos sean usados para ETCS, los respectivos atributos deberán estar rellenos en el elemento <etcs>. Sus elementos son *id*, *code*, *name*, *description*, *xml:lang, trainProtectionMedium, trainProtectionMonitoring, softwareVersion, onBoardUnitID* (todos ellos ya conocidos) y *type* que da información sobre el tipo de sistema de protección usado.

#### 1.1.2.1.3 - trainRadio:

Definición de los sistemas de radio que se encuentran en el tren. En este elemento, se utilizan los siguientes atributos para su definición: *id*, *code*, *name*, *description*, *xml:lang, softwareVersion, onBoardUnitID* y *modulation* [analogue, digital, other:anything] si la transmisión de radio se hace por modulación analógica, digital, o por otros medios.

#### 1.1.2.1.3.1 - radioSystem:

Este elemento es especial dado que es el único cuya definición no se hace mediante atributos, sino que se rellena con los datos intrínsecos. Para aclarar esto, se añade un ejemplo que define el sistema de radio como GSM-R:

<radioSystem>GSM-R</radioSystem>.

#### 1.1.2.1.4 - otherEquipment:

Elemento que se añade para definir todo aquel elemento de tráfico ferroviario que no afecta a la seguridad y no tiene cabida en <nationalSystem> ni en <trainRadio>. Se define por los atributos *id*, *code*, *name*, *description*, *xml:lang, softwareVersion* y *onBoardUnitID.*

### 1.1.2.2 - pantograph:

El pantógrafo es un elemento con gran significación dentro de la definición de los motores de un tren, puesto que sin pantógrafo no hay transmisión de corriente desde la catenaria. Este elemento contiene los atributos:

- *orderNumber* para dar el orden del pantógrafo en el vagón dentro del total de pantógrafos. El valor del que se encuentre más adelante debe ser 1, mientras que el que esté más atrasado debe ser en número *n.*
- *designType* incluyendo un nombre descriptivo del tipo de pantógrafo usado.

- *positionOnSection* dando la posición del pantógrafo en la longitud del vagón. Sus valores posibles son [front, frontSecond, middle, rearSecond, rear, other:anything] según si el pantógrafo se encuentra al frente, en segunda posición, en el medio, en penúltima posición, en la parte trasera u otra posición.
- *fittedSection* para incorporar el caso de vagones articulados. En este atributo se dará el número de la sección en que se encuentra el pantógrafo.
- *controlType* con una lista de valores [cable, spring, air, other:anyhting] definiendo si la actuación del pantógrafo (subida o activación) se efectúa por cable, muelle, neumática, u otro tipo
- *headWidth* conlleva la longitud en metros de la cabeza del pantógrafo (con seis decimales)
- *maxCurrentDriving* da información sobre la máxima corriente en amperios que es capaz de soportar el pantógrafo cuando el tren está en movimiento.
- maxCurrentStandstill contiene la máxima corriente en amperios que es capaz de soportar el pantógrafo cuando el tren está en parado.

#### 1.1.2.2.1 - dedicatedSupplySystem:

Elemento que se utiliza para precisar las capacidades del sistema de suministro para las que está diseñado el pantógrafo. Se define por los atributos *voltage* con el el voltaje nominal en voltios, y *frequency* con la frecuencia nominal en hercios. En caso de que la corriente sea continua, se establecerá el valor de la frecuencia a cero herzios.

#### 1.1.2.3 - propulsion:

Elemento que contiene toda la información necesaria para modelar el sistema de propulsión relativo a un sistema de suministro en concreto (esto significa que si hay tres diferentes sistemas de suministro definidos, se necesitará modelar tres elementos <propulsion>. Contiene bastantes elementos descendientes, y también atributos, que son:

- *id*, *code*, *name*, *description*, *xml:lang,voltage, frequency*
- *power:* es la potencia instalada en vatios
- *powerType:* siendo este el tipo de propulsión que se usa, eligiendo de la lista de valores [electric, diesel, steam, other:anything] siendo estos valores para potencia eléctrica, diésel, vapor u otros
- *transimission:* según si la transmisión de la potencia es eléctrica, hidráulica, mecánica u otro tipo [electric, hydraulic, mechnical, other:anything]
- *controlType:* para definir el control de la salida de potencia con un valor de [unknown, camshaftControl, contactorControl, rectifier, thyristorControl, other:anything] dependiendo de si se controla por: medio desconocido, árbol de levas, contacto, rectificador, tiristor, u otros
- *maxTractEffort*: máximo esfuerzo tractor que se puede alcanzar en rueda
- *rotationMassFactor*: factor de multiplicación de la resistencia a la rodadura a causa de las masas rotatorias.
- *additionalRotationMass*: aumento de resistencia a la rodadura en toneladas

- *remoteControl*: booleano que denomina si el vehículo puede ser controlado de manera remota
- *numberNotches*: número de dientes que tiene la rueda de control (para maquinaria antigua)
- *wheelDiameter*: valor nominal del diámetro de las ruedas en metros
- *maxBrakeEffort*: máximo valor de frenado en ruedas, dado en Newton (usado en conjunto con el elemento <brakeEffort>)
- *maxBrakePower*: esfuerzo máximo de frenado por el freno regenerativo (usado en conjunto <brakeCurrent>)
- *totalTractEfficiency*: eficiencia media del sistema completo de propulsión en tracción (usado en conjunto con <tractiveVehicleEfficiency>)
- *totalBrakeEfficiency*: eficiencia media del sistema completo de propulsión en frenado (usado en conjunto con <brakeVehicleEfficiency>)
- *tractionOffUnderVoltageThreshold*: valor límite en voltios para el que el sistema de propulsión debe apagarse, si el suministro en pantógrafo baja de ese valor
- *zeroSpeedCurrentLimitation*: corriente límite en amperios para el sistema de propulsión en parado
- *maxRegenerativeVoltage*: valor máximo de voltaje en voltios en el pantógrafo en frenada regenerativa
- *speedRange*: valor de [slow, fast, dontcare] según si los datos del sistema de propulsión es válido para velocidades lentas, rápidas, o cualquiera.
- *forwardSpeed*: velocidad máxima permisible en dirección normal en km/h
- *reverseSpeed*: velocidad máxima permisible en dirección marcha atrás en km/h
- *activationStandstill*: booleano que informa si el sistema de propulsión puede ser activado en parado

#### 1.1.2.3.1 - link:

Este elemento da información acerca de cómo se unen los circuitos del sistema de tracción. Se define por medio de dos atributos *nominalVoltage* refiriendo al voltaje que usa el circuito en voltios y *nominalCurrent* con el valor nominal de corriente en amperios. Ambos atributos son obligatorios para definir este elemento.

#### 1.1.2.3.2 - tractiveEffort:

Elemento que contiene el esfuerzo disponible en rueda en función de la velocidad del tren. Esto se rellena por medio de una tabla, que es incluye como subelemento. Los valores que completan dicha tabla, se han obtenido, directamente de documentación de una locomotora genérica que ha sido utilizada en la red española. Por motivos de sencillez de definición del caso práctico, se ha añadido solamente una pareja de conjuntos de valores.

#### 1.1.2.3.2.1 - valueTable:

Tabla que incluye los valores de tractiveEffort. Su funcionamiento se describe en el apartado 4.3.3 Tipos de Datos. Sus subelementos son 1.1.2.3.2.1.1 - columnHeader, 1.1.2.3.2.1.2 - valueLine, y 1.1.2.3.2.1.2.1 - values

#### 1.1.2.3.3 - brakeEffort:

Contiene información del esfuerzo de freno mecánico en rueda en función de la velocidad. También utiliza el dato de tablas <valueTable>.

##### 1.1.2.3.3.1 - valueTable:

Elemento que incluye los valores de brakeEffort en forma de tabla.

#### 1.1.2.3.4 - auxiliarySupply:

Contiene datos del suministro de potencia que se utiliza en el sistema auxiliar, en términos de cálculos de consumo de potencia. Se completa con los atributos *power* señalando la potencia constante consumidas por los auxiliares en vatios*, powerPhi* representando el ángulo de cambio de fase potencia aparente en los auxiliares dada en el atributo anterior*, resistance* con la resistencia en ohmios*, powerBraking* con la potencia constante en vatios que los auxiliares consumen adicionalmente en el modo de frenado*, powerPhiBraking* informando sobre el ángulo de forma de la potencia aparente en la potencia dada en powerBraking*, resistanceBraking* como la resistencia en ohmios de los sistemas auxiliares presentes solo en el modo de frenado.

#### 1.1.2.3.5 - transformer

Elemento que guarda la información referente al transformador del sistema de propulsión, que se modela por los atributos *count* conteniendo el número de transformadores disponibles, *assemblyName* incluyendo el nombre descriptivo de los transformadores, *meanEfficiency* con la eficiencia media del sistema, *ferrumResistance* para la resistencia en ohmios del circuito equivalente del transformador, *additionalResistance* con la resistencia adicional de dicho circuito equivalente debida a corrientes parásitas, y *mainInductance* guardando la inductancia media del circuito equivalente en Henrios.

Contiene, además, dos subelementos:

##### 1.1.2.3.5.1 - efficiency

Elemento que contendrá una tabla de valores (como se vio anteriormente, incluyendo el subelemento <valueTable>) referida a la eficiencia en función de su velocidad o de su porcentaje de carga (contiene el elemento descendiente <valueTable>).

##### 1.1.2.3.5.2 - winding

Información relativa al devanado del transformador. Esta información se proporciona por meido de los atributos *assemblyName, nominalVoltage, nominalCurrent, transformationRatio* con el coeficiente entre devanado primario y secundario, *primaryResistance* atesorando la resistencia total del devanado primario en ohmios, *secondaryResistance* almacena la resistencia total del devanado secundario en ohmios (transformada al valor del devanado primario), y *primaryLeakageInductance secondaryLeakageInductance* para la inductancia del devanado primario y secundario respectivamente en Henrios (la del secundario, transformada al del primario).

#### 1.1.2.3.6 - brakeCurrent:

Elemento que guarda en una tabla <valueTable> la corriente de frenado producida por el sistema de frenado regenerativo en función de la velocidad (contiene el elemento descendiente <valueTable>). No se consideran corrientes alimentadas a las resistencias reostáticas del vehículo.

#### 1.1.2.3.7 - brakeCurrentLimitation

Elemento que guarda en una tabla <valueTable> la limitación de corriente máxima en función del voltaje en el pantógrafo en modo de frenado (contiene el elemento descendiente <valueTable>).

#### 1.1.2.3.8 - brakeVehicleEfficiency

Elemento que guarda en una tabla <valueTable> la eficiencia del sistema en función de la velocidad en modo de frenado (contiene el elemento descendiente <valueTable>).

#### 1.1.2.3.9 - diesel

Contiene los datos referentes a los motores diésel utilizados para tracción. Se define por los atributos *count* incluyendo en un entero positivo el número de motores **del mismo tipo** usados en un vagón (aún no se contempla en RailML® la posibilidad de usar dos tipos de motores diésel en el mismo vagón), *assemblyName* tiene el nombre descriptivo del motor, *nominalPower* guarda un entero con los vatios de potencia nominal que el motor genera, *availableAuxiliaryPower* con la potencia que se deja para sistemas auxiliares (de un motor), *fuelCapacity* para la capacidad de combustible de todos los depósitos del vagón, *meanEfficiency* teniendo la eficiencia media del motor en condiciones normales.

#### 1.1.2.3.10 - fourQuadrantChopper

Elemento con la información referente a los circuitos de frenos para las configuraciones de cuatro cuadrantes. Los atributos utilizados en este caso son *count, assemblyName, meanPhi, meanPhiRegeneration y meanEfficiency.* Contiene los subelementos <phi> y <efficiency>, siendo estos rellenos por elementos tabla.

#### 1.1.2.3.11 - gear

Elemento que contiene los datos para la transmisión de potencia desde el motor a las ruedas. Tiene un subelemento <efficiency> que se puebla con un elemento tabla (<valueTable>) y, además, contiene los siguientes atributos:

*count, assemblyName, gearRatio* con la relación de multiplicación con un valor entre cero y uno, *meanEfficiency, designType, manufacturerName* almacenando el nombre del fabricante, *nominalPower* con la potencia en vatios que el grupo puede transmitir y *torqueConversion* seleccionando uno de los valores [converter, coupling, hydrodynamicTransmission, hydrostaticTransmission, hydromechanicallTransmision, hydraulicTransmission, other:anything] señalando si hay un convertidor de par, un embrague, transmisión hidrodinámica, hidrostática, hidromecánica, hidráulica, u otros tipos.

#### 1.1.2.3.12 - rackTraction

Se rellena en caso de que haya ruedas dentadas para ejercer el esfuerzo tractor. Solo contiene tres atributos, que son number para la cantidad de ruedas dentadas por vagón, rackSystem on un valor entre [Riggenbach, Riggenbach-Klose, Abt2Bars, Abt3Bars, Locher, Strub, Wetli, Marsh, Roll, other:anything] y *resilentCogWheel* infromando con un booleano si la rueda está suspendida o si por el contrario es fija.

#### 1.1.2.3.13 - tractionInverter

Elemento que contiene la información relativa al inversor de corriente unido a los motores de tracción. Se rellena por tres atributos ya conocidos que son *number, assemblyName* y *meanEfficiency.* También, para mayor información, se pueden rellenar dos subelementos <efficiency> y <pulsePattern> que mantendrán la información relativa a la eficiencia y al patrón de pulsos del inversor respectivamente, ambos en subelementos tabla.

#### 1.1.2.3.14 - tractionMotor

Destinado a dar información relativa a los motores destinados a tracción. En muchas ocasiones, puede ser detallado como el circuito equivalente con sus resistencias y reluctancias. Para su correcta definicón, el elemento consta de varios atributos, como son *count, assemblyName, meanEfficiency, nominalVoltage, nominalCurrent, nominalFrequency, nominalRevolutions* con las revoluciones nominales para las que el motor está diseñado dadas en herzios*, nominalFlux* con la cifra del flujo nominal medido en voltios por segundo*, nominalPhi, numberPolePairs* da el número de **pares** de polos que el motor contiene*, statorResistance* incluyendo la resistencia en ohmios del circuito estator*, rotorResistance* dando el valor de la resistencia en ohmios de *, ferrumResistance, additionalResistance, mainInductance, statorLeakageInductance, rotorLeakageInductance.*

Este elemento se completa con dos subelementos <efficiency> y <mechanicalLosses>, ambos con un subelemento de tipo tabla, y representando datos de eficiencia y de pérdidas mecánicas, respectivamente, en función de la velocidad

#### 1.1.2.3.15 - tractiveCurrent

Descripción guardada en una tabla <valueTable> de la corriente neta del sistema en función de la velocidad en modo de de tracción (contiene el elemento descendiente <valueTable>).

#### 1.1.2.3.16 - tractiveCurrentLimitation

Elemento que guarda en una tabla <valueTable> la limitación de corriente máxima en función del voltaje neto en el pantógrafo en tracción (contiene el elemento descendiente <valueTable>).

#### 1.1.2.3.17 - tractiveVehicleEfficiency

Elemento que guarda en una tabla <valueTable> la descripción de la eficiencia en función de la velocidad en tracción (contiene el elemento descendiente <valueTable>).

### 1.1.3 - wagon

Contenedor con atributos, que da toda la información necesaria relativa a la carga del vehículo. Los atributos por los que este elemento se rige son los siguientes *rotationMassFactor, additionalRotationMass, auxiliaryPowerConsumption, headShape* con uno de los valores siguientes [angular, rounded, streamlined, other:anything] según la forma de la cabeza del vagón sea angular, redonda, lineal,u otro tipo, *headSurface* con una valoración del área la sección de cabeza*, bearingType* [rollerBearing, ballBearing, plainBearing, other:anything] según si las sujeciones de los ejes sean por rodamientos de cilindros, rodamientos de bolas, cojinetes, u otros.

#### 1.1.3.1 - auxiliarySupplySystem

Utilizado para detallar los sistemas de suministro que pueden usar los auxiliares del vagón. Se define por medio de los atributos *voltage* y *frequency.*

#### 1.1.3.2 - couplers:

Elemento que contiene los datos referentes a los acoples entre vagones. Esta información se guarda dentro de los subelementos, que son detallados a continuación.

##### 1.1.3.2.1 - electricalCoupler:

Elemento a rellenar si el acople tiene aparatos de tipo eléctrico, como son los cables usados para el control del acople. Se define este elemento por medio de cuatro atributos:

- *desingType* es la descripción del diseño y funcionalidad del material del acople eléctrico. Tiene varios valores posibles que se detallan a continuación:

  - pushTrainControl36: control común con treinta y seis líneas
  - pushTrainControl34: control común con treinta y cuatro líneas
  - emergencyBrakeDeactivation: cableado para la desactivación del freno de emergencia
  - UIC-IS-cable21: cable con veintiuna líneas según el estándar UIC-IS
  - UIC556: cable estándar de acuerdo con UIC 556
  - UIC558-18: cable estándar de dieciocho líneas de acuerdo con UIC 558
  - UIC568-13: cable estándar de trece líneas, deacuerdo con UIC 558
  - wireTrainBus: cable tipo WTB (wire train bus)
  - multifunctionVehicleBus: cable tipo MVB (multifunction vehicle bus)
  - trainCommunicationNetwork: cable TCN (train communication network) que equivale al conjunto de los dos anteriores
  - vehicleSpecificConnection: conexión eléctrica específica para el vagón en que está montado
  - fibreOptics: conexión de fibra óptica en general
  - unknown: tipo desconocido
    other:anything: otro tipo

- *positionOnCarEnd* representa la posición del acople (si está en en el final del vagón, en el principio, o en ambos) [rear, front, both]
- *positionOnMechanicalCoupler*: este elemento define la posición del aparato eléctrico en relación al acople mecánico. Sus valores posibles son [none, aside, ontop, below] según

si el aparato eléctrico está sin posición cerca del aparato mecánico, a un lado de él, encima, o debajo de él.

- *numberContacts*: representa el número de contactos o líneas que el acople eléctrico provee.

#### 1.1.3.2.2 - mechanicalCoupler:

Este es el elemento que se utiliza para guardar las características de los aparatos mecánicos de acople. Para esto, define los siguientes atributos: *positionOnCarEnd* (igual que en el elemento anterior), *couplingHeight* representando la altura sobre el suelo a la que se sitúa el acople en metros, *pullingForce* con el valor en newton que el acople puede aguantar cuando el esfuerzo es de tracción, *pushingForce* albergando el valor en newton de la fuerza que el acople aguanta si el esfuerzo es de compresión y *designType* señalando el tipo de diseño del acople, este atributo se rellena con uno de los valores de la siguiente lista:

- *AAR_KnuckleCoupler*: acople estándar en los ferrocarriles norteamericanos
- *AK69e*: acople según diseño de Knorr compatible con acoples SA-3 y willison
- *AlbertCoupler*: acople utilizado en tranvías
- *APTA_tightlock*: acople automático de la asociación americana de transporte público
- *BSI_compact*: acople de tranvías hecho por BSI
- *BSI_shunting*: acople automático para maniobras, por BSI
- *buckeyeKnuckleCoupler*: acople de nudillos automático usado en trenes británicos de pasajeros
- *BuddPinCup*: acople desarrollado por la compañía Budd Company
- *buffersChainScrewCoupler*: acople estándar usado en la mayoría de trenes europeos
- *C-AKv*: acople compacto willison
- *centralBuffer1Chain*: acople amortiguado con una cadena por debajo, como se usa, por ejemplo, en Noruega
- *centralBuffer2Chains*: acople amortiguado central con una cadena a cada lado
- *FK-3-2.5*: acople automático de tranvías por Georg Fisher AG
- *FK-5.5-4*: acople automático de tranvías por Georg Fisher AG
- *FK-9-6*: acople automático de vagones de metro por Georg Fisher AG
- *FK-15-10*: acople automático de trenes principales por Georg Fisher AG
- *FK-15-12*: acople automático de trenes principales por Georg Fisher AG
- *GF_Coupler*: acople semiautomático por Georg Fisher AG
- *GFN*: acople semiautomático por Georg Fisher AG
- *GFT*: acople semiautomático por Georg Fisher AG
- *GFV*: acople semiautomático por Georg Fisher AG
- *Intermat*: acople desarrollado para redes de Europa del este
- *linkPin_flat*: acople de tranvías con unión plana de barra y pines
- *linkPin_round*: acople de tranvías con unión redonda de barra y pines
- *NorwegianMeatChopper*: acople amortiguado central con un gancho mecánico
- *SA3_WillisonCoupler*: acople de los trenes de la antigua Unión Soviética
- *Scharfenberg*: acople Scharfenberg
- *Scharfenberg10*: acople Scharfenberg10
- *Scharfenberg35*: acople Scharfenberg35
- *Scharfenberg55*: acople Scharfenberg55
- *Scharfenberg140*: acople Scharfenberg140
- *Scharfenberg330*: acople Scharfenberg330

- *Scharfenberg430*: acople Scharfenberg430
- *SchwabCoupler*: acople automático de schwab Verkehrstechnik AG
- *Wedgelock*: acople automático de la red de metro de Londres
- *unknown*: acople desconocido
- *other:anything*: cualquier otro tipo de acople

#### 1.1.3.2.3 - pneumaticCoupler:

Datos relativos a los aparatos neumáticos del acople. También cubre las tuberías conectadas al vehículo. Se define por medio de tres atributos, que son *positionOnCarEnd, integratedOnMechanicalCoupler* definiendo con un booleano si el aparato neumático está integrado en el sistema mecánico del acople, y *coupleFunction* eligiendo un valor de entre [mainAirPipe, mainAirReservoirPipe, additionalBrakeControlPipe, vacuumAirPipe, steamHeatingPipe, unknown, other:anything] para designar el elemento como la tubería principal de control, la tubería principal para alimentar la reserva de aire, tubería para el control de un freno adicional, tubería para uso de freno de vacío, tubería utilizada para el calentamiento de vapor, uso desconocido, u otros tipos.

### 1.1.3.3 - trainClearanceGauge:

Indicación de la distancia libre alrededor del tren para la que está hecho el vagón. Se rellena con un solo atributo *code* que contiene el código de dicha distancia libre según normativa europea.

### 1.1.3.4 - driversCab:

Contiene la información del puesto del maquinista en el vagón si la hay. Si el elemento se omite en la descripción, se entenderá que no hay cabina de maquinista en este vagón. Sin embargo, si se incluye, debe tener completos sus tres atributos *orderNumber, position* y *acousticSignaller* [bell, alarmBell, horn, airChime, whistle, other:anything] para definir su señal acústica como campana, elemento similar a una campana pero con cierta frecuencia (campanas con actuadores electromagnéticos), claxon por aire presurizado, claxon de más de un tono, silbato u otros tipos.

### 1.1.3.5 - goods:

Elemento que denota que el vagón contiene mercancías, bien sea en el vagón completo o en una parte adaptada. Su definición se divide entre un subelemento <service> y cinco atributos , que son *load* conteniendo la carga máxima permitida en toneladas*, loadArea* con el área libre por el que se puede cargar la mercancía en metros cuadrados*, volume* conteniendo el volumen de carga disponible*, freightCarType* eligiendo un valor de la lista [open, covered, platform, refrigerated, tank, self-discharging, special, other] para denotar si el vagón (o el espacio de carga) es abierto, cerrado, una plataforma sin paredes, cerrado y refrigerado, ajustado para gases o líquidos, con auto descarga, especial u otros tipo;  y *freightCarUICType* con una clasificación del vagón, en concordancia con UIC.

En el ejercicio práctico se ha entendido que todos los vagones son vagones de pasajeros y no se ha modelado este tipo de vagones.

#### 1.1.3.5.1 - service:

Este elemento se utiliza para definir los servicios adicionales que son provistos en el vagón. Se modela por medio de los siguientes atributos *count, name, description, value* y *type* [mobileCatering, WLAN, wheelchairLift, toiletClosed, toiletOpen, toiletHc, Snack, SelfService, PIS, HVAC, APC, SecurityCamera, other] para definir el servicio como cátering, wifi, acceso para silla de ruedas con elevador, aseo con alcantarillado cerrado, aseo con alcantarillado abierto, aseo para discapacitados, servicio de quiosco con bebidas y tentempiés, máquinas de vending, sistemas de información a pasajeros, calefacción ventilación y aire acondicionado, servicio de recuento automático de pasajeros, cámara de seguridad y otros.

### 1.1.3.6 - passenger:

Denominación de un vagón como vagón de pasajeros. La mayoría de la configuración de este elemento se hace por medio de sus subelementos, aun así, existen cinco atributos que se pueden dar a este elemento, que son *deck* definiendo si es un vagón de uno, dos o tres pisos [single, double, triple]*, drivingCab* representando en un dato booleano si el vagón tiene cabina de maquinista*, tilting* también booleano que indica si el vagón tiene sistema de inclinación*, airTightness* incluyendo un booleano que define si el vagón tiene sistema de resistencia ante ondas de presión del exterior*, emergencyBrakeDeactivation* último booleano que determina si el freno de emergencia puede ser desactivado.

Estos elementos deben definirse con cierta precaución dado que algunos de estos datos se pueden

#### 1.1.3.6.1 - gangway:

Establece si hay pasarela entre los vagones o no. En caso de que este elemento se omita, se entenderá que dichas pasarelas no existen. Se puede definir si las pasarelas están en la parte delantera del vagón, en la trasera, en ambas o en niguna. Esto se hace por medio de los dos primeros atributos *frontEnd, rearEnd* que serán verdaderos si la pasarela está en la parte delantera y en la trasera respectivamente. Asimismo, también tiene los siguientes atributos: *position* con los valores [standardUIC, upperDeck, lowerDeck other:anything] para posición acorde con el estándar UIC, posición en la planta superior, en la planta inferior,u otros; *designType* según los valores [standardUIC561, special, nonPublic, sigI, sigII, unknown, other:anything] apra indicar el tipo de pasarela (estándar de UCI, especial, usada solo por personal del tren, tipo I definido por SIG, tipo II definido opr SIG, desconocido, u otros); *floorLevel* señalando la altura en metros del carrusel sobre el suelo, *gangwayHeight, gangwayWidth* que dan, en metros, la altura y anchura respectivamente de la pasarela para pasajeros.

#### 1.1.3.6.2 - places :

Definición de la capacidad del vagón en términos de número de plazas, y la categoría de las mismas. Si existen plazas de más de un tipo, se deben definir varios de estos elementos, cada uno de ellos para cada tipo de plaza. Contiene los siguientes atributos: *tapTsiType9039Code* con el código de las plazas según los códigos de los documentos técnicos ERA*, count* almacenando el número de plazas o habitáculos (según la categoría)*, description* con una descripción de las

plazas y *category* guardando el tipo de plaza, según el valor de la lista [class1, class2, class3, standing, standingArea, wheelchair, bicycle, couchette, bed, sleepingCompartment, chair, bistro, restaurant, foldingSeat, impairedToilet, toilet, business, businessCompartment, family, familyCompartment, toddler, toddlerCompartment, other:anything] según si la plaza es de clase 1, de clase 2, clase 3, plazas de pie, área disponible para estar de pie (en metros cuadrados), plazas para sillas de ruedas, plazas para bicicletas, salas con literas, número de camas, habitaciones dormitorio, sillas para dormir, plazas en área de bistró, plazas en restaurante, plazas plegables, plazas en aseos para discapacitados, plazas en servicios, plazas en zona business, número de compartimentos business, plazas en compartimentos familiares, número de compartimentos familiares, plazas para niños, compartimentos de plazas para niños, y otros.

En la siguiente figura, se observa cuál es la forma de presentar los datos relativos a las plazas del vagón que se esté estudiando en cada momento.

| coche_turista | | |
|---|---|---|
| **longitud [m]** | 20,0 | |
| **peso bruto [t]** | 90,0 | |
| **peso neto [t]** | 28,8 | |
| **número de ejes** | 4 | |
| **información de fabricante** | fabricante desconocido num.1 | BOETSII |
| **uso de pasajeros** | **categoría** | **recuento** |
| | wheelchair | 6 |
| | class2 | 35 |
| | other | 2 |

Fig. 6: Ejemplo de representación de plazas y otras características de un vagón definido. Extraído del caso práctico que se ha generado durante este trabajo.

#### 1.1.3.6.3 - service:

Elemento que señala los servicios que se ofrecen en el vagón, con la misma definición que 1.1.3.5.1 - service.

#### 1.1.3.6.4 - doors:

Información relativa a los accesos al vagón. Es válido para uno de los lados del vagón, es decir, que para ambos lados habrá que definir dos elementos de este tipo. Sus atributos son *number* con el número de puertas*, entranceHeight* para la altura de las puertas en metros*, entranceLength* distancia entre puertas en metros*, entranceWidth* anchura de la puerta en metros *, footStepHeight* altura del primer paso que se debe dar para subir al vagón y *lockingSpeed* dando la velocidad de límite en km/h a la que se pueden cerrar las puertas.

#### 1.1.3.6.4.1 - passengerFlowSpeed:

Elemento que define, para fines de estadística y planificación, la capacidad de flujo de pasajeros por los accesos a los vagones. Los atributos son *FlowSpeed* y *platformHeight*, definiendo la velocidad de flujo en número de pasajeros por segundo, y la altura del andén en milímetros para una velocidad de entrada o salida determinada.

Esto significa que se pueden definir varios elementos passengerFlowSpeed para sendas alturas.

#### 1.1.3.6.5 - tilting:

Elemento que define la capacidad de inclinación del vagón. Su funcionamiento es el mismo que el definido en el punto 9.1.3 - tilting>

### 1.1.3.7 - rackTraction

Elemento que debe rellenarse en caso de que el vagón esté equipado con sistema de tracción que transmiten el par al suelo por medio de ruedas dentadas.

Su definición se vio ya en el elemento 1.1.2.3.12 - rackTraction>.

### 1.1.4 - vehicleBrakes

Elemento contenedor de datos relativos a la capacidad de frenado del vehículo. Contiene los subelementos eddyCurrentBrake, mechanicalBrakeEffort y vehicleBrake.

#### 1.1.4.1 - vehicleBrake

Aquí se define cada uno de los sistemas de freno. Dicha definición se hace a través de los atributos *brakeType* de la lista [none, compressedAir, vacuum, handBrake, parkingBrake, cableBrake, other:anything] para los valores respectivos ningún freno, freno de aire comprimido, de vacío, aplicado manualmente, de aparcamiento, de cable y otros; *airBrakeApplicationPosition* para el valor de posición de freno de aire comprimido [N/A, G, P, R] según no aplicable, mercancías, personas, o tren rápido; *regularBrakeMass* con la masa en toneladas para frenos normales no automáticos, *emergencyBrakeMass* para la masa en toneladas para frenos de emergencia no automáticos, *maxDeceleration* con el valor de deceleración máxima en metros por segundo cuadrado, *meanDeceleration* guardando el valor medio de deceleración, *loadSwitch* con información opcional de si el tren está en posición "lleno" o "vacío" en los sistemas que tienen este selector [full, empty], *autoBrakePercentage* dando el porcentaje de freno regulado automáticamente con el peso real del vehículo, y *maxAutoBrakeMass* con la máxima masa para el sistema de frenos automático.

#### 1.1.4.1.1 - auxiliaryBrakes

Definición de los valores de los posibles sistemas auxiliares de frenado. Se rellena con los atributos a continuación: *brakeUse* [unknown, normal, emergency, both] señalando si el tipo de freno es desconocido, normal, de emergencia o de ambos tipos, *H* booleano que será cierto si el freno funciona junto con un freno hidrodinámico, *E* será un booleano que marca si el freno funciona junto con un freno eléctrico, *Mg* booleano que será verdadero si el freno funciona

junto con un freno magnético, *Wb* con otro booleano que será verdadero si el freno funciona junto con un freno de corrientes parásitas y *ep* booleano verdadero si el freno funciona con una activación eléctrica del freno neumático para reducir el tiempo de respuesta.

#### 1.1.4.2 - mechanicalBrakeEffort

Elemento que contendrá la tabla de valores (tipo de formato ya visto anteriormente) que describe el esfuerzo en rueda del freno mecánico en función de la velocidad.

##### 1.1.4.2.1 - valueTable

Dato tipo tabla que define los valores del elemento anterior (este elemento ya se definió anteriormente en 1.1.2.3.2.1 - valueTable).

#### 1.1.4.3 - eddyCurrentBrake

Definición de los frenos por efecto corriente de Foucault. Su definición se basa en tres puntos, como son la potencia máxima en vatios que puede proporcionar el freno (almacenado en el atributo *maxPower*), el esfuerzo máximo de frenado en newton que puede generar el freno en rueda (en el atributo *maxEffort*), y la velocidad a la que el freno se desactiva (en *minSpeed*).

### 1.1.5 - curvingLimitation

Con este elemento se dan las definiciones de las limitaciones físicas que se puedan aplicar a los vehículos con relación a los radios de curvatura de la vía tanto en vertical como en horizontal. Esto se hace a través de tres datos definidos en los atributos *horizontalCurveRadius, verticalCrestRadius* y *verticalSagRadius*, definiendo respectivamente el mínimo radio en metros que el tren puede enfrentar en una curva horizontal, en una curva vertical tipo CREST, y en una curva horizontal tipo SAG.

Este elemento no tiene ningún descendiente dado que queda completa y perfectamente definido con los tres radios que se especifican en él.

### 1.1.6 - maintenanceIntervals

Elemento contenedor que sirve para agrupar varios tipos diferentes de tareas de mantenimiento que se deben llevar a cabo en el vagón. Esta estructura tiene un sentido, dado que no se maneja necesariamente el mismo intervalo entre mantenimientos para los frenos que para el sistema de apertura de puertas, por ejemplo. Para ello, se deberá crear un elemento <maintenanceInterval> para cada uno de los diferentes tipos de mantenimiento que se deban hacer al vagón.

#### 1.1.6.1 - maintenanceInterval

Elemento importante que da la información referente a los ciclos de mantenimiento e intervalos que el vagón puede estar entre mantenimientos de un tipo.

El elemento se define con los siguientes atributos: *id*, *code*, *name*, *description*, *xml:lang*, *maximumIntervalDays* con un entero positivo marcando el número de días que puede transcurrir entre dos mantenimientos consecutivos del tipo en cuestión, y

*maximumIntervalDistance* señalando la máxima distancia recorrida entre un mantenimiento y el siguiente del mismo tipo.

Aunque en otros casos los atributos nombre y descripción no cobran tanta importancia, en este caso los períodos de mantenimiento dependerán de cuál sea este (como se ha dicho anteriormente, no será el mismo intervalo para un sistema del tren que para otro), por lo que es necesario especificar de qué mantenimiento se trata (si bien esta distinción, a nivel código, queda hecha por los IDs, a la hora de reconocer dichos mantenimientos un ID no está hecho para aportar la información necesaria para reconocer dichos mantenimientos, sino para identificar elementos entre sí).

#### 1.1.6 - loadLimitMatrix:

Elemento contenedor que sirve para definir varios posibles límites de carga para el vehículo. En este elemento solo se definirá la velocidad máxima permisible para el chasis del vagón, sin tener en cuenta las capacidades de freno, en el atributo *chassisSpeed* mientras que la definición de los límites de carga como tal se harán en subelementos <loadLimit>. Se habla en plural porque se pueden definir varios límites en función de la velocidad y del tipo de línea.

##### 1.1.6.1 - loadLimit:

En este elemento se define la velocidad a la que se evalúa la carga máxima que se va a valorar en los subelementos. Esta velocidad máxima se almacena en el atributo *maxSpeed*. Contiene dos subelementos, que son <railNetwork> y <lineClassification>.

###### 1.1.6.1.1 - railNetwork:

Elemento opcional que contiene la designación de la red a la que afecta este límite de carga. Dicha designación se guarda por medio del nombre de la red en el atributo *name*.

###### 1.1.6.1.2 - lineClassification:

Element crucia para la definición de los límites de carga, puesto que en él es donde se define el límite en sí mismo. Por un lado, en el atributo *payLoad* se definirá la máxima carga admisible en toneladas y, por otro, en el atributo *name* se guardará la clasificación de la línea con respecto a la denominación RIV[16,17], según la cual se dará uno de los valores, [A, B1, B2, B3, B4, C1, C2, C3, C4, CM2, CM3, CM4, D2, D3, D4, E, other:anything] según sea su categoría. Ambos atributos son, obviamente, obligatorios.

#### 2 - formations:

Elemento contenedor de las definiciones de los trenes al completo, como conjunto de vagones definidos en los puntos anteriores. Puede contener todos los elementos <formation> que se necesiten, sabiendo que cada uno de estos elementos representa por sí mismo un tren. Este elemento no contiene atributos.

[16] https://www.tuv-akademie.at/fileadmin/dateien/Downloads/07_EBT_120202_Technische_Normung_Endlicher_V2.pdf
[17] https://uic.org/cdrom/2007/7_siafi2007/docs/april/2_mardi/atelier_bSchmitt_cuusiafi07_en.pdf

### 2.1 - formation

Elemento que representa la formación de un tren a partir de los vagoes de los puntos anteriores. Su definición es larga por su complicación, y comienza por los atributos de este elemento, que son : *id*, *code*, *name*, *description*, *xml:lang*, *formationCount* modelando la posibilidad de que un tren se repita en varias ocasiones (o una parte del tren) aunque normalmente se modele como 1, *length* con la longitud del tren al completo en metros*, speed* guardando la velocidad máxima que el tren está permitido a alcanzar*, bruttoWeight* guardando el peso del tren más su carga en toneladas*, nettoWeight* para el peso en toneladas de la carga, *tareWeight* guardando el valor del peso en toneladas del tren en vacío.

#### 2.1.2 - trainOrder

Elemento contenedor de subelementos <vehicleRef>. Este elemento da una idea de cuál es el orden de los vagones en el tren que va a transitar.

##### 2.1.2.1 - vehicleRef

Da el orden en sí mismo por medio de cinco atributos: *orderNumber* para dar el puesto en el que se encuentra el vagón que se va a definir en el siguiente atributo*, vehicleRef* es una referencia por ID al vagón (o tren, ya que se puede formar un tren a través de una unión de dos trenes) que va a formar parte del tren que se está definiendo*, vehicleCount* es un atributo que se puede utilizar para repetir *n* veces lo definido en este elemento de manera que se pueda decir que hay *n* vagones como éste seguidos*, frontGangway* y *rearGangway* representan con booleanos si el vagón que se define tiene o no una pasarela en la parte delantera o en la parte trasera.

#### 2.1.3 - categoryRef

Con su atributo *ref* asocia este elemento a un elemento <category> (o más, si se definen más) del esquema de timetable, por medio de su ID. No tiene más atributos ni descendientes.

#### 2.1.4 - trainEngine

Elemento que da valores importantes de la aceleración del tren completo. Se debe asegurar que estos datos son consistentes con los dados en los valores de los vagones. Se define por dos atirbutos que son *trainMeanAcceleration* que contiene la aceleración media de la formación en metros por segundo cuadrado contando todo el espectro de velocidades para el cálculo y *trainMaxAcceleration* que contiene el valor máximo de aceleración del tren.

#### 2.1.5 - trainBrakes

Elemento que describe los sistemas de freno del tren. Para ello, contiene los atributos *brakeType, airBrakeApplicationPosition, regularBrakeMass, emergencyBrakeMass* (ya vistos en 1.1.4.1 - vehicleBrake>), y *maxDeceleration* cuantifica el valor de la deceleración máxima en metros por segundo al cuadrado, y *meanDeceleration* con el valor medio de la deceleración calculada en todo el espectro de velocidades.

#### 2.1.5.1 - auxiliaryBrakes

Aporta datos sobre los frenos auxiliares, del mismo modo que se hace en el elemento 1.1.4.1.1 - auxiliaryBrakes>.

#### 2.1.6 - trainResistance

Elemento importante que guarda, en un elemento tipo tabla, los valores de resistencia al avance del tren (teniendo en cuenta todas las resistencias) según la velocidad y la inclinación.

#### 2.1.6.1 - valueTable

Tabla que recoge el valor de la resistencia al avance según velocidad e inclinación.

### 6.1.3 Elementos de Timetable and Rostering

Este esquema es el encargado de dar la información necesaria referente a la organización temporal (tanto horaria como diaria) de la red de ferrocarriles. En el caben desde la información de los días de fiesta (o de servicio restringido) hasta los horarios de los trenes con un enfoque de venta de billetes.

Para definir cualquiera de los otros esquemas, este necesita una definición puesto que se puede dar el caso de tener más de un elemento timetable a la vez. Además, se puede (y se debe siempre que se pueda) hacer una relación entre los esquemas. Puesto que, con lo visto hasta ahora, uno define la infraestructura, otro el material rodante, y el otro define su funcionamiento, es lógico que se pueda establecer una relación entre ellos de manera que se denote claramente qué tren pasará por qué vía en qué momento. Para esta definición, se utilizan atributos como son *id*, *code*, *name*, *description*, *xml:lang*, *version*, *xml:base*, *rollingstockRef* (vistos, por ejemplo, en la definición del esquema infraestructure), *infrastructureRef*

#### 1 - additionalName:

Posible nombre adicional que se le pueda dar al horario. Puede servir para almacenar dicho nombre en otro (u otros) idiomas, para dar un nombre común por el que se pueda conocer, o cualquier otro tipo de nombre que se pueda precisar. Su funcionamiento ha sido explicado ya en otros puntos del texto.

#### 2 - timetablePeriods:

Elemento contenedor sin atributos que agrupará diferentes períodos con la misma distribución horaria. Es muy importante tener claro que los períodos que se definan nunca deben superponerse, ni coincidir en el tiempo (sí es posible tener dos perídos que contengan el mismo día puesto que existe la definición de la hora a la que comienza o termina el período).

#### 2.1 - timetablePeriod:

Cada uno de los elementos <timetablePeriod> que se definan será un conjunto de días (más o menos largo) que cumplen con la misma organización horaria y semanal entre sí. Esto significa que si, por ejemplo, en verano hay un horario y en invierno otro, se podrán generar dos

elementos diferentes, definidos propiamente, con la información relatva a dichos horarios de ambos períodos, así como sus fechas especiales (vacaciones, días de más servicio, días de menos servicio, etcétera).

Tiene los siguientes atributos: *id*, *code*, *name*, *description*, *xml:lang*, *startDate* que guarda el día del año en que comienza dicho período, *endDate* que almacena el último día del período que se define, y *startTime* y *endTime* almacenando la hora concreta del día a la que comienza y termina el período. Estos cuatro atributos se definen con tipos de dato específicos, que son aaaa-mm-dd para la fecha (es decir, año con cuatro cifras, mes en dos cifras y días también en dos cifras, separados por guiones simples), y hh:mm:ss para la hora (hora, minutos y segundos con dos puntos para separación).

#### 2.1.1 - holidays:

Elemento contenedor sin atributos. Agrupa elementos <holiday>

#### 2.1.1.1 - holiday:

Elemento que define períodos o días sueltos en los que el servicio definido no se proveerá. Se define por dos atributos que son *holidayDate* para guardar la fecha (mismo formato aaa-mm-dd), y *description* (opcional) con una breve explicación sobre qué festivo es el definido.

#### 3 - operatingPeriods:

Elemento contenedor que agrupa elementos del tipo <operatingPeriod>.

#### 3.1 - operatingPeriod:

Descripción de la operación de un tren. Esto se puede hacer basado en días de calendario, o en una semana abstracta. Para su definición contiene dos subelementos y varios atributos, que son *id*, *code*, *name*, *description*, *xml:lang*, *timetablePeriodRef* (como referencia del elemento timetablePeriod que se una a este período operativo)*, startDate, endDate* y *bitMask* que representa una máscara de bits, es decir, una cadena de ceros y unos en la que se representa cada día del período, señalando con un uno cada día que el servicio se ofrece como se define, y con un cero un día que el servicio no se ofrece como se define. Es importante saber que este último atributo no representa una semana, sino toda la duración del período que se define en el elemento operatingPeriod referenciado. Es decir, si el período referenciado (<timetablePeriod>) contiene 20 días, comenzando por un lunes, y el servicio se ofrece de lunes a jueves, bitMask tendría una apariencia como 11110001111000111100 (con cada número se rellena la posición de un día de la cadena LMXJVSDLMXJVSDLMXJVS). Si además, en ese período, el segundo martes fuera fiesta y, por ello, no hubiera servicio, la cadena se transformaría en 11110001011000111100.

El comienzo y final de operatingPeriod deben estar contenidos o ser coincidentes con los de timetablePeriod. En caso de que la duración de operatingPeriod sea menor que la duración de timetablePeriod, se deberá modelar el atributo bitMask con todos los días del elemento timetablePeriod, señalando con ceros aquellos días que no se incluyan en el período de operatingPeriod.

### 3.1.1 - operatingDay:

Elemento que da una visión semanal de un periodo operativo para un día normal. Los días que se salgan de esta definición (como vacaciones, por ejemplo) quedan definidos en el siguiente elemento. El elemento puede ser utilizado más de una vez, siempre que esté correctamente definida la validez de cada uno de ellos con los atributos startDate y endDate

Los atributos que se utilizan en este elemento son *operatingCode* para guardar una clave de tipo bit mask que modela con ceros y unos una semana (siete cifras) *, onRequest* modela si los días que se encuentran con un cero en el atributo anterior pueden tener tráfico bajo demanda*, startDate* y endDate*.*

#### 3.1.1.1 - operatingDayDeviance:

Utilizado para marcar días dentro del elemento anterior que no cumplen con la definición habitual dada en el atributo operatingCode. Esta desviación se modela teniendo en cuenta el modelo de días festivos que se ha hecho previamente en 2.1.1 - holidays. Es importante entender bien el funcionamiento de este elemento para su correcto modelado puesto que, aunque más complicada, esta manera de modelado es más útil y efectiva que el modelado día a día. Contiene cuatro atributos *operatingCode, onRequest, holidayOffset* y *ranking.* Los dos primeros ya se han visto. El tercero da una relación entre el día que se modele y un dia festivo de modo que si su valor es 0, esa definición será válida para los días festivos, si es 1, se implicarán los días siguientes a festivos, si es -1 serán las vísperas de festivos (y así sucesivamente). El cuarto atributo, por su lado, da una relación de prioridad en caso que haya definiciones contradictorias. De este modo, si se establece un elemento como el siguiente, se implicará que el horario normal definido aquí afecta de lunes a viernes (por el 1111100 de la primera línea) pero que cerrará cualquier día de la semana que sea festivo (segunda línea)

```
<operatingDay operatingCode="1111100"/>
        <operatingDayDeviance operatingCode="0000000" holidayOffset="0"/>
</operatingDay>
```

Si se eligiese el código 0011100 y el valor de offset -1 significaría que los días previos a festivos que además sean lunes, martes, sábados o domingos, el tren no dará este servicio; pero si esas vísperas coinciden con un miércoles, jueves o viernes, sí habrá servicio (esta diferenciación viene de la mano del código 0011100 que indica qué pasará en función de qué día de la semana coincida.

### 4 - categories:

Elemento contenedor donde se almacenarán los elementos <category>. No tiene atributos

### 4.1 - category:

Elemento diseñado para definir un tipo de tren, o categoría de tren. Este elemento es vital, para cualquier línea que vaya a tener más de un tipo de tren. Además, en ocasiones, se buscará tener información de cuáles son las características de un tren en concreto (por ejemplo, si tiene plazas para bicicletas, y esas consultas se harán por medio de referencias a este elemento. Para definirlo, se necesitan los siguientes atributos *id*, *code*, *name*, *description*, *xml:lang*, *trainUsage*

guardando el tipo de uso que se le va a dar entre pasajeros, mercancías o mixto [passenger, goods, mixed], *deadrun* será un booleano que marcará si el tren va vacío por fines operacionales*, categoryPriority* para describir la importancia de esta categoría en relación a las otras (por ejemplo, si definimos trenes de alta velocidad y trenes de corta distancia, obviamente se establecerá este valor a 1 para el tren de alta velocidad, y un 2 o más para el tren de corta distancia, para denotar que el primero es más prioritario).

#### 4.1.1 - additionalName:

Nombre adicional que se le puede otorgar a la categoría.

### 5 - annotations:

Contenedor de elementos sin atributos.

#### 5.1 - annotation:

Elemento que contiene texto general dedicado a mensajes a mostrar a los pasajeros o al maquinista. Este es, como la definición de la radio usada, uno de esos elementos que usan el propio valor del elemento para guardar el mensaje que se quiera dar a los pasajeros y usuarios. Por eso, si se quiere guardar el mensaje "Bienvenidos al tren de la Fresa", se tendrá que modelar el siguiente código (usando los atributos *id*, *code*, *name*, *description*, *xml:lang)*

*<annotation id='msg_Fresa_01' code='msg_bvnidos' xml:lang='ES'> <text> Bienvenidos al tren de la Fresa </text> </annotation>*

#### 5.1.2 - text:

Este será el texto que se ha de dar (o bien al maquinista, o bien a los pasajeros). Su atributo opcional es *xml:lang*. En el elemento anterior se muestra un ejemplo de cómo debería ser llamado este elemento para generar la nota de texto.

### 6 - trainParts:

Elemento contenedor sin atributos.

#### 6.1 - trainPart:

Descripción de la parte más básica del tren, de modo que no se permita un cambio de configuración durante el paso del mismo. Contiene los atributos *id*, *code*, *name*, *description*, *xml:lan, line* describiendo la línea del trainPart (que puede ser diferente de la línea del tren) , *trainNumber* con el número de tren que se le asocia al tren que usa esta parte o a la parte en sí(se recomienda no utilizar este atributo dado que su función queda ampliamente cubierta por otros atributos en el elemento train)*, additionalTrainNumber* para proveer unicidad al elemento trainNumber*, cancellation* indicat si este elemento se encuentra cancelado por alguna razón , *debitCode* es el código del deudor para consideraciones financieras (en el caso práctico se ha pasado por alto dado que no tiene mucha importancia en él, y tampoco tiene implicaciones más allá de este elemento)*, remarks* se utiliza como campo libre para aclaraciones o notas*, timetablePeriodRef* contiene el valor del ID con que se referencia al timetablePeriod que se asocia a este elemento (esto significa que, si este atributo está relleno, esta parte del tren se

regirá por los horarios especificados en el elemento timetablePeriod referido) y *processStatus* cuyos valores [planned, actual, calculated, toBeChecked, changed, imported, other:anything] señalan el estado de trabajo que trainPart conlleva.

Además de dichos atributos, el elemento trainPart contiene varios subelementos, que se definen a continuación.

#### 6.1.1 - additionalName:

Nombre opcional que se le puede dar al elemento en el que está englobado.

#### 6.1.2 - formationTT:

Descripción del uso que un tren (elemento formation) le da a un trainPart. Importante aclarar que para que este elemento tenga efecto alguno sobre la información que proporciona el código, el elemento gemelo <formation> del esquema Rollingstock debe estar también definido.

La definición de este elemento se hace básicamente por los siguientes elementos:

- *formationRef* con el valor del ID que define el elemento <formation> al que se refiere
- *weight* guarda el peso real del tren cuando esté en uso, incluyendo los motores, en toneladas
- *load* almacena el peso real de la carga del tren, excluyendo los motores
- *length* con la longitud máxima de la formación en uso
- *speed* para la velocidad máxima del tren en uso
- *timetableLoad* peso en toneladas de la carga de acuerdo con el horario
- *orientationReversed* es un booleano que marca si la formación se encuentra siendo usada en el orden inverso al indicado.

##### 6.1.2.1 - equipmentUsage:

Elemento contenedor. Entre él y todos sus subelementos <equipment> se definirá cuál es la forma en que el equipamiento es usado por el tren o algún vagón en particular.

###### 6.1.2.1.1 - equipment:

Descripción de los requerimientos técnicos que el tren hace al equipamiento. Su definición queda completa por medio de los atributos *type* para una especificación del tipo de equipamiento que se requiere con una referencia a uno de los valores de la lista provista de tipos de equipamiento de seguridad[18], *uses* es un booleano que define si el equipamiento está en uso (verdadero) o solo está presente (falso) y *description* para guardar una descripción más detallada. Este elemento se verá, exactamente igual, en la definición del tren (trainPartSequence).

###### 6.1.2.1.1.1 - ects:

Elemento que sirve para completar la definición de los requerimientos técnicos, con una descripción de los niveles del sistema etcs utilizado por el tren, con respecto al horario que se

[18] https://wiki.railml.org/index.php?title=Dev:TrainProtectionSystems Esta parte todavía se encuentra en desarrollo, por lo que su funcionalidad no es completa.

está definiendo. Este elemento ya se ha utilizado anteriormente en la definición del tren dentro del esquema de rollingstock. Esto es así porque, si bien un tren puede tener *n* sistemas con *m* niveles de  etcs, cuando este tren se sitúe en una vía para hacer un recorrido determinado, estos niveles pueden ser o no utilizables en función de la vía, de modo que se hace necesaria esta diferenciación.

Su definición, y el modo de uso de los atributos quedó definido en los elementos 7.1.7.1.1.6 - <etcs> incluidos en la definición de las señales de la vía.

#### 6.1.2.2 - passengerUsage:

Elemento destinado a recoger la información del uso del tren por parte de los pasajeros. Esto se hace por medio de los descendientes, puesto que este elemento no tiene atributos.

##### 6.1.2.2.1 - places:

Información relativa a las plazas del tren que ya ha sido hecha en el elemento 1.1.3.6.2 - <places>, sin embargo, igual que en el elemento etcs, la definición no es exactamente la misma aunque sí su uso. Esta diferencia de definición es la respectiva a la diferencia que hay entre las plazas disponibles en un tren, y las plazas que vaya a haber utilizables en una ruta determinada. Si pensamos en asientos, tal vez no tenga mucho sentido, pero el elemento places no define solo asientos, sino también sitios para bicicletas, aseos, camas, etcétera; que pueden estar o no disponibles en función del recorrido, o la longitud del mismo, por ejemplo.

##### 6.1.2.2.2 - services:

Elemento que también ha sido definido anteriormente en 1.1.3.5.1 - service. De la misma manera que se ha mencionado en los elementos anteriores, existe la misma diferencia entre el elemento utilizado aquí, y en la definición del tren.

#### 6.1.2.3 - reservationInfo:

Elemento contenedor sin atributos que abarcará los datos referentes a la información de reservas y compra de billetes.

##### 6.1.2.3.1 - booking:

Elemento comprendido dentro de reservationInfo, que detalla la información relativa a un vehículo de manera específica. Esta definición se hace a través de tres atributos, que son *bookingNumber* que almacenará el número de reserva, que podrá ser transferido al sistema de reservas, *posInFormation* que contendrá la posición del vehículo en la formación, que diferenciará al vagón en caso de que haya más de un vagón del mismo tipo en la formación; y por último *vehicleRef* que tendrá la referencia del ID del elemento asociado <vehicle> (es decir, será la referencia a en qué vehículo se hace la reserva).

### 6.1.3 - operatingPeriodRef:

Elemento simple que va a relacionar la definición que se está haciendo con el elemento operatingPeriod que se necesite relacionar en cada caso. Esto se completa con el atributo *ref* que, como ya se sabe, indica el valor del ID del elemento que se quiere relacionar.

#### 6.1.4 - ocpsTT:

Elemento contenedor sin atributos.

##### 6.1.4.1 - ocpTT:

Elemento que describe un punto en el camino del tren donde está situado un ocp (descrito previamente). Este elemento se utiliza, entre otras cosas, para saber a qué hora se espera que cada tren pase por cada ocp, para saber qué conexiones se pueden hacer a través de dicho ocp, o para guardar ciertas estadísticas de uso, por ejemplo.

Los atributos que se necesitan para rellenar este elemento son *ocpRef* con la referncia al ocp que define, *trackRef* para identificar la vía que va a ser definida en el elemento, *trackInfo* para describir textualmente la línea, *ocpType* valor de [stop, pass, beggin, end] con la definición de si el tren para, en este ocp, pasa de largo, comienza o termina en él; *remarks, trainReverse* definiendo si el tren cambia de dirección en la estación que se está modelando, *alignment* almacena un valor de la lista [head, center, rear] definiendo si el punto kilométrico que se da en el ocp coincide con la cabeza, centro o final del tren; *offset* definiendo un posible margen en metros con respecto al atribut anterior, y *shuntingTime* para definir los tiempos de maniobra que se toman en el ocp, si es necesario.

Es importante ver que aquí se puede definir una tren que pare o no en según qué estaciones gracias al atributo *ocpType*.

###### 6.1.4.1.1 - times:

Elemento que señala los tiempos de llegada y salida del ocp en su alcance de actuación. Para dicha definición tiene cinco atributos que son *scope* [actual, calculated, published, scheduled, earliest, latest, other:anything] señalando si la configuración descrita es respectiva al horario real, calculado, publicado, programado, más temprano, más tardío, u otros; *arrival* señalando la hora de llegada al ocp (no se debe dar si el tren no para en la estación o si la estación es de partida); *departure* señalando la hora a la que el tren sale del ocp, y los atributos *arrivalDay* y *departureDay* complementando el los valores de las horas de llegada y salida, con un número que es el número de días que transcurren desde que el tren sale, hasta que acontece el *arrival* o el *departure.* Esto es así para asegurar la posibilidad de modelar trenes que, durante su trayecto estén en funcionamiento en más de un día, tomando siempre como base el día de salida del primer ocp (por ello el elemento departureDay del primer ocp debe ser siempre 0). Por ejemplo, si un tren sale de una estación A a las 22:00 del martes, pasa por una estación intermedia llegando a las 23:55 del martes y saliendo de ella a las 00:15 del miércoles y llega a su destino C a las 01:30 del miércoles, su modelado deberá llevar la siguiente secuencia:

```
<ocpTT ocpRef='ocp_A'>
        <times scope='actual' departure='22:00:00' departureDay='0'/>
</ocpTT>
<ocpTT ocpRef='ocp_A'>
        <times scope='actual' arrival='23:55:00' arrivalDay='0' departure='00:15:00'
        departureDay='1'/>
</ocpTT>
<ocpTT ocpRef='ocp_A'>
```

```
        <times scope='actual' arrival='01:30:00' arrivalDay='0'/>
</ocpTT>
```

#### 6.1.4.1.2 - connections:

Elemento contenedor sin atributos, que servirá para definir todas las conexiones de diferentes líneas que coexisten en el ocpTT relativo que incluye a este elemento.

#### 6.1.4.1.2.1 - connection:

Elemento que se puede modelar para definir cada una de las conexiones que se pueden efectuar en el ocpTT que contiene a este elemento. Esto significa que por cada una de las conexiones que se puedan efectuar, se generará una línea más. Es trivial ver que si un ocpTT tiene trenes en una misma línea en dos sentidos, no es necesario modelarsus *conexiones* mientras los dos sentidos se produzcan en una misma línea (así pues, dependiendo de la definición de línea, a veces será obligatorio hacer esta distinción si se quiere tener constancia de esa posibilidad de cambio). Contiene los siguientes atributos: *trainRef* con el ID del elemento train asociado, *minConnTime* con el mínimo tiempo que se necesita para la operación de conexión, *maxConnTime* guardando el tiempo máximo planeado para dicha conexión, *connType* [commercial, operational] define si el tipo de la conexión es comercial u operacional, *connOperation* almacenando el valor [none, meet, IsWaitingFor, IsExpectedBy, other:anything] según si la operación de conexión es no específica, si los dos trenes coinciden en dicho ocp, si el tren espera a que llegue otro tren puesto que hay pasajeros que deban hacer el trasbordo, si otro tren estará esperando a que dicho tren llegue, u otros modos; *ocpRef* con la referencia del ID del ocp al que se refiere*, trainPartRef* señalando al ID del trainPart relativo a este elemento*, operatingPeriodRef* referido al ID del elemento operationPeriod*, notGuaranteed* guardando un booleano que indica si hay riesgo de que la conexión no esté garantizada*, nonConnection* marca que dicha conexión no es correcta (en caso de cambios, por ejemplo) y *samePlatform* es un booleano que indica si la conexión es en el mismo andén.

#### 6.1.4.1.2.1.1 - externalReference:

Elemento contenedor sin atributos que se puede utilizar para definir que la conexión se establece con un elemento externo a la red definida (si por ejemplo, la conexión es entre un tren de la red nacional con un metro urbano). Es importante entender que, aquí, con externo nos referimos a algo que no esté identificado dentro del mismo archivo RailML®.

#### 6.1.4.1.2.1.1.1 - tafTapTsiTrainID:

Elemento que hace referencia a la definición del tren externo con el cual se produce la conexión. Sus atributos son *objectType* [TR, PA, CR, PR] estableciendo la definición de con qué tipo de objeto se hace la conexión; *companyCode* contando con qué código ID se representa a la compañía gestiona el tren; *core* manteniendo la parte principal del identificador determinado por la compañía; *variant* con una relación entre dos identificadores, *timetableYear* refiriendo al periodo (año) en que las conexiones pueden ser efectuadas *y startDate.*

#### 6.1.4.1.2.1.1.2 - trainNumber:

Referencia al número externo del tren con el que se establece la conexión. El atributo *trainNumber* recoge dicho dato.

#### 6.1.4.1.2.1.1.2.1 - organizationalUnitBinding:

Información que se da sobre los metadatos del tren que se referencia como conexión, cuando ese tren es de una red externa.

#### 6.1.4.1.2.1.1.2.1.1 - vehicleOperator:

Referencia al operador del vehículo por medio de su ID, guardado en el atributo *ref*.

#### 6.1.4.1.2.1.1.2.1.2 - customer:

Referencia al cliente o usuario (compañía) del vehículo por medio de su ID, guardado en el atributo *ref*.

#### 6.1.4.1.2.1.1.2.1.3 - railwayUndertaking:

Referencia al operador del vehículo por medio de su ID, guardado en el atributo *ref*.

#### 6.1.4.1.2.1.1.2.1.4 - operationalUndertaking:

Referencia al operador de la red del vehículo por medio de su ID, guardado en el atributo *ref*.

#### 6.1.4.1.2.1.1.2.1.5 - concessionaire:

Referencia al empresa concesionaria del vehículo por medio de su ID, guardado en el atributo *ref*.

#### 6.1.4.1.2.1.1.2.1.6 - contractor:

Referencia a la empresa que contrata el vehículo por medio de su ID, guardado en el atributo *ref*.

#### 6.1.4.1.2.1.1.3 - lineNumber:

Informa acerca de la línea con la que se hace la conexión. Su atributo es *lineNumber*, que guarda el valor del número de línea.

#### 6.1.4.1.2.1.1.4 - information:

En su atributo *description* guarda una posible información adicional que se tenga sobre la conexión.

#### 6.1.4.1.2.1.2 - annotationRef:

Elemento usado para referenciar al ID de algún <annotation> que tenga influencia en esta conexión. Su definición se hace por medio de *ref* con el ID del elemento, y *operatingPeriodRef* guardando el ID del elemento <operatingPeriod> durante el cual es válida la nota.

#### 6.1.4.1.3 - statistics:

Elemento contenedor de <statistic> sin atributos.

##### 6.1.4.1.3.1 - statistic:

Elemento contenedor que se puede utilizar para almacenar datos estadísticos (de uso principalmente). Cada uno de los elementos presentes en este (salvo statisticAnalyses), contendrá los atributos:

- *arrival:* hora de llegada
- *arrivalDay:* día de llegada
- *departure:* hora de salida
- *departureDay:* día de salida
- *arrivalDelay:* retraso de llegada*
- *departureDelay:* retraso de salida*
- *stopTime*: tiempo invertido en la parada*

Representando en su caso, diferentes figuras estadísticas (media, mediana, etc.). Los atributos con un asterisco serán de tipo xs:duration PThhHmmMssS (por ejemplo, 12 horas y 6 minutos será PT12H06M00S, o un año, un día y seis horas será P01Y00M01DT06H00M00S).

###### 6.1.4.1.3.1.1 - mean:

Media aritmética de cada uno de los valores presentes por medio de sus atributos.

###### 6.1.4.1.3.1.2 - median:

Mediana de cada uno de los valores presentes por medio de sus atributos.

###### 6.1.4.1.3.1.3 - standardDeviation:

Desviación típica de cada uno de los valores presentes por medio de sus atributos.

###### 6.1.4.1.3.1.4 - statisticAnalyses:

Representa una conclusión formal de los datos estadísticos, que se pueden presentar. Contiene los atributos description, *percentage, value* y *figure;* guardando respectivamente la descripción de la figura que se estudia, su porcentaje, su valor, y qué tipo de figura de análisis contiene.

#### 6.1.4.1.3 - sectionTT:

Este elemento describe datos referentes a la manera en que un ocpTT . Sus atributos son *section* guarda información que descirbe la secicón usada, *lineRef* es una referencia al ID del elemento <line> involucrado, *trackInfo* es una descripción de la sección usada, description guarda una descripción libre, *remarks* contiene datos adicionales, *percentageSupplement* con un porcentaje del tiempo de servicio suplementario planeado, y *distance*, con la distancia al siguiente ocpTT en metros.

#### 6.1.4.1.3.1 - trackRef:

Elemento que refiere a cada una de las vías utilizadas por el tren que se está definiendo. Esta se relaciona con el tren por medio de dos atributos, que son *dir* y *ref*. El primero define la dirección (con respecto al kilometraje) que sigue la definición que se está dando según los valores [up, down] si la circulación será en sentido ascendente o descendente, mientras que el segundo relaciona la pista mediante el ID del elemento <track>.

#### 6.1.4.1.3.2 - runTimes:

Descripción de la estructura interna del tiempo hasta el siguiente ocpTT. Se define por medio de tres atributos: *minimalTime* es el tiempo mínimo de funcionamiento*, operationalReserve* define el tiempo adicional para propósitos operativos, y *additionalReserve* define tiempo adicional extra sin atender a ninguna disposición preestablecida.

### 6.1.5 - organizationalUnitBinding:

Este elemento contenedor se usa para describir las unidades organizativas que son responsables de este trainPart. Para esta descripción utiliza los descendientes que se detallan a continuación. Este elemento y sus descendientes quedan exactamente definidos en el punto 6.1.4.1.2.1.1.2.1 - <organizationalUnitBinding> aunque, esta vez, definiendo propiedades de un tren de la misma red. (Por tanto, no se profundizará en los siguientes seis elementos ya vistos).

#### 6.1.5.1 - vehicleOperator

Visto en el punto 6.1.4.1.2.1.1.2.1.1 - <vehicleOperator>

#### 6.1.5.2 - customer

Explicado en el punto 6.1.4.1.2.1.1.2.1.2 - <customer>

#### 6.1.5.3 - railWayUndertaking

Definido en el punto 6.1.4.1.2.1.1.2.1.3 - <railWayUndertaking>

#### 6.1.5.4 - operationalUndertaking

Analizado en el punto 6.1.4.1.2.1.1.2.1.4 - <operationalUndertaking>

#### 6.1.5.5 - concessionaire

Visto en el punto 6.1.4.1.2.1.1.2.1.5 - <concessionaire>

#### 6.1.5.6 - contractor

Ya estudiado en el punto 6.1.4.1.2.1.1.2.1.1 - <contractor>

### 6.1.6 - annotationRef:

Elemento que permite referir a notas en el elemento <annotation>. Esto se hace por medio de dos atributos, que son *ref* para almacenar el ID del elemento annotation asociado, y *operationPeriodRef* con el ID del periodo en que esta anotación es válida.

### 6.1.7 - xs:any:

Atributo extra para añadir información de cualquier tipo. Para su uso, se precisan ciertos cambios en el esquema de definición que se utiliza, de modo que no se utilizará en el ejemplo práctico que se ha dado.

## 7 - trains:

Elemento contenedor sin atributos, en el que se definirán elementos <train>

### 7.1 - train:

Este elemento define un tren desde diferentes puntos de vista. Este tren será, básicamente, una sucesión de elementos trainPart. Se define por medio de los siguientes atributos: *id*, *code*, *name*, *description*, *xml:lang*, *type* como referencia al tipo de tren [operational, commercial] según la definición del tren como operacional o comercial, *trainNumber* guarda el número del tren para su identificación, *additionalTrainNumber* guardando un posible número adicional para dar unicidad al atributo trainNumber; *scope* mantiene una distinción entre trenes operacionales que tengan el mismo número de tren, eligiendo un valor de entre [primary, secondaryStart, secondaryEnd, secondaryInner] según si la ruta es la principal, una ruta secundaria al comienzo de la misma, secundaria al final, o secundaria en uan sección intermedia; *processStatus* define el estado del tren en relación a su estado de funcionamiento [planned, actual, calculated, toBeChecked, changed, imported, other:anything] para marcar si el horario está descrito según lo planeado, según el horario real, según se haya calculado, si necesita ser comprobado, si ha sido cambiado, o si ha sido importado; el siguiente atributo es *remarks* que sirve para incluir notas e inclusiones a posteriori, y *cancellation* que guarda un booleano que señala si el tren no es válido por su cancelación.

### 7.1.1 - additionalName:

Elemento ya visto en varias ocasiones

### 7.1.2 - trainPartSequence:

En este elemento se da una agrupación de todos los elementos <trainPart> que pertenecen al tren que se define. Se define por medio de cuatro descendientes, y los siguientes atributos: *sequence* sirve para dar el orden de los sucesivas partes del tren a lo largo de la ruta[19], *pathStatus* señala si el estado del trayecto del tren según [planned, ordered, confirmed, detailsRefused, cancelled, notAvailable, other:anything] (planeado, pedido, confirmado, detalles rechazados, cancelado, no disponible, otros), y el atributo *categoryRef* que hace referencia al ID del elemento <category> asociado, que define muchas de sus propiedades (esta categoría puede ser diferente de la categoría que sus <trainPart> utilicen).

[19] Es importante destacar que esta secuencia no debe confundirse con la secuencia de vagones en un tren, sino que define los trenes que tienen un mismo punto de salida y llegada.

#### 7.1.1.1 - trainPartRef:

Este elemento da un listado de todos los elementos trainPart que componen la secuencia de trenes. Contiene dos atributos que son *ref* y *position* que indican, respectivamente, el ID que referencia al elemento trainPart, y la posición que ocupa en este tren.

#### 7.1.1.2 - equipmentUsage:

Elemento contenedor que describe el equipamiento del tren que es usado en un trayecto determinado (el que se está modelando). Este elemento y sus descendientes se han visto ya anteriormente en 6.1.2.1 - <equipmentUsage>.

#### 7.1.1.2.1 - equipment y 7.1.1.2.1.1 - etcs:

Vistos previamente.

#### 7.1.1.3 - brakeUsage:

Aquí se define cada uno de los sistemas de freno, igual que se ha hecho a la hora de definir el material rodante (1.1.4.1 - vehicleBrake), pero en este caso, dando valor solamente a los sistemas de freno que entran en el contexto del uso que se está dando a dicho tren. Esta definición se hace a través de los atributos *brakeType*; *airBrakeApplicationPosition*; *regularBrakeMass*, *emergencyBrakeMass*, *maxDeceleration*, *meanDeceleration* (que ya se han visto previamente) añadiendo dos atributos que son *regularBrakePercentage* y *emergencyBrakePercentage* que dan porcentajes (ratios) de la masa de frenado entre la masa total del vehículo.

#### 7.1.1.3.1 - auxiliaryBrakes:

Explicación de los frenos auxiliares que pueden ser utilizados en el trayecto que se está definiendo. Su definición es la misma que se hace en el elemento 1.1.4.1.1 - <auxiliaryBrakes>.

#### 7.1.1.4 - speedProfileRef:

Elemento que asigna un perfil de velocidad principal que se asigna al tren en este recorrido. Se completa por el atributo *ref* que guarda el ID del elemento speedProfile asociado.

#### 7.1.3 - tafTapTsiTrainID:

Elemento que almacena información relativa a la identificación del tren por medio del TransportID según taf/tap TSI, siglas de Technical Specification for Interoperability relating to Telematics Applications for Freight/Passenger Services.

#### 8 - trainGroups:

Elemento contenedor sin atributos.

#### 8.1 - trainGroup:

Elemento que da la definición de cada una de las agrupaciones de trenes que se pueden formar en un horario. Su definición se hace por medio de un subelemento y los atributos *id*, *code*,

*name*, *description*, *xml:lang*, *type* define al grupo según [interval, other:anything] para dar información de si el grupo se rige por intervalos (iguales entre los trenes) o de otra forma, *trainNumber* guarda el número del tren para el grupo y *processStatus* es un atributo que se ha definido ya en 7.1 - train.

#### 8.1.1 - trainRef:

Referencia a los trenes que componen el grupo. Esta referencia se hace por medio de los atributos *ref* y *sequence* que guardan el valor del ID del elemento <train>, y su posición en la secuencia del grupo, respectivamente.

### 9 - rosterings:

Elemento contenedor de elementos <rostering> sin atributos.

### 9.1 - rostering:

Elemento que contiene los datos relacionados con el plan de trenes. Se necesitan los atributos *id*, *code*, *name*, *description*, *xml:lan, vehicleRef* con el ID del elemento <vehicle> asociado, *formationRef* contiene una referencia al ID del elemento <formation>, *depot* tiene el dato de la localización a la que pertenece el rostering, *defaultPreProcessingTime* y *defaultPostProcessingTime* guardando el tiempo que transcurre desde el principio del bloque hasta el comienzo del blockPart y desde el final del blockPart hasta el final del bloque; y el atributo *scope* manteniendo un valor de [conceptional, operational, timetable, other:anything] según si su alcance es de concepto, de operación, o de horario (u otros).

#### 9.1.1 - blockParts:

Elemento contenedor de <blockPart>.

##### 9.1.1.1 - blockPart:

Elemento que contiene los datos relativos a cada parte de los elementos <block> (que se describirán más adelante). Sus atributos son *id*, *code*, *name*, *description*, *xml:lang, begin, beginDay, end, endDay* (ya vistos), *startOcpRef* con el ID que referencia al ocp que se encuentra al principio del bloque, *endOcpRef* guarda el ID del ocp que se encuentra al final del bloque, *trainPartRef, operatingPeriodRef, mission* contiene un valor de [timetable, fullRun, emptyRun, outOfOrder, fillIn, preheating, refuel, shunting, depotrun, standBy, cleaning, maintenance, inspection, other:anything] (horario, trayecto lleno, trayecto vacío, fuera de servicio, en proceso de llenado, en repostaje, en maniobras, depósito, en espera, limpieza, mantenimiento, inspección, otros), *fixed* es un booleano que tiene información de si el bloque ha terminado la inspección o el mantenimiento correctamente, *runLength* con la longitud en kilómetros de funcionamiento, *vehicleRef* y *formationRef* (también vistos).

### 9.2 - blocks:

Elemento contenedor sin atributos.

#### 9.2.1 - block:

Elemento que contiene la información de servicios obligatorios de un tren, conteniendo varios elementos <blockPart>. Sus atributos son *id*, *code*, *name*, *description*, *xml:lang, blockGroupNumber* y *fixed.*

##### 9.2.1.1 - blockPartSequence:

Aquí, se referencian los blockPart, agrupándolos en una secuencia de tareas. Se deben rellenar los atributos *sequence* almacena el orden en que este elemento aparece dentro del elemento <block>*, dayOffset* define el día en que el elemento blockPart empieza si el block funciona durante la noche*, preProcessingTime* y *postProcessingTime* son atributos ya estudiados y *basicBlockRef* contiene el ID del elemento <block> al que hace referencia.

###### 9.2.1.1.1 - blockPartRef:

Con este elemento se refiere el ID del <blockPart> que se quiere agrupar en el elemento block.

#### 9.3 - circulations:

Elemento contenedor de elementos <circulation> sin atributos.

#### 9.3.1 - circulation:

Elemento usado para encadenar bloques para un plan de trayecto completo. Sin este elemento, toda la declaración de elementos previamente hecha carece de utilidad. *blockRef* contendrá el ID del elemento <block> que se relaciona, *startDate* da el valor del día de comienzo del período de validez de este elemento (si este atributo no se encuentra definido, operatingPeriodRef tendrá que estar definido)*, endDate* almacena el valor del día de final del período de validez de este elemento (este período definido no puede sobreponerse a otro)*, operatingPeriodRef* como una referencia al elemento <operatingPeriod> por medio de su ID*, repeatCount* contiene el número de elementos <circulation> que ocurren en el mismo periodo*, vehicleCounter* es un indicador del número exacto de vehículos dentro del elemento circulation*, vehicleGroupCounter* ejerce una indicación del número exacto de grupos de vehículos dentro del elemento <circulation>*, nextBlockRef* es una referencia al siguiente elemento <block> según el sentido de la circulación *, nextOperatingPeriodRef* es una indicación del ID del operatingPeriod que gobierna el siguiente bloque.

#### 10 - metadata:

Elemento utilizado para introducir metadatos, con los mismos subelementos que el propio esquema de metadatos common/metadata, como le creador del esquema, la fecha o el formato. Es útil pero no necesario puesto que, entre otras cosas, estos datos también se pueden definir de forma global para el archivo completo.

### 6.1.4 Elementos de Common/metadata

Este esquema tiene pocos elementos, pero ciertos de ellos son de alta importancia, como por ejemplo, los campos reservados a la autoría del archivo, o del versionado del mismo. Todos sus

elementos se rellenan con el valor del elemento (como se ha explicado antes, el valor fuera de los atributos, del modo <elemento>**VALOR**</elemento>). La mayoría de estos elementos tienen una definición precisa del algunos grupos, pero el más extendido es el Dublin Core Metadata

#### 1 - any:

En este elemento se pueden incluir elementos de cualquier tipo, que no se acoja a la descripción o a las necesidades de los otros elementos.

#### 2 - dc:contributor:

Contiene toda la información relativa a entidades responsables para contribuir al recurso.

#### 3 - dc:coverage:

Define la cobertura de la aplicación del archivo.

#### 4 - dc:creator:

Atesora la información relativa al creador o los creadores del documento. Este elemento lleva un atributo xml:lang para definir el lenguaje en que se dan dichos datos.

#### 5 - dc:date:

Refleja la fecha de creación o última modificación del archivo. También conlleva el atributo xml:lang.

#### 6 - dc:description:

Contiene una descripción del archivo en general.

#### 7 - dc:format:

Aquí se define el formato en que está hecho el archivo.

#### 8 - dc:identifier:

Este elemento guarda toda la información referente al identificador unívoco del archivo.

#### 9 - dc:language:

El elemento da información sobre el lenguaje utilizado globalmente en las definiciones de los elementos del archivo. Es importante no confundir este elemento con la definición del lenguaje XML que se hace al comienzo de cada archivo.

#### 10 - dc:publisher:

Elemento con referencia a la empresa que publica el código.

#### 11 - dc:relation:

Define la relación de fuentes utilizadas.

### 12 - dc:rights:

Información relativa a los derechos mantenidos (de autor, distribución, etcétera).

### 13 - dc:source:

La fuente de la que deriva la información relativa al archivo.

### 14 - dc:subject:

En este elemento se define la intención y el tema del archivo

### 15 - dc:title:

Posible título que se le puede dar al código.

### 16 - dc:type:

Aquí, se dará la información de la naturaleza del recurso.

### 17 - organizationalUnits:

Este es un elemento contenedor sin atributos. A diferencia de los elementos definidos anteriormente dentro de este esquema, este elemento no guarda metadatos (datos que definen información relativa a los datos del código) sino datos comunes que se establecen en este esquema por falta de conveniencia en otros esquemas. Es decir, da información global sobre la red, no sobre el código. Se define por sus subelementos, definidos a continuación. Además, se debe indicar que estos elementos son predefiniciones, es decir, son definiciones de elementos que más tarde se podrán referenciar por medio de su ID cuando se necesiten dentro de los diferentes esquemas.

Todos los elementos de el elemento <organizationalUnits> tendrán estos atributos para definirse: *id*, *code*, *name*, *description*, *xml:lang*.

### 17.1. - infrastructureManager:

Elemento que informa sobre el nombre de la empresa gestora de la infraestructura. Sus atributos son

### 17.2. - vehicleManufacturer:

Indicaciones sobre el fabricante de los trenes.

### 17.3. - vehicleOperator:

Detalla la empresa que opera el tren.

### 17.4. - customer:

Identifica la autoridad o compañía que he solicitado el servicio de transporte. Obviamente, esto no se refiere al usuario del tren.

### 17.5. - railwayUndertaking:

Compañía responsable por el servicio prestado.

### 17.6. - operationalUndertaking:

Compañía responsable por el plan de operaciones de la red.

### 17.7. - concessionaire:

Compañía concesionaria si las hay.

### 17.8. - contractor:

Empresa contratista si las hay.

# 7 Interpretación de RailML®

## 7.1 Generalidades

RailML® es un lenguaje XML. Por ello, en el mundo de la informática puede haber infinidad de formas de interpretación del mismo, comenzando desde la lectura del propio código, pasando por programas de estructuración simples que ayudan a visualizar el código según la parte que interese, hasta programas de interpretación diseñados al efecto (como es el caso de RailVIVID®, del que se hablará a continuación).

Como se ha dicho en la introducción a este texto, es perfectamente posible, e incluso sencillo, crear programas que interactúen con el código que se ha generado de varias maneras: es posible crear una herramienta que automatice la creación del código a partir de unas reglas simples y un listado de datos referentes a la vía, al igual que es posible definir y personalizar software de lectura de dichos datos.

Propongamos tres escenarios diferentes. El primero es la lectura del código por un operario de estación, en que dicho operario necesitará datos de horarios, billetes, precios, incluso hasta estadísticas de uso. Otra posibilidad es que el código vaya a ser interpretado por el maquinista, con lo que necesitará la información de las vías, señales, enclavamientos, etcétera. Sin embargo, si este código va a ser usado en un tercer escenario por operarios de los enclavamientos (o por sistemas informáticos de los mismos), se necesitará una visión global de vías, horarios, restricciones de uso y especialmente información sobre bloqueos (que se incluye en RailML® más adelante). En un principio podría parecer que necesitan tres tipos de información diferente pero la posibilidad de adaptación que RailML® ofrece hace que eso no sea correcto, sino que necesite un solo esquema de información pero tres vías diferentes de interpretación. Para ello, se pueden crear programas que faciliten el acceso a una u otra información de manera rápida según las necesidades del usuario.

## 7.2 RailVIVID®

Obviamente, no todo usuario que tenga algún tipo de relación con el mundo ferroviario tiene conocimientos de programación o informática y, más en específico, de lenguajes estructurados de datos como XML. Así, se hace necesario que exista software de interpretación, que ayuden al usuario a convertir el código y traducirlo a un lenguaje comprensible por todos los usuarios, bien sea de manera literal, gráfica, o mixta.

Una vez introducido el concepto de programa de interpretación, hay uno que merece una mención especial, entre otras cosas, por ser el programa oficial de la UIC, y por ser el que se ha usado para el desarrollo del proyecto.

RailVIVID® es un programa que transforma el código en una interpretación gráfica, en planta o alzado, o en una tabla de elementos. Es una herramienta simple con una interfaz clara y fácil de utilizar, con ciertas carencias específicas y varias ventajas. La mayor de sus ventajas es su simplicidad y facilidad de uso que hace que prácticamente cualquier sistema informático pueda

ejecutarlo y que, todos y cada uno de los usuarios que analicen un archivo RailML® con él, puedan conseguir los valores gráficos que necesiten en cuestión de minutos. Es, gracias a dicha simplicidad, una herramienta rápida y sin muchos problemas de cálculo (aunque sí se ha notado alguna incidencia que no está completamente depurada todavía). Además, puesto que es una herramienta flotante y por ello no necesita instalación, su ejecución es rápida y no supone ningún problema para prácticamente ningún sistema (de todos es conocido que según qué software, se necesitan capacidades de cálculo importantes, pero eso no es ningún impedimento aquí).

Una de los puntos más a echar en falta de este programa durante la creación del código es que no da información sobre la estructura del código en sí. Es decir, es capaz de evaluar el código de RailML® una vez que es capaz de compilarlo pero, en caso de que no sea capaz, no arroja ninguna información sobre cuáles son los posibles problemas que están imposibilitando esa compilación. Esto puede llegar a ser incluso engorroso a la hora de generar el código, dado que hay (al menos) un error que no permite compilar, y no se obtiene información de dónde está. Sin embargo, gracias a que la estructura es de XML puro, se puede hacer uso de uno de los muchos comprobadores de estructura XML para conocer dónde están los fallos que no permiten su lectura. Una vez este paso está resuelto, RailVIVID® da información precisa de errores que no concuerdan con los esquemas de definición de RailML®.

Dado que hoy en día se utilizan diferentes formatos para compartir los datos que se reflejan en el código RailML®, RailVIVID® da la posibilidad de exportar la información que se haya creado en varios formatos (como son PDF, imagen PNG, CSV o Excel, entre otros). Esto es muy útil puesto que da la posibilidad de adaptar la nueva tecnología a las que son utilizadas hoy en día, haciendo la transición entre las dos más fácil y lisa.

A continuación, se va a hacer una breve explicación de los puntos de los que se compone la herramienta y su uso.

### 7.2.1 Validador

Este es un elemento principal del software. En él se comprueba que el código generado cumple con las normas que los esquemas imponen. Este primer paso es necesario para comprobar que los datos que se han plasmado serán reconocibles por el software. Dado que prácticamente todos los elementos del archivo van a tener relaciones con otros elementos, es importante ver que ciertas definiciones (sobre todo los ID) están correctamente hechas. Este validador comprueba, por ejemplo, que los atributos de tipo referencia hace mención a un ID que se puede encontrar y está definido en el elemento que debe estar.

NOTA: la validación no significa certificación. Solo implica un seguro de que la ejecución del texto va a ser correcta, pero no que se aplican los requisitos de UIC para el uso de RailML.

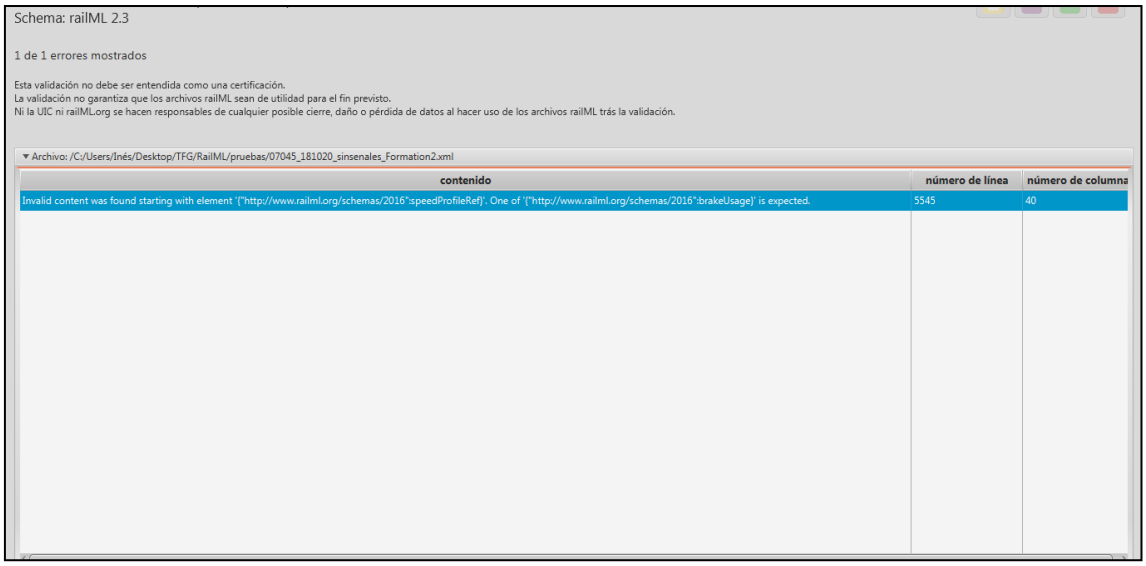


Fig. 7 Ejemplo de validación negativa

Este es un ejemplo de la información que se puede ver en el validador cuando se encuentran fallos, en la captura superior, en que se muestra la cantidad de errores y una fila por cada error con la descripción del mismo, y la fila y columna en que este error se ha encontrado. Por otro lado, en la captura inferior, se ve la información detallada que se muestra cuando no se encuentran errores (dándose los metadatos que están detallados en el archivo, y los subesquemas que se han modelado en el archivo, estén o no completos).

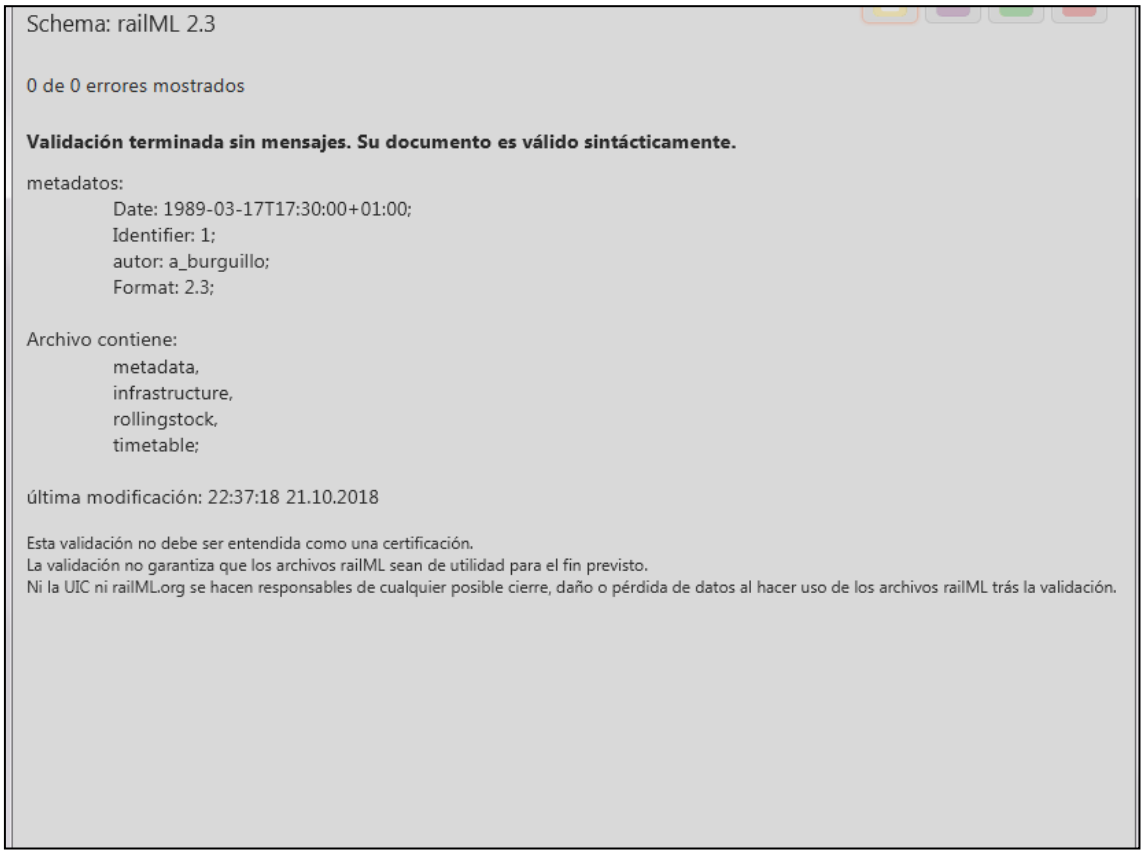


Fig. 8 Ejemplo de validación positiva.

### 7.2.2 Infraestructura

Esta es la parte del software donde se pueden comprobar todos los datos referentes a la infraestructura. Este aspecto del programa se basa en tres apartados, que son la vista topográfica, la vista de mapa y la vista de tabla.

La primera de estas da información sobre la topología de la pista en lo que se refiere a pendientes en cada uno de los sentidos de circulación, velocidades máximas en cada dirección, protección del tren en cada sentido, electrificación, altura, y radios.

A continuación se detallan dos ejemplos extraídos del caso práctico en los que se ve claramente la diferencia entre definir diferentes elementos para cada sentido de circulación, o definir la misma información en ambas. En la primera imagen se ve la vía con definición de velocidad en ambos sentidos (dos primeras líneas), gradiente general y radio general (afectan a ambos sentidos). En ella también se ve un punto negro en la representación de la vía. Esto es un ocp, de modo que los ocp que se definan en el código, aparecerán aquí. Esto abre la posibilidad a añadir datos que, en un principio, no están disponibles en la vista topográfica.

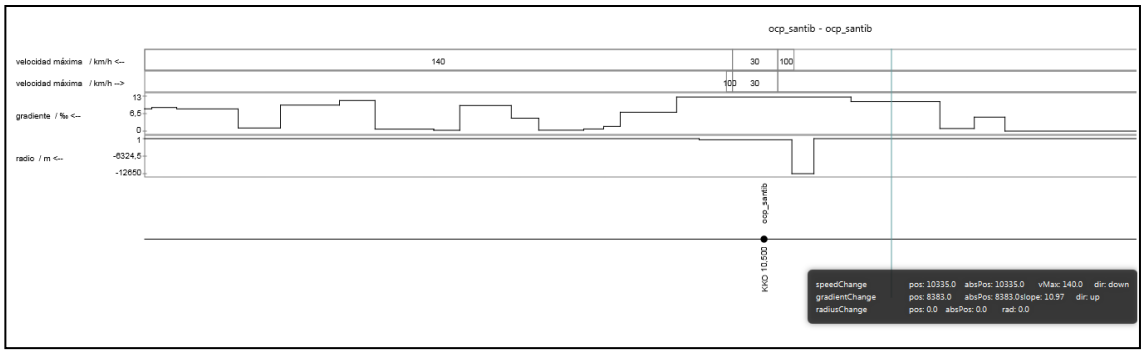


Fig. 9 Ejemplo de vista topográfica

La segunda imagen, sin embargo, muestra una posible definición de vía en que el radio y la pendiente cambian según el sentido de circulación. Esto, obviamente, depende de la forma en que se modele la vía puesto que, como se vio anteriormente, se puede modelar una pareja de vías paralelas como una sola vía, aunque eso resta bastantes posibilidades a la hora de completar la información sobre ambas. Por esto, si se elige el modelado por medio de un modelo para cada vía, esta posibilidad y la captura inferior carecen de sentido. Aún así, se ha optado por modelar tres vías con esta diferencia de sentidos, con el mero objetivo de ilustrar dicha posibilidad y su forma de proceder.

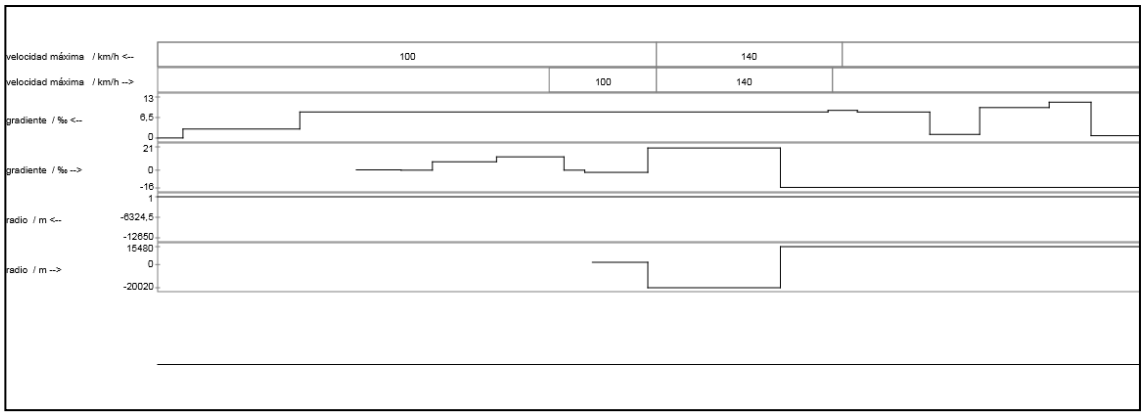


Fig. 10 Ejemplo de vista topográfica, con valores diferentes para cada sentido

Como segunda posibilidad, la vista de mapa ofrece una vista de planta de las vías, con todos sus elementos (puentes, señales, pasos a nivel, etcétera) representados todos ellos por símbolos específicos. Una vez se ponga el puntero del ratón sobre estos elementos, se mostrará un cuadro de texto con toda la información que se haya modelado anteriormente.

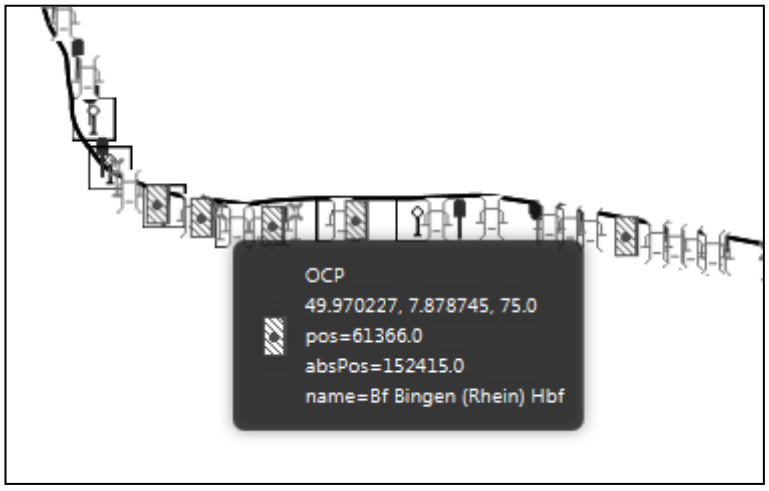


Fig. 11 Ejemplo de vista de mapa para una infraestructura ( Wolf, 2016) [20]

La última visualización es una tabla de elementos que muestra cada uno de los valores asociados a dichos elementos.

El siguiente paso a definir es cómo se elige qué parte de la estructura se quiere mostrar. Para ello, como se ve en la siguiente captura, se puede elegir seleccionar una o más vías (si se quiere seleccionar más de una vía, estas deben estar unidas de alguna manera, por ejemplo la vía tr10021 está conectada con la tr10031, y así sucesivamente); también se pueden seleccionar varios ocp que darán la información de las vías que están entre dos o más ocp. Es importante notar que esta forma solo servirá si los ocp se encuentran en el mismo elemento track. Por último se puede elegir una visión completa de la red por medio de la elección por red ("by net") en que se podrá seleccionar únicamente la vista de mapa.

[20] Imagen obtenida de uno de los códigos de ejemplo proporcionados por www.railml.org

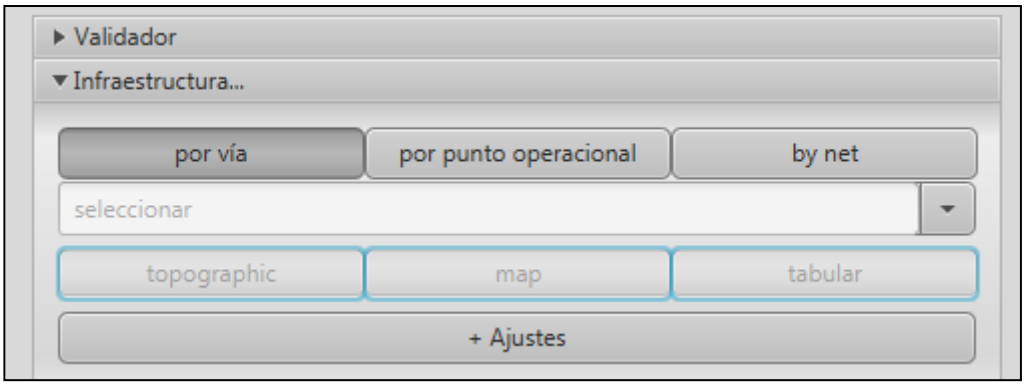


Fig. 12 Selección de infraestructura por vía

Si la forma de elegir es por vías, la forma de proceder es eligiendo la primera vía, y de ahí, elegir una a una las siguientes. Cada vez que se elija una, se desplegará un campo más para elegir la siguiente.

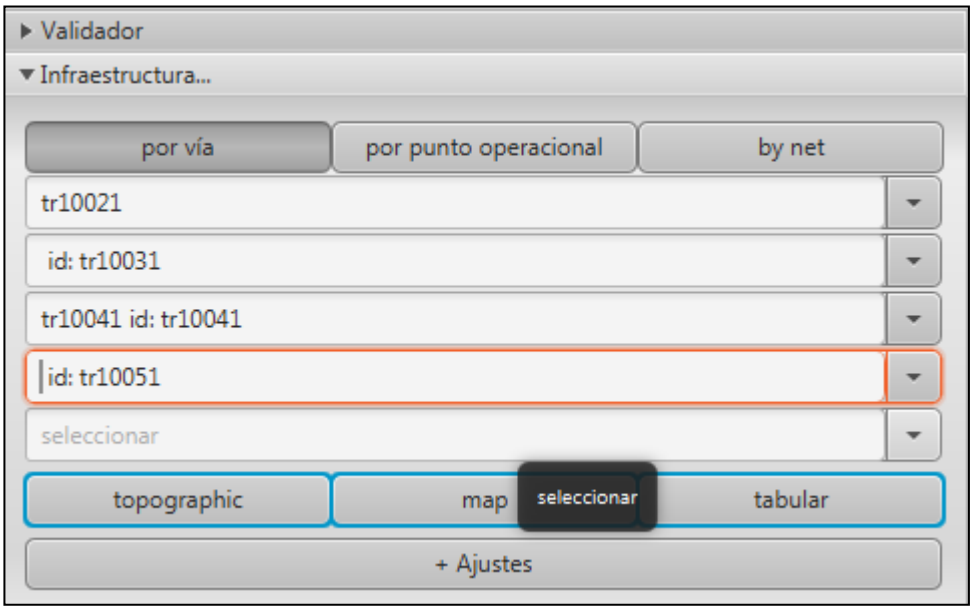


Fig. 13 Selección de infraestructura por vías

### 7.2.3 Tratamiento Horario

En este campo, se mostrará toda la información relativa al esquema timetable. En él, se podrá elegir qué datos quieren visualizarse según si quiere verse la información relativa a un tren o a un ocp. Además, se dará la posibilidad de mostrar los horarios en una tabla o en un modo gráfico.

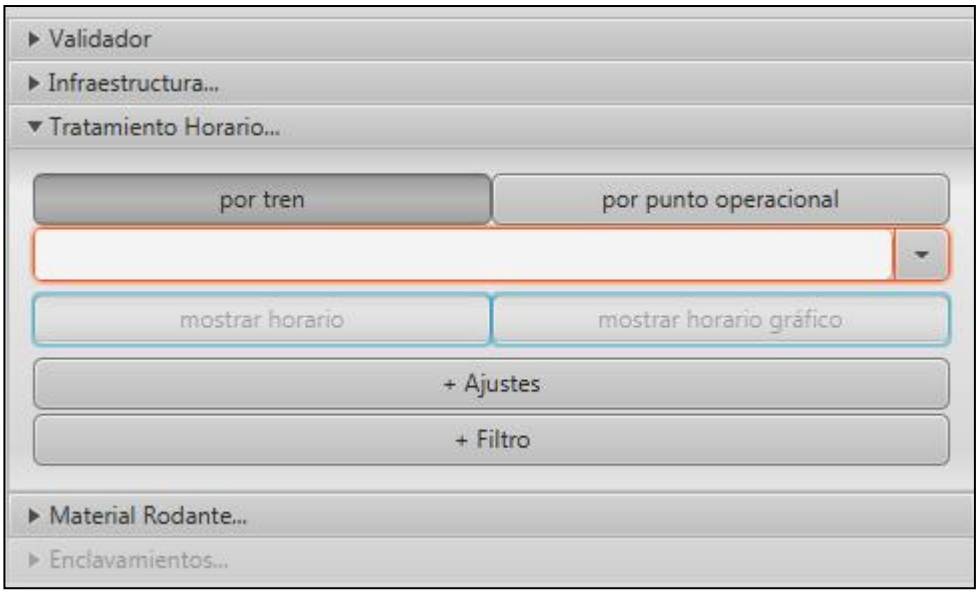


Fig. 14: Selección de horarios, por tren o por punto operacional.

Si se elige el primero, se dará una tabla como la que se muestra a continuación en que se darán horas de parada en cada estación, tipo de plazas especiales disponibles (como son las plazas para sillas de ruedas, y para bicicletas), y el período operativo definido (en la parte de abajo, mostrando el nombre que se le haya dado a dicho período).

| tren | tr28020 | tr39699 | |
|---|---|---|---|
| de | | | |
| ocp_santib | 15:15 | 15:35 | |
| ocp_bifPajr | 15:36 | 00:30 | |
| ocp_roblaAV | 16:15 | 16:15 | |
| ocp_campAV | 18:05 | 18:05 | |
| ocp_polLen | 18:59 | 20:15 | |
| ocp_ujo | 19:46 | 19:46 | |
| a | | | |
| 1 | Lu a sá 18 | | |

Fig. 15: Muestra de información horaria en tabla.

La otra posibilidad da un gráfico en que se muestran líneas representando las rutas de los ferrocarriles con sus paradas en cada respectivo ocp definido.

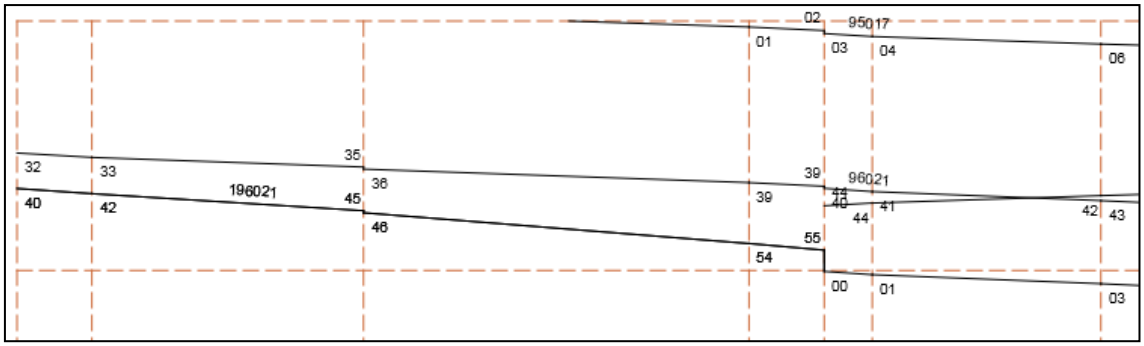


Fig. 16: muestra de información horaria en forma gráfica.[21] (Norway Railway Network, 2017)

Es obvio que, según se definan más horarios y más períodos de validez de los mismos, estas tablas se harán progresivamente más grandes y contendrán más información.

En lo respectivo a la selección de configuración de la información que se desea, se puede elegir entre dos maneras, la selección de horarios en función del tren (dando la información relativa al tren y a todos los ocp en que para dicho tren), o en función de los ocp (que nos proporcionará información sobre los tiempos de salida de cada tren desde los ocp que se hayan seleccionado).

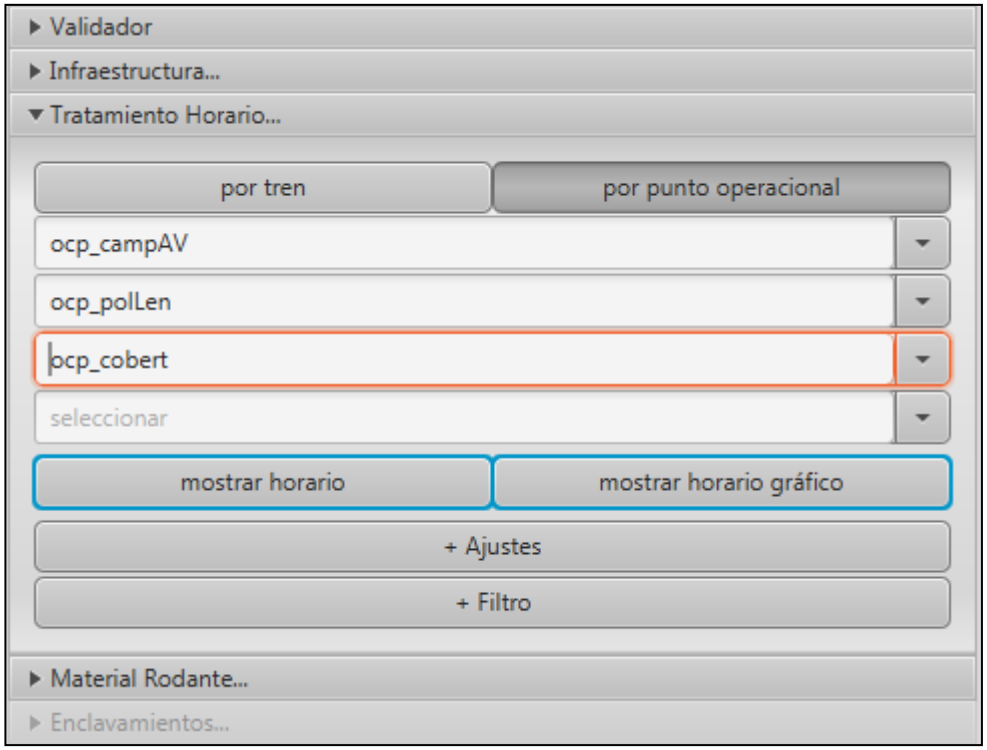


Fig. 17: Selección de tratamiento horario según ocp.

En el apartado, se pueden hacer, además varias selecciones (en el campo "+ Ajustes") como el cambio de ejes en que se muestra el horario gráfico, el color de fondo, o las escalas. Una de las posibilidades más importantes es la posibilidad de diferenciar los trenes asignándoles colores.

[21] Imagen obtenida de un código de muestra de railml.org

### 7.2.4 Material rodante

Todos los datos que se definan en el subesquema rollingstock se podrán consultar y ver en este apartado. Aquí, se hará la selección de los datos que se quieren ver por medio de los elementos <vehicles>, de modo que se verán las características relativas al elemento <vehicle> que se seleccione.

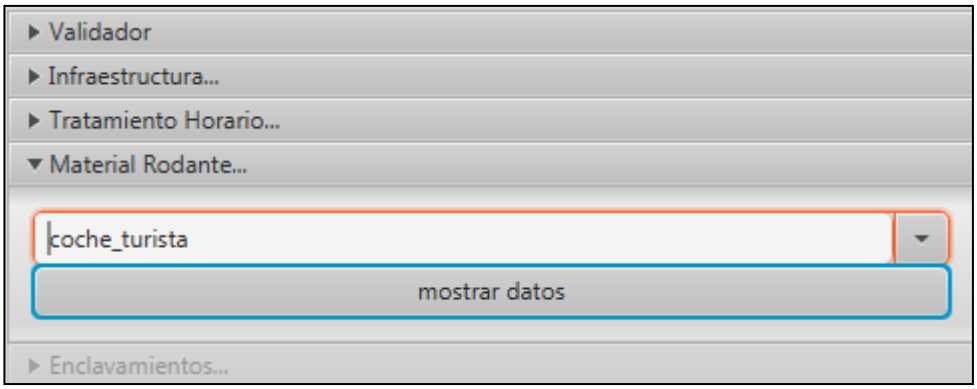


Fig. 18: Selección de material rodante a mostrar.

Habiendo seleccionado uno de los elementos, se podrá ver toda la información referente al mismo, como sus datos técnicos (longitud, o peso), diagramas de caracterización (tracción, frenado, etcétera), o los datos organizativos, como plazas disponibles o el fabricante.

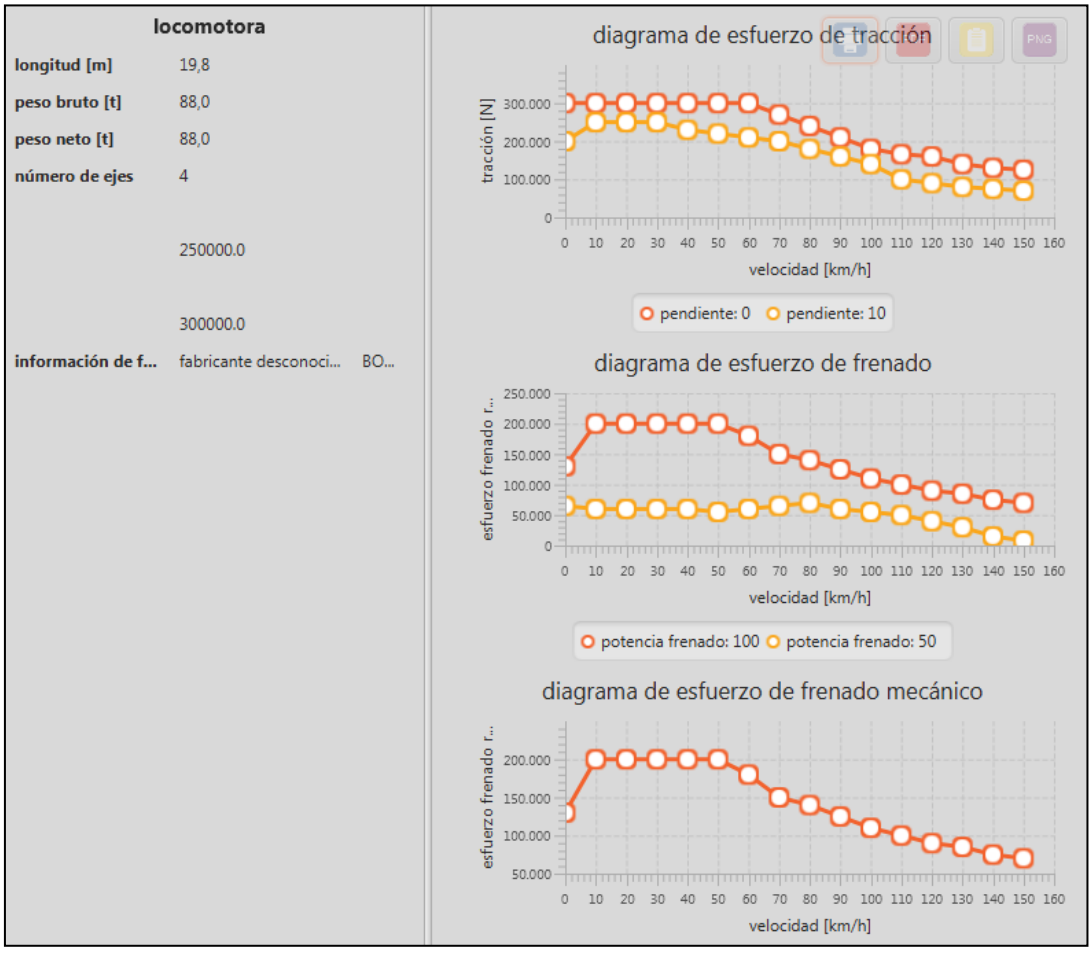


Fig. 19: Muestra de valores del material rodante. Ejemplo de tablas de tracción y frenado.

| coche_turista | | |
|---|---|---|
| **longitud [m]** | 20,0 | |
| **peso bruto [t]** | 90,0 | |
| **peso neto [t]** | 28,8 | |
| **número de ejes** | 4 | |
| **información de fabricante** | fabricante desconocido num.1 | BOETSII |
| **uso de pasajeros** | **categoría** | **recuento** |
| | wheelchair | 6 |
| | class2 | 35 |
| | other | 2 |

Fig. 20 Presentación de datos físicos de un vagón.

### 7.2.5 Enclavamientos

Por último, se dará la información referente a enclavamientos y sus usos y prioridades, así como sus casos de uso. Este campo dará la información guardada en el subesquema interlocking. Sin embargo, este subesquema está bajo desarrollo. Por tanto, esta funcionalidad no está disponible en el programa todavía, aunque se presente en el menú principal.

# 8 Resultados

## 8.1 Ventajas y carencias

Es trivial darse cuenta de que, según la versión que se utilice, las ventajas o las carencias que se encuentren en el código pueden ser completamente diferentes. Por eso, se debe aclarar que las conclusiones que se puedan leer en este documento son única y exclusivamente aplicables a la versión que es objeto de estudio en este texto (versión 2.3).

### Ventajas

Son muchas las ventajas que puede aportar la digitalización de la información de las infraestructuras en general, y RailML® en particular. Lógicamente, en los tiempos en que vivimos, el avance de las tecnologías se produce a una velocidad realmente impresionante, esto significa que cada día se inventan más maneras de trabajar con los datos. De este modo, la organización digital de datos abre la puerta a una infinidad de posibilidades, como son la facilidad de transmisión de datos, el control de corrupción de archivos, la posibilidad de implementar dobles comprobaciones con archivos duplicados, el cálculo masivo de datos con tecnologías como Big Data, o la inclusión de sistemas de procesado de datos para obtener cualquier tipo de dato referente al apartado comercial desde los archivos de RailML® (por ejemplo, los horarios).

Por otro lado, RailML® tiene ventajas inherentes a la definición del lenguaje y es que, independientemente del sistema o formato que cada país, empresa, u organización utilice para leer datos referentes al control ferroviario, el código se genera de la misma manera. Esto significa que el código será el mismo sin importar quién lo ha hecho o quién lo va a utilizar.

Así, podría surgir la duda de que, si una organización cualquiera utiliza un sistema o un formato diferente al elegido para definir RailML®, esto podría afectar a la posible lectura del archivo. Sin embargo, dado que la lectura del archivo se produce a través de un programa de interpretación, sería sencillísimo programar el software de interpretación de modo que haga el cambio de unidades, formatos, o sistemas mientras traduce el código a una interfaz gráfica mostrable. Esto es así porque, gracias a los esquemas de definición, se sabe qué formato o unidades lleva cada atributo o elemento, y se da la posibilidad de modificar el dato a placer una vez sea este extraído del código.

### Carencias

Tal vez, la carencia más importante es la imposibilidad de modelado de bloqueos, preferencias y formas de configurar el tráfico. Sin embargo, esta carencia se solventará en próximas versiones (en la versión 3.0), y podrá generar archivos con este tipo de información.

Otro de los grandes inconvenientes de este sistema es el cambio que conllevaría, con el tiempo que se requiere para traducir los datos existentes a este formato, y la inversión inicial que se debería hacer para tener una cobertura mínima para que estos archivos tengan una verdadera utilidad.

Además, si bien este sistema ayuda a conseguir una homogeneidad de datos entre los diferentes usuarios y desarrolladores de documentación, también da la posibilidad de generar un archivo de documentación que tenga solo la información que al desarrollador o al consumidor le sean interesantes en ese momento, de manera que se pueda dar el caso que la documentación generada por un fabricante tenga ciertos aspectos, y la generada por otro fabricante no contenga esos datos que, de partida, no son obligatorios para la consecución del código. Aunque esto, claro está, no es una desventaja si se compara este sistema con el actual en que ningún documento es capaz de exigir ciertos tipos de información si no son exigidos por ley (exactamente igual que sucede con RailML® y su sistema)

Sin embargo, la mayoría de las carencias vienen de la mano del software que se utiliza para la conversión del código a formato gráfico, puesto que no hay ninguna posibilidad de configuración de los datos que se necesitan ver o qué datos no. De la misma manera, sería excelente tener algunas características más, como perfiles de usuario que guarden las características predefinidas en función de qué necesita dicho usuario. Aun así, este problema es una carencia muy sencilla de subsanar, dado que una vez que se tiene el código RailML®, cada usuario (o compañía) puede desarrollar una herramienta que tenga todas las características que sean necesarias para la interpretación del código (sin olvidar, por supuesto, el consiguiente gasto económico que ellos conllevaría). Este punto es muy a tener en cuenta, puesto que la configuración de RailVIVID® no permite ver, por citar un ejemplo, los elementos de las vías (tales como puentes o señales) en la vista de perfil, lo que sería de extraordinaria utilidad.

A modo personal, una de las carencias más notadas es cierta falta de información presentada por el software de interpretación que, sin embargo, existe en el código; como son los datos relacionados con la formación del tren al completo, así como algunas características de puntos específicos de la infraestructura (por ejemplo la altura de los andenes, o características de las señales)

### Problemas a la hora de modelar

Todo principio es complicado y, como no podía ser menos, comenzar a modelar con RailML® tiene aspectos que dificultan la tarea. El mayor de ellos es la dificultad de lectura y comprensión de los esquemas de definición. Con cualquier lenguaje estructurado, se tienen que respetar ciertas reglas y, estas son más fáciles de identificar en unos casos que en otros. Esto quiere decir que hay ciertos errores que, por pequeños que sean, tienen una influencia grande dado que no permite que el texto sea identificado como un archivo XML. Por eso se ha creído que puede ser útil crear un directorio de errores comunes que consultar a la hora de modelar. A continuación se detallan algunos de las reglas que son más comunmente olvidadas:

Los atributos deben estar separados entre sí por espacios, dado que si entre dos atributos no hay espacio separándolos, el programa compilador intentará resolver un solo atributo que contendrá dos elementos "=". Asimismo, entre el atributo y su valor, no debe existir ningún espacio, por el mismo motivo.

Los atributos y los valores determinados en listas se deben deltrear exactamente igual que en los esquemas, puesto que si no se hace así, no se podrá emparejar con la estructura que el compilador crea a la hora de leer el código.

Si un elemento no va a tener subelementos contenidos, se debe (no es obligatorio en todos los casos, pero siempre es recomendable) cerrar el elemento en los mismos corchetes que abren el elemento, es decir, se debe usar *<elemento atributo='valor'/>* en lugar de *<elemento atributo='valor'></elemento>.*

Por último, se debe indicar que, aunque en muchas ocasiones los atributos parezcan autoexplicativos, en algunas ocasiones hay matices que hacen que un atributo describa una propiedad diferente a la que el nombre del atributo sugiere. Por tanto, se recomienda enormemente prestar atención a la definición de cada uno de estos elementos a conciencia antes de comenzar a dar los valores que definan los elementos.

Los atributos y valores deben ser concordantes con la versión utilizada. Esto parece obvio pero, a la hora de modelar es muy posible consultar documentación que no tenga esta diferenciación en cuenta.

## 8.2 Posibilidad de uso e implantación

RailML® es un lenguaje con mucho potencial. Tiene cubiertas muchas partes de la organización ferroviaria que pueden ser utilizadas de muchas maneras diferentes, lo que hace que las posibilidades de implantación sean enormes. Obviamente, desde un punto de vista técnico, este lenguaje debe ir acompañado de una infraestructura importante detrás, como son bases de datos que contengan la información, programas de interpretación que sean capaces de extraer todo el potencial que el lenguaje ofrece. Para eso, como se ha comentado anteriormente, hace falta una inversión inicial bastante importante, que debe llevar aparejado un estudio de posibilidades para convenir si dicha inversión compensa o no.

También es crítico tener en cuenta para qué sirve este lenguaje. Este es, sobre todo, un lenguaje que facilita la transmisión de datos, puesto que si solo se quiere almacenar datos, aunque sí se puede usar RailML® para el almacenaje de datos, hay vías más rentables y de fácil utilización, como son las bases de datos relacionales. También es cierto, que trabajar con bases de datos, su ordenación, creación, modificación y lectura necesitan de mayor grado de especialización que los archivos XML (recordemos que este lenguaje está creado con una máxima: la simplicidad).

RailML® añade al diseño de los datos de control ferroviario una directriz de uniformidad bastante clara, puesto que, se quiera o no, se necesita dar ciertos valores para poder completar los archivos y que estos sean reconocibles; así pues, cualquier elemento vía tendrá un identificador ID único, lo que evita incongruencias dentro de la misma vía. Además, los valores numéricos se deben dar siempre en las mismas formas, magnitudes y formatos, lo que facilita mucho la comprensión de los mismos.

Por todo esto, es trivial que RailML® aporta grandes posibilidades de uso y, sobre todo, un altísimo potencial de desarrollo, que depende principalmente de desarrollos paralelos de otras aplicaciones, y sistemas (como se ha dicho antes, sobre todo, del programa que pueda interpretar el código). Así, se define RailML® como una herramienta adaptable a las necesidades de la red española y a sus métodos.

## 8.3 Impacto en el panorama actual

RailMl® supone principalmente la digitalización de mucho material que actualmente existe, esto es, se necesita crear un proceso de migración que desarrolle todo el trasvase de conocimientos de un medio físico actual a un medio digital, por lo que se requiere gran cantidad de tiempo para que se pueda llevar a cabo dicho cambio. Asimismo, también es necesaria una implantación de sistemas adecuados al intercambio de datos por medio de archivos de texto plano como es RailML®. El conjunto de estos dos puntos hace que una aproximación a este tipo de archivos sea complicada y necesite un tiempo de coexistencia de ambos tipos que, en la mayoría de casos, será grande. Por último, también es destacable, de nuevo, la inversión inicial que ello conllevaría. El principal retorno de dicha inversión se derivaría de la rapidez y facilidad para cambiar datos de organización y control, lo que puede abrir mercado a la utilización de las redes por diferentes operadores.

Además, dada la alta cantidad de material que habría que digitalizar, así como por las altas posibilidades de mejora que ofrece, una inclusión de RailML® supondría una inyección importante de puestos de trabajo cualificado.

# 9 Análisis de proyecto

## 9.1 Estudio temporal

El proyecto se ha distribuido en un total de trescientas treinta horas. En ellas, hay principalmente una fase de investigación y otra de desarrollo de la idea. Durante la primera fase, se establecieron una fase principal de acopio de información global sobre las herramientas , y otra fase de búsqueda de información sobre los requisitos de definición mínimos que se necesitan para modelar correctamente una vía, así como de los comúnmente utilizados. Esto requirió una suma de 85 horas, que se repartieron a lo largo del trabajo puesto que en fases últimas de la parte práctica, se necesitó volver a investigar sobre las mejores maneras de modelar ciertos elementos en RailML®. La segunda parte, fue la más larga con diferencia, puesto que aun con un nivel de dificultad aceptable, el trabajo de acopio de información y su transcripción a código es larga. Es por ello que se optó por invertir un poco de este tiempo en crear una herramienta basada en VBA (Visual Basic for Applications) que redujo drásticamente la cantidad de tiempo invertido por cada elemento que se representa en las vías. Aún así, la ejecución práctica tomó un total de 175 horas de trabajo, mientras que la elaboración de la memoria de 75 horas. El reparto de las mismas se puede apreciar en el gráfico que se muestra a continuación.

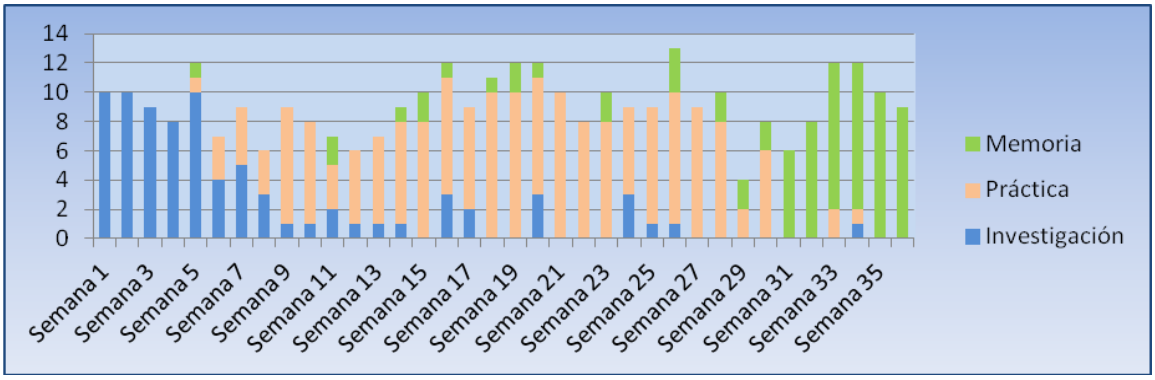


Tabla 6: Distribución de horas de dedicación

## 9.2 Estudio económico

El proyecto se ha basado en el uso de un software de licencia gratuita para el desarrollo del código. Aunque el uso de este programa es gratuito, la certificación tiene un coste, aunque este coste no tendrá influencia en este proyecto puesto que no se ha llevado a cabo dicha certificación. Además de esto, se tendrá en cuenta el tiempo invertido en la elaboración del proyecto, de la investigación y búsqueda de información pertinente para la generación de un modelo fiable y utilizable de gestión y control de una infraestructura ferroviaria. Este tiempo asciende a 330 horas que, estableciendo un precio por hora de diez euros por cada hora trabajada, ascienden a un total de 3300 euros. Asimismo, se tendrá en cuenta la utilización del material necesitado para la realización del trabajo, donde entran básicamente el ordenador

utilizado para el mismo con su amortización durante los nueve meses que se han destinado al proyecto (un total de 1000 euros que suponen una amortización de 250 euros) y material de oficina por un importe de 80 euros.

Esto hace una cantidad total de coste de proyecto de 3630 euros sin IVA, que se traducen en 4392 euros.

| **Coste de Ejecución** | Coste unitario | Cantidad | Subtotal |
|---|---|---|---|
| Horas de trabajo | 10,00 € | 330 | 3.300,00 € |
| Amortización mensual del ordenador | 27,77 € | 9 | 249,93 € |
| Material de oficina | 80,00 € | 1 | 80,00 € |
| Total | | | 3.629,93 € |
| Total con IVA | | | 4.392,22 € |

## 9.3 Impacto medioambiental

En este caso, no tiene demasiado sentido un estudio medioambiental completo, puesto que el proyecto se basa únicamente en desarrollo digital. Sin embargo es mencionable que, a día de hoy, todo el traspaso de información se efectúa por papel, con el gasto de papel y deforestación que ello conlleva. La digitalización es la base de este trabajo y un traspaso de la información a una estructura digital provocaría que en todos los posibles cambios futuros, así como las nuevas inclusiones de datos no gasten ningún tipo de consumible salvo ordenadores, que obviamente ya están presentes en todas las redes. Esto implicaría una reducción de la cantidad de papel utilizado que, a día de hoy, es un bien que debe tratarse con moderación. Lógicamente, habría que tener en cuenta la energía consumida por los ordenadores que se utilizaran a tal efecto, así como la posible necesidad de reciclaje de los mismos sin embargo, puesto que estos aparatos están ya en cualquiera de las compañías que los necesitarían, el incremento de esta energía sería nimio.

# 10 Conclusiones

## 10.1 Conclusiones del Trabajo

Tomando como primer referente de las conclusiones un análisis del potencial de RailML® en aplicaciones de control ferroviario; un punto destacable es que, al ser un lenguaje informático, aporta una gran flexibilidad de adaptación y un tremendo potencial. Se ha visto como, en un archivo de menos de medio megabyte de espacio en memoria, se han modelado ciento veinte mil kilómetros de vías teniendo, en ocasiones, hasta cuatro vías en paralelo. Esto hace pensar que este método de modelado es muy ligero, además de útil y que puede dar pie a facilitar mucho el movimiento de información entre fuentes diversas. Esto abre las puertas a muchas vías para compartir dicha información. En el estado actual de RailML®, hay una carencia clara, que es el modelado de prioridades y bloqueos, en el esquema interlocking pero, una vez este esquema esté disponible, se podrá sustituir por RailML® la práctica totalidad de los métodos de divulgación e información que se usan en este momento.

Como segundo punto, no sin alguna dificultad (sobre todo en los inicios), se ha conseguido traducir a código RailML® toda la información que se pretendía, en la red ferroviaria elegida. Si bien, estos resultados no se muestran al cien por cien de la manera esperada en el software utilizado para la representación gráfica de los mismos, esto es un aspecto no inherente al código, sino al programa usado para su representación. Es decir, para que este modelo cumpliese las expectativas al completo, no se necesita ningún cambio latente en los esquemas aparte de la inclusión del ya mencionado esquema interlocking. Dada la complejidad del código en una primera aproximación, la amplitud y extensión del mismo, y las necesidades de conocimientos básicos de programación en XML, el ejemplo práctico se ha considerado un éxito contando, por supuesto, las limitaciones propias del lenguaje. Por los mismos motivos, se considera que la dificultad del proyecto es alta, sobre todo como primera aproximación al mundo de la gestión ferroviaria como tal.

Asimismo, dado que la validación que se propuso del código con el software RailVIVID® se ha conseguido satisfactoriamente, se considera ese objetivo como conseguido.

Es importante destacar que es necesaria una gran cantidad de datos previos a la elaboración del código, puesto que, debido a la gran extensión y posibilidades de definición de esta herramienta, se debe dar la mayor información posible y esta ha de ser precisa. Esto hace que haya una gran cantidad de trabajo previo a la codificación en sí misma. Esta es una de las principales dificultades que se ha encontrado en la ejecución del trabajo y, de hecho, una de las partes no se ha podido cumplir exactamente (la referente a la geolocalización) puesto que no se ha dispuesto de ciertos datos y se han tenido que hacer estimaciones o interpolaciones. También, se han dado por buenas ciertas aproximaciones como pensar que solo un tipo de trenes circulará por las vías puesto que el objetivo de este trabajo no es el modelado exacto de una línea y sus intervinientes, sino exponer las posibilidades de este sistema.

## 10.2 Conclusiones académicas con RailML®

Este estudio ha sido un primer paso en la generación de una orientación acerca de cómo se deben modelar archivos RailML que, en pasos siguientes, podrá generar archivos más complejos, de más extensión y con mayor cantidad de red cubierta. También se ha conseguido demostrar que las capacidades y aptitudes informáticas necesarias para modelar este archivo son lo suficientemente sencillas para conseguir una base rápidamente sobre la que construir un modelo lo suficientemente amplio. En este caso, la inmersión completa en el lenguaje, hasta poder comenzar a escribir código se consiguió en apenas 15 horas de trabajo contando únicamente la formación para el manejo del lenguaje.

Se ha visto, durante la ejecución del caso práctico, que hay errores aceptables en el modelado (o, mejor dicho, ausencias aceptables), como es la falta en el ejemplo del elemento <passengerFlowSpeed> que da un valor orientativo de la velocidad de entrada o salida personas en un tren por medio de una puerta determinada y, aunque proporciona datos que pueden ser de utilidad, ni restringen el uso de RailML®, ni provocan una falta de información tan grande como para ser tenida en cuenta en una primera aproximación. Esto mismo, se podría dar con muchos otros elementos que puedan ser prescindibles. Sin embargo, la gran ventaja a este respecto es que dada la sencillez de cambio de datos en un archivo RailML®, se pueden modificar dichos datos de manera rápida sin tener un sobreesfuerzo desmesurado. Esto propicia que los códigos se puedan ir formando por partes y se pueda dividir el trabajo. De este modo, y dadas las condiciones existentes en la red ferroviaria española, se da por concluido que este sistema y esta herramienta se pueden aplicar perfectamente al control de la red española así como a su organización.

## 10.3 Conclusiones personales

El mundo de la gestión ferroviaria es amplio, profundo y, sobre todo, de una altísima importancia. Entre otros, el principal motivo de esta significación es la seguridad que debe ser asegurada sin posibilidad de error, puesto que las dificultades de inherentes al ferrocarril (en control y comprobación de estado de trenes y de vías) hacen que el control tenga que ser preciso, y medido al milímetro.

Por otro lado, ha quedado patente la necesidad de unificación que hace que empresas y entidades de tan diferentes puntos del Mundo se sumerjan en un proyecto común de una envergadura tan grande, y de tan amplio espectro. Durante la ejecución del trabajo, se han consultado opiniones y materiales provenientes de países como Noruega, Alemania, Inglaterra, o Polonia entre otros. Esto pone en evidencia la importancia de este proyecto y la necesidad de montar una infraestructura común para poder abrir la puerta a la común circulación de ferrocarriles de diferentes operadores, por las vías de diferentes controladores (es decir, en un final último, para la liberalización completa del mercado).

Uno de los puntos más complicados de este proyecto ha sido la organización del reparto de tiempo entre la investigación y la generación del código. Del mismo modo, es destacable que la curva de aprendizaje y esfuerzo necesaria para el desarrollo de un proyecto relativo a esta herramienta, es muy pronunciada al principio, relajándose más y más cuanta más experiencia se

coge en el terreno. Dicho así, parece una cosa obvia, pero es importante destacar que la mayoría de esfuerzo necesario en este proyecto es relativo a la parte informática, y a la consecución de la validación del código, no siendo tanta la dificultad técnica de la parte relativa a las redes ferroviales.

Por último se ha conseguido obtener una visión comercial de qué ventajas daría la implantación de este sistema en las redes nacionales de diferentes países. Siempre, obviamente, es importante el aspecto económico en estos proyectos, por eso, se ha tenido presente el impacto económico en una posible implantación de un sistema digital y electrónico de gestión y control de ferrocarriles.

# 11 Referencias y bibliografía


- Esquemas de ejemplo para el uso de RailML, railml.org, 2016-2018, https://www.railml.org/en/download/schemes.html
- https://wiki.railml.org
- "Estadísticas de transporte de viajeros", Eurostat, 2017 https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Passenger_transport_statistics/es#Estad.C3.ADsticas_de_transporte_de_viajeros
- " The railML.org initiative", railml.org, 2018 https://www.railml.org/en/introduction/background.html
- "Namespace Policy for the Dublin Core Metadata Initiative (DCMI)", Dublin Core Group, 2018, http://dublincore.org/documents/dcmi-namespace/
- "DCMI Documents", Dublin Core Group, 2018, http://dublincore.org
- "Tags for Identifying Languages", M. Davis Y A. Phillips, 2006, https://www.ietf.org/rfc/rfc4646.txt
- "TrainClearanceGauges", railML.org, 2018, https://wiki.railml.org/index.php?title=Dev:TrainClearanceGauges
- "List of numeric codes for railway companies (RICS Code)", UIC, 2014 https://www.cit-rail.org/media/files/public/Freight/Circular_letter_23-2014_Appendix_1_List_of_codes.pdf
- "Assignment of Values To ETCS Variables", European Union Agency for Railways, 2018, https://www.era.europa.eu/sites/default/files/activities/docs/ertms_040001_etcs_variables_values_en.pdf
- "Technische Normung im Eisenbahnwesen", Dr. Karl-Otto ENDLICHER , 2012, https://www.tuv-akademie.at/fileadmin/dateien/Downloads/07_EBT_120202_Technische_Normung_Endlicher_V2.pdf
- https://uic.org/cdrom/2007/7_siafi2007/docs/april/2_mardi/atelier_bSchmitt_cuusiafi07_en.pdf
- "Consideraciones a tener en cuenta durante la construcción de nuevas plataformas de vía de alta velocidad para la posterior ubicación de instalaciones", E. Molina Soto, 2002, https://www.fomento.gob.es/recursos_mfom/consid_const_platraformas_via.pdf
- "Boletín oficial del Estado" (gálibos y semianchos de vía), Ministerio de Fomento, 2015, https://www.boe.es/boe/dias/2015/08/04/pdfs/BOE-A-2015-8765.pdf

- " Catálogo Oficial de Señales de Circulación Ferroviaria en la Red Ferroviaria de Interés General.", Ministerio de Fomento, 2017, https://www.boe.es/buscar/act.php?id=BOE-A-2017-556

# 12 Índice de tablas

## 12.1 Imágenes



## 12.2 Tablas