Guía de desarrollo, Anexo 17.2

Normativa de Proyectos Gradle

Oficina de Calidad

GERENCIA INFORMÁTICA
JOSEFA VALCÁRCEL, 44
28027-MADRID

Índice General

Índice de Ilustraciones y Tablas

1 Introducción

1.1 Objetivo

Este documento describe las directrices de configuración Gradle que deben seguir los proyectos desarrollados en la Dirección General de Tráfico, a partir de ahora DGT.

1.2 Audiencia

Este documento está dirigido a todas las personas que colaboren en labores relacionadas con la gestión, desarrollo, auditoría, implantación y explotación de los sistemas de información de la Gerencia de Informática de DGT.

1.3 Glosario

Los términos y acrónimos que se utilizan en este documento y en el resto de documentos de la guía se encuentran recogidos por orden alfabético en el Anexo 30. Glosario con el objetivo de facilitar su lectura y comprensión

1.4 Estructura del documento

Este documento está distribuido en 5 capítulos, con los siguientes contenidos:

- Capítulo 1: Introducción, contiene información relativa al propio documento
- Capítulo 2: Describe las características a tener en cuenta de Gradle
- Capítulo 3: Incluye convenciones y buenas prácticas a seguir en el desarrollo de software con Gradle
- Capítulo 4: Describe las directrices a seguir para las configuraciones Gradle
- Capítulo 5: Este capítulo incluye ejemplos de configuración para proyectos multimódulo

- Capítulo 6: Artefactos generados

1.5 Ámbito

El presente documento trata de forma genérica aquellos puntos clave de la configuración de proyectos mediante la herramienta Gradle, definiendo unas directrices a partir de las cuales los equipos de desarrollo extenderán sus funcionalidades de forma análoga, consiguiendo de esta forma que todos los proyectos Gradle contengan una configuración heterogénea y por tanto los procesos de construcción, tanto en entornos locales como de integración continua, se realicen de forma estandarizada y homogénea.

Queda fuera del alcance de este documento, definir la configuración de todas aquellas necesidades particulares de cada uno de los proyectos, teniendo que ser definidas e implementadas por los equipos de desarrollo a partir de las bases propuestas en este documento.

Toda la normativa y los enlaces a la documentación oficial se encuentran asociados a la última versión disponible current.

Todas las directrices descritas en el presente documento, y sus ejemplos, se encuentran implementados en Groovy DSL de Gradle.

1.6 Resolución de dudas

Para cualquier duda se debe realizar una consulta al Departamento de Calidad a través de la Herramienta de Gestión de servicios TI de la Gerencia de Informática.

2 Gradle

Gradle es la herramienta de automatización de construcción de aplicaciones elegida y certificada por DGT, sobre la que se ha creado la normativa que se define detalladamente en el apartado Directrices de configuración Gradle del presente documento. Cada directriz de la normativa se basa en puntos clave de la herramienta, por lo que a continuación se realizará una introducción a

aquellas características más importantes para poder llevar a cabo una correcta configuración de proyecto, que se ajuste a las directrices y buenas prácticas definidas desde la Gerencia de Informática de DGT.

2.1 Instalación de Gradle

El software Gradle se distribuye como un empaquetado en formato comprimido, por lo que su instalación es un simple proceso de extracción de los archivos contenidos en una carpeta local. Para obtener la distribución de la versión liberada de Gradle podemos acceder a la web oficial https://gradle.org/releases/.

Windows/linux:

```
unzip gradle-x.y.z-bin.zip
```

Una vez establecida la ubicación del descomprimido es necesario añadir la ubicación de la carpeta bin al path del sistema.

Como requisito previo a la instalación, es necesario comprobar que disponemos de una instalación de una JDK que se ajuste a las necesidades de la versión Gradle a instalar, para comprobar si nuestra versión Java se corresponde con las especificaciones de Gradle podemos acceder a la web oficial https://docs.gradle.org/current/userguide/installation.html#sec:prerequisites. La instalación de la JDK debe ser accesible mediante el path y en su defecto a través de la variable de entorno JAVA_HOME.

Windows:

```
echo %path%
echo %JAVA_HOME%
```

Linux:

```
echo $path
echo $JAVA_HOME
```

Para confirmar que la instalación es correcta, únicamente es necesario ejecutar el comando gradle -version y nos mostrará la información correspondiente a la versión de gradle y java que hemos configurado. Por ejemplo:

```
gradle -version
------------------------------------------------------------
Gradle x.y.z
------------------------------------------------------------

Build time:   2019-02-08 19:00:10 UTC
Revision:     f02764e074c32ee8851a4e1877dd1fea8ffb7183

Kotlin DSL:   1.1.2
Kotlin:       1.3.20
Groovy:       2.5.4
Ant:          Apache Ant(TM) version 1.9.13 compiled on July 10 2018
JVM:          1.8.0_201  (Oracle Corporation 25.73-b02)
OS:           Windows 10 10.0 amd64
```

La distribución de Gradle incorpora sus propias librerías Groovy y Kotlin por lo que nos es necesaria la instalación de una distribución de este lenguaje de programación. En caso de que en el entorno de ejecución de Gradle exista otra distribución de Groovy o Kotlin será ignorada utilizando la existente en la instalación de Gradle.

Para más información podemos consultar las indicaciones en la web oficial de Gradle https://docs.gradle.org/current/userguide/installation.html

2.1.1 Gradle Wrapper

Con el objetivo de poder ser ejecutado en cualquier máquina sin la necesidad de que ésta tenga instalada una distribución con la versión utilizada por el proyecto, Gradle ofrece la posibilidad de ser ejecutado a través de un Wrapper.

Siguiendo la recomendación de Gradle, todos los proyectos, de forma obligatoria, deberán incorporar Gradle Wrapper y la configuración de versionado necesaria para su instalación en el proyecto.

Para añadir Gradle Wrapper al proyecto únicamente es necesario incluir la configuración específica del proyecto para el Wrapper, indicando la versión utilizada por el proyecto en la propiedad gradleVersion y ejecutar dicha tarea. Para obtener más información se puede visitar la página https://docs.gradle.org/current/userguide/gradle_wrapper.html

A continuación se muestra un ejemplo de configuración:

```
wrapper {
    gradleVersion = '5.6.4'
```

2.1.2 Gradle Daemon

La herramienta Gradle dispone de un mecanismo para reducir el tiempo de construcción de los proyectos a través del cacheo de cierta información del proyecto y manteniendo vivos ciertos hilos de ejecución en segundo plano, de esta forma las ejecuciones reiteradas necesitan menos tiempos para finalizar y de esta forma se aumenta la productividad de los desarrolladores. Este mecanismo se llama Gradle Daemon y su documentación oficial se encuentra en la url https://docs.gradle.org/current/userguide/gradle_daemon.html

A partir de la versión 3.0, Gradle incorpora este mecanismo activado por defecto, por lo que en entorno de desarrollo puede ser utilizado sin necesidad de realizar ninguna configuración.

Este mecanismo no debe encontrarse activo en entornos de Integración Continua (IC), tal y como recomienda Gradle en su documentación, por lo que la plataforma IC de DGT desactivará esta funcionalidad a través de la configuración de la propiedad org.gradle.daemon en el fichero gradle.properties situado en el home del usuario, de esta forma no afectará a la configuración propia del proyecto ni será necesaria la configuración por parte de los equipos de desarrollo, como veremos en la sección 2.2 Entorno de construcción.

A continuación se muestra un ejemplo de desactivación:

```
…
org.gradle.daemon=false
…
```

"${GRADLE_USER_HOME}/gradle.properties"

2.2 Entorno de construcción

El proceso de construcción de un proyecto debe tener en cuenta ciertos cambios de entornos de ejecución, por lo que es indispensable que la configuración de la construcción tenga la capacidad de ser portables y se permita ser ejecutada en distintos entornos, cuya configuración difiere entre ellos.

Para solventar esta situación Gradle proporciona varios niveles y mecanismos de configuración de propiedades, por lo que un proyecto puede disponer de configuración asociada al entorno local de desarrollo que puede ser modificada o sustituida utilizando el mecanismo de jerarquía de configuración, para adaptarse a las particularidades de cada entorno.

El orden de jerarquía utilizado para definir el valor de las propiedades de configuración es el siguiente:

- **gradle.properties** situado en el raíz del proyecto
- **gradle.properties** situado en el directorio de usuario de Gradle (**GRADLE_USER_HOME**)
- Propiedades de sistema incluidas en la construcción a través de la línea de comandos.

Para obtener más información de las posibilidades que ofrece la configuración del entorno de ejecución y de propiedades del proyecto, aconsejamos el estudio de la documentación oficial disponible en la web https://docs.gradle.org/current/userguide/build_environment.html.

A través del mecanismo de jerarquía de propiedades incluidas en el fichero gradle.properties, todos los proyectos podrán ser construidos en cualquier entorno disponible en DGT, como por ejemplo, la plataforma de Integración Continua. Para ello es necesario que todos los proyectos sigan estrictamente la normativa de propiedades asociada al fichero gradle.properties, con el objetivo de estandarizar la nomenclatura y los valores que pueden ser utilizados en su configuración, garantizando de esta forma la correcta ejecución en cualquier entorno sin necesidad de configuraciones particulares.

Las directrices sobre las propiedades definirán la nomenclatura y los valores que deben seguir todas las propiedades incluidas en cada proyecto, además de indicar todas las propiedades catalogadas como obligatorias y aquellas que serán opcionales en función de las necesidades del proyecto.

2.3 Propiedades extras de proyecto

Para poder modelar la información, configuración y construcción de un proyecto, es necesario disponer de cierta información de forma global y accesible desde cualquier script o tarea Gradle, para ello Gradle proporciona un espacio en el que definir todas las propiedades que sean necesarias mediante el objeto ext.

En este objeto se pueden incluir distintos tipos de propiedades, pudiendo organizarlas en colecciones o arrays. Este mecanismo será uno de las bases de la normativa Gradle que se definirá en el presente documento.

A continuación se muestra un ejemplo del uso del objeto ext:

```
…
ext {
  projectInfo = [
        …
    sourceJdkVersion : '1.8',
    targetJdkVersion : '1.8'
        …
  ]
}
…
sourceCompatibility = projectInfo.sourceJdkVersion
targetCompatibility = projectInfo.targetJdkVersion
…
```

Build.gradle

Como vemos en el ejemplo, la creación de propiedades puede definirse dentro de maps Groovy, con los que facilitar e incluir información implícita agrupando conceptos.

Para obtener más información sobre la definición y el uso de propiedades del proyecto definidas por el usuario, podemos acceder a la URL https://docs.gradle.org/current/userguide/writing_build_scripts.html#example_using_extra_properties.

2.4 Aplicación de funcionalidad mediante Plugins

Al igual que muchas de las herramientas de uso genérico para la construcción de aplicaciones, Gradle se basa en un amplio catálogo de funcionalidades disponibles a partir de plugins. Éstos deben ser importados en la configuración Gradle de los proyectos, en función de las necesidades específicas de cada uno y la tecnología utilizada para diseñarlos e implementarlos. Para ello Gradle dispone de un mecanismo de aplicación que deberá ser incorporado en los ficheros gradle de los proyectos, para indicar aquellas funciones que serán utilizadas a lo largo del ciclo de vida de construcción del proyecto. Para obtener información detallada sobre las características y la aplicación de plugins, es recomendable visitar la web https://docs.gradle.org/current/userguide/plugins.html.

Existen dos formas de añadir comportamiento y extender funcionalidad a partir de **plugins** dentro de la configuración de un proyecto, Gradle los denomina **Binary plugins**, con las que podemos añadir funcionalidades existentes en el núcleo de la herramienta, estando disponibles a partir de la instalación de su distribución, y **Scripts plugins**, con los que incorporar funcionalidades a partir de ficheros **gradle** definidos e implementados dentro de los proyectos.

2.4.1 Binary plugins

Gradle proporciona un grupo de **plugins** de uso genérico asociados a tecnologías y herramientas, como pueden ser **Java**, **War**, **Jacoco**, **Maven**, etc., disponibles desde su instalación y aplicables en ficheros **build.gradle** mediante la utilización del bloque **plugins**.

A continuación se muestra un ejemplo de aplicación de plugins:

```
plugins {
    id 'java'
}
```

build.gradle

Como podemos ver en el ejemplo, el proyecto que incorpore estas líneas estarán añadiendo todas las funcionalidades del **plugin** de Gradle para **Java**, con las que podrá compilar, generar artefactos de tipo **Jar**, crear ficheros **Manifest**, etc.

Para obtener el listado completo de todos los **plugins** disponibles en el núcleo de Gradle, podemos acceder a la URL https://docs.gradle.org/current/userguide/plugin_reference.html.

2.4.2 Scripts plugins

Además de comportamientos genéricos, en función de la tecnología utilizada para su implementación, o su forma de empaquetado, distribución e instalación, los proyectos pueden necesitar generar comportamiento específico, por lo que Gradle proporciona un mecanismo de aplicación de funcionalidad a partir de scripts **DSL** propios del proyecto, almacenado en ficheros .**gradle**. Para obtener información adicional sobre los **plugins** definidos mediante scripts, podemos acceder a la documentación oficial disponible en la web https://docs.gradle.org/current/userguide/plugins.html#sec:script_plugins.

A continuación se muestra un ejemplo:

```
task testScriptPlugin {
        doLast {
                println 'apply script plugin OK'
        }
}
```

${PROJECT_ROOT}/gradle/scriptPlugin.gradle

```
apply from: 'gradle/scriptPlugin.gradle
```

build.gradle

2.5 Gradle Script Blocks

Con el objetivo de facilitar y estandarizar la configuración de proyectos, Gradle proporciona un conjunto de **bloques de configuración** asociado a cada una de las funcionalidades más genéricas de una construcción, como pueden ser la definición de dependencias, la configuración del acceso a repositorios de artefactos, la configuración de plugins como **JaCoCo**, ubicación de fuentes, etc.

Estos bloques de configuración deben incluirse en los ficheros **build.gradle** de construcción, y formarán una de las partes más importantes de la normativa que se define en el presente documento.

Cada bloque deberá seguir estrictamente las indicaciones que se dicten en el punto de definición de la normativa a tal efecto, enmarcadas dentro del punto 4 Directrices sobre coordenadas en proyectos Gradle.

Gran parte de la configuración realizada en estos bloques se deberá realizar a través de las propiedades del proyecto que hayan sido declaradas en el objeto ext, tratado en el punto 2.3 Propiedades extras de proyecto.

A continuación se muestra un ejemplo de configuración:

```
…
apply from: "gradle-project-conf/manifest.gradle"
jar {
  manifest = project.manifest {
    from projectManifest
  }
}
…
```

build.gradle

En el ejemplo podemos ver un bloque de configuración asociado al plugin **Jar**, en el que se ha configurado los datos que serán incluidos en el fichero **Manifest** durante el ensamblado del artefacto **Jar**. En él se hace referencia a una propiedad del objeto **ext** llamado projectManifest, creado en el

fichero gradle-project-conf/manifest.gradle. La utilización de ficheros .**gradle** y su aplicación se detallará a lo largo del punto 5.4 Directrices de configuración de ficheros build.gradle

Para obtener información de todos los bloques de configuración, aconsejamos la lectura de su documentación oficial disponible en la web https://docs.gradle.org/current/dsl/org.gradle.api.Project.html#N16899.

2.6 Gradle Tasks

Gradle está basado en dos conceptos básicos, **projects** y **tasks**, proyectos y tareas respectivamente. En su definición, Gradle define las tareas como trabajos **atómicos** que realizan una acción determinada, como por ejemplo compilar, generar un JAR o publicar un artefacto en un repositorio. Mientras que para Gradle un **proyecto representa una aplicación completa o partes de ella**, como por ejemplo una librería java, una aplicación web, etc.

Cada proyecto Gradle está constituido por una o varias tareas, que mediante su ejecución construirán cada una de las partes del proyecto.

Para obtener más información sobre proyectos y tareas Gradle, podemos acceder a la documentación oficial a través de la URL https://docs.gradle.org/current/userguide/tutorial_using_tasks.html#sec:projects_and_tasks

2.6.1 Ciclo de vida

Gradle define un ciclo de vida de fases estándar, **initialization**, **configuration** y **execution**. Durante la primera fase, Gradle determina que proyecto debe ser instanciado, durante la fase de configuración define qué proyectos deben ser configurados para su ejecución, y durante la última fase, Gradle ejecuta cada una de las tareas indicadas durante la ejecución del comando **gradle**. Se puede obtener una descripción de cada una de las fases en la documentación oficial disponible en la URL https://docs.gradle.org/current/userguide/build_lifecycle.html

Durante la fase de ejecución se lanzarán aquellas tareas que formen parte del ciclo de vida modelado por el **plugin** que la ofrece.

A continuación se muestra un ejemplo de ciclo de vida del **plugin Java:**

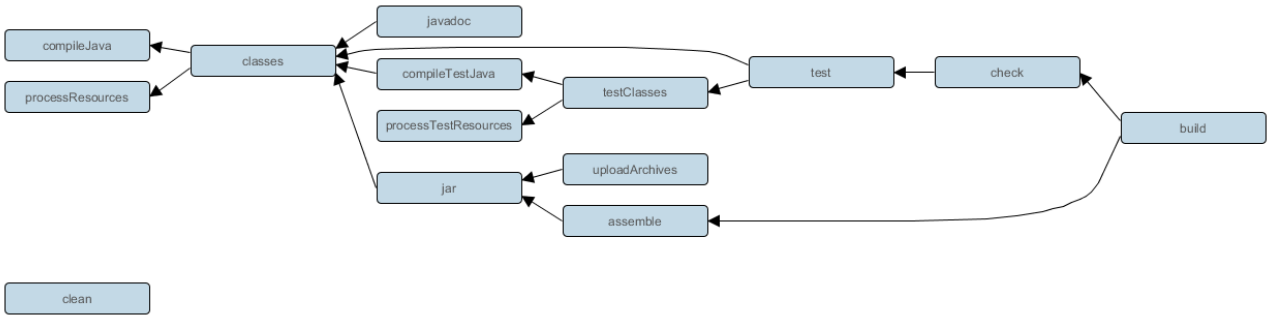

Ilustración 1. Ciclo de vida del plugin-java

Como vemos en el diagrama del ciclo de vida, las tareas se encuentran relacionadas, por lo que Gradle debe determinar que tareas debe ejecutar y en qué orden, para realizar lo solicitado por línea de comando.

Para ejecutar cada una de las tareas disponibles en un proyecto, únicamente es necesario ejecutar **gradle** junto al nombre de la **tarea** o **tareas** a ejecutar.

A continuación se muestra un ejemplo de ejecución en la que se construirá la aplicación, se generará la documentación **javadoc**, tras haber realizado un limpiado de ejecuciones anteriores.

```
gradle clean build javadoc
```

línea de comandos

La dependencia, representada mediante flechas en el diagrama, muestra las relaciones y el orden de ejecución de las fases genéricas del plugin Java de Gradle, por lo que si nos fijamos en el ejemplo anterior, la única tarea no ejecutada por herencia, es **uploadArchives**, ya que no se encuentra relacionada con la tarea **build**, siendo necesario para poder ser ejecutada que se exprese de forma explícita como se ha realizado con **javadoc** en el ejemplo.

Gradle posibilita la generación de tareas interrelacionadas mediante dependencias, con las que podemos construir nuestro proyecto de forma sencilla con una única instrucción. Para poder modificar o ampliar el alcance de las funcionalidades que nos ofrece Gradle, podemos crear tareas que implementen las necesidades de nuestro proyecto, e incluirlas, dentro del **ciclo de vida de Gradle**, mediante la definición **dependencias y herencia** de tipos.

Para obtener más información sobre el ciclo de vida Gradle, tenemos disponible la definición oficial en las web https://docs.gradle.org/current/userguide/build_lifecycle.html y https://docs.gradle.org/current/userguide/more_about_tasks.html#sec:lifecycle_tasks.

2.6.2 Definición de tareas

La implementación de una **tarea** en nuestro proyecto es bastante sencilla, ya que disponemos de una gran cantidad de tareas tipo, de las que podemos heredar su comportamiento permitiéndonos modificar algunos parámetros de configuración, ajustando de esta forma la funcionalidad que nos ofrece a nuestras necesidades.

En caso de que entre todas las tareas tipo existentes en el ecosistema Gradle no satisfagan nuestras necesidades, podremos implementar una tarea desde cero a través de **DSL**, el lenguaje generado por Gradle para implementar comportamiento específico, basado en **Groovy** o **Kotlin**. Estas tareas podrán ser configuradas para que se integren e interactúen con las tareas propias de Gradle a través de herencia y tendrán acceso a todas las propiedades del proyecto y aquellas propiedades de extensión en caso de existir.

Para definir una tarea en nuestros ficheros Gradle, es necesario indicar la palabra reservada **task**, el **nombre** que la identificará y su **implementación**. Adicionalmente y en caso de ser necesario, en la definición podremos asignarle un tipo genérico de tarea, dependencias de ejecución, y otras características de comportamiento. Para obtener más información acerca de la definición de tareas en Gradle, aconsejamos la lectura de la documentación oficial disponible en la URL https://docs.gradle.org/current/userguide/more_about_tasks.html#sec:defining_tasks.

A continuación se muestra un ejemplo de definición de tarea

```
task cleanUpRemovedProjects (type: Delete) {
        delete removedProjects.collect { "subprojects/$it" }
}
```

build.gradle

Como podemos ver en el ejemplo, la tarea se llama cleanUpRemovedProjects y es de tipo **Delete**, por lo que podrá utilizar su funcionalidad modificando algunos parámetros de su configuración. En este caso en concreto utiliza el método **delete** pasándole una colección de elementos a eliminar. Esta implementación hace uso de **Gradle build language** o **DSL**, cuya documentación oficial se encuentra en la web https://docs.gradle.org/current/dsl/.

Es indispensable tener un amplio conocimiento de cada uno de los tipos de tarea que nos ofrece Gradle a través de su lenguaje **DSL**, ya que a través de sus **propiedades**, **métodos** y **bloques de scripts**, construiremos gran parte del script de construcción de nuestro proyecto.

2.6.3 Configuración de tareas

Cada tarea del núcleo de Gradle, contiene una serie de propiedades a las que deberemos asignar valores, adaptando de esta forma su funcionalidad a las necesidades particulares de nuestro proyecto.

La configuración de tareas no se basa únicamente en la asignación de valores sobre las **propiedades** del tipo de tarea, si no que podemos utilizar todos sus **métodos** disponibles y en caso de existir, **bloques de scripts**. Para obtener información detallada sobre la configuración de tareas, es recomendable leer la documentación ofrecida por Gradle en la web https://docs.gradle.org/current/userguide/more_about_tasks.html#sec:configuring_tasks.

A continuación se muestra un ejemplo de configuración de tarea:

```
task copyConfigFiles (type: Copy) {

        from('src/main/config') {
                include '**/conf-*.properties'
        }

        into 'build/target/config'
```

```
        caseSensitive = false

        eachFile {
                println "file copied: $it.name"
        }
}
```

build.gradle

El ejemplo anterior, con el que ilustraremos esta sección, utiliza la tarea **Copy**, cuya documentación se encuentra accesible desde la URL https://docs.gradle.org/current/dsl/org.gradle.api.tasks.Copy.html.

Como vemos en la definición de la tarea copyConfigFiles, de tipo **Copy**, se encuentra configurada mediante los métodos **from**, **include**, **eachFile** e **into**, y la propiedad **caseSensitive**. Tanto Gradle DSL, como Groovy (ya que Gradle build script soporta programación mediante este lenguaje), nos permiten definir los parámetros que pasaremos a los métodos y los valores que asignaremos a las propiedades, así como la funcionalidad a ejecutar por ciertos métodos.

Para poder identificar cada una de las propiedades, métodos y bloques de las tareas del núcleo Gradle, disponemos de la documentación oficial en la web https://docs.gradle.org/current/dsl/index.html.

2.6.4 Dependencias entre tareas

Para modelar fácilmente la ejecución de tareas definidas en nuestro proyecto y que no sea necesario incluir en la línea de comandos cada una de las tareas a ejecutar, además de posibilitar la asociación entre tareas propias del proyecto y tareas del núcleo de Gradle, disponemos del mecanismo de **dependencias**.

La configuración de dependencias entre tareas se puede realizar en la misma definición, o mediante la asignación de la dependencia sobre el objeto de la tarea en una línea aparte. Independientemente de la forma que utilicemos, es necesario utilizar la palabra reservada **dependsOn**.

A continuación se muestra un ejemplo de dependencias entre tareas:

```
task signJar (type: Exec) {
        …
        println 'signJar task'
        …
}

task verifySignedJar (type: Exec, dependsOn: signJar) {
        …
        println 'verifySignedJar task'
        …
}
```

build.gradle

Como podemos ver en el ejemplo, existen dos tareas propias del proyecto, signJar y verifySignedJar, cuya dependencia se genera a través de la definición de la segunda tarea, indicando que verifySignedJar depende del resultado de signJar, por lo que si ejecutamos la tarea verifySignedJar se ejecutará antes signJar.

A continuación se muestra un ejemplo de la ejecución de las tareas:

```
gradle verifySignedJar
        :signJar
        signJar task
        :verifySignedJar
        verifySignedJar task

BUILD SUCCESFULL
```

línea de comandos

Cada tarea puede incluirse en el ciclo de vida de las tareas genéricas de Gradle, para ello es necesario que se creen dependencias entre tareas asociadas a fases y las tareas propias del proyecto.

A continuación se muestra un ejemplo de dependencia con fases o tareas del núcleo:

```
task signJar (type: Exec) {
        …
```

```
        println 'signJar task'
        …
}

task verifySignedJar (type: Exec, dependsOn: signJar) {
        …
        println 'verifySignedJar task'
        …
}

build.dependsOn verifySignedJar
```

build.gradle

Como podemos ver en el ejemplo, la tarea **build** incluye una nueva dependencia, esta vez con la tarea verifySignedJar, por lo que si ejecutamos la fase de construcción completa a través de la tarea **build**, se ejecutarán también las dos tareas específicas del proyecto por dependencia.

```
gradle build
        :signJar
        signJar task
        :verifySignedJar
        verifySignedJar task
        :build

BUILD SUCCESFULL
```

línea de comandos

Para obtener una descripción en detalle de todas las implicaciones de la dependencia entre tareas, es conveniente acceder y leer con detenimiento la documentación oficial disponible en la URL https://docs.gradle.org/current/userguide/more_about_tasks.html#sec:adding_dependencies_to_tasks.

2.6.5 Saltando ejecución de tareas

En ocasiones la construcción de los proyectos se encuentra condicionada por situaciones específicas del proyecto o del entorno, existiendo casos en los que no deben o pueden ejecutarse

ciertas tareas. Para ello Gradle dispone del mecanismo **skipping**, cuya configuración permitirá decidir si una tarea no debe ser ejecutada en una construcción determinada.

Existen varias formas de utilizar el mecanismo de **skipping**, a continuación se muestran mediante un ejemplo aquellas que serán utilizadas en la definición de la normativa:

```
task skipTaskWithProperty (type: Exec) {
        …
}
skipTaskWithProperty.onlyIf { !project.hasProperty('skipSkipTaskOWithProperty') }



task skipTaskWithFlagEnabled (type: Exec, dependsOn: signJar) {
        …
}
skipTaskWithFlagEnabled.enabled = false
```

build.gradle

Como podemos ver en el ejemplo, la tarea skipTaskWithProperty, no será ejecutada si durante la construcción existe una **propiedad de proyecto** llamada skipSkipTaskOWithProperty. Este mecanismo es apropiado para decidir en cada construcción si la tarea debe ser ejecutada o no. En cambio la tarea skipTaskWithFlagEnabled, se encuentra deshabilitada, por lo que nunca se ejecutará a menos que se elimine la línea de la asignación del valor false a la **propiedad enabled**, o se asigne el valor true a dicha propiedad, esta opción se puede aprovechar en tiempo de desarrollo para desactivar ciertas tareas ahorrando tiempo o evitando fallos mientras se modifican implementaciones que puedan afectarle.

Para obtener más información sobre los mecanismos de **skipping**, aconsejamos la lectura de la de la documentación oficial sobre este tema en disponible en la URL https://docs.gradle.org/current/userguide/more_about_tasks.html#sec:skipping_tasks

2.6.6 Ejecución de tareas por defecto

Para poder ejecutar de forma simple un grupo de tareas, Gradle dispone de mecanismos de agrupación y de asociación al proyecto para que se ejecuten por defecto sin la necesidad de incluirlas en la ejecución por línea de comandos.

Durante la normativa se definirá la obligatoriedad de la configuración de asociación de todas aquellas tareas que deban ser ejecutadas para que un proyecto se construya al completo, sin la necesidad de incluir en la línea de comandos todas aquellas tareas que no se hayan configurado como dependencias de la fase de construcción.

Para poder definir el grupo de tareas que debe ejecutarse en un proyecto para completar su construcción al completo, se debe utilizar la palabra reservada **defaultTasks**.

A continuación se muestra un ejemplo de configuración:

```
defaultTasks = 'clean', 'build', 'projectReport'
```

build.gradle

Para obtener más información sobre la designación de tareas por defecto a un proyecto, acceder a la web oficial mediante la URL https://docs.gradle.org/current/userguide/tutorial_using_tasks.html#sec:default_tasks.

2.7 Gestión de dependencias en Gradle

Gradle ofrece mecanismos de relación y dependencia de artefactos y proyectos, con el que poder utilizar librerías de terceros en un proyecto y definir relaciones entre los módulos que conforman el mismo. Existen varios tipos de relaciones, como por ejemplo los nombrados anteriormente y tipos de dependencias en función de la fase en la que se necesitarán, como por ejemplo en tiempo de compilación, de ejecución, etc. Para obtener toda la información necesaria para entender cada uno de los tipos de dependencias y como deben ser utilizados, recomendamos la lectura en profundidad de la documentación oficial sobre este tema, disponible en la URL https://docs.gradle.org/current/userguide/introduction_dependency_management.html.

Para poder definir todas las dependencias de un proyecto, Gradle proporciona la sección **dependencies**, en la que deberemos incluirlas, siguiendo la notación que se indica en la URL https://docs.gradle.org/current/userguide/declaring_dependencies.html#declaring_dependencies.

2.7.1 Dependencias de librerías de terceros

Las dependencias de librerías externas, proporcionan la posibilidad de utilizar funcionalidades de terceros en nuestras aplicaciones, pudiendo ser necesarias en tiempo de compilación y/o de ejecución. Para obtener información de cómo definir de forma adecuada cada tipo de dependencias de artefactos externos, es recomendable acceder a la web https://docs.gradle.org/current/userguide/dependency_types.html#sub:module_dependencies.

A continuación se muestra un ejemplo de dependencias de librerías externas:

```
dependencies {
   runtime "org.groovy:groovy:2.2.0@jar"
   runtime group: 'org.groovy', name: 'groovy', version: '2.2.0', ext: 'jar'
}
```

build.gradle

2.7.2 Dependencias entre proyectos

Las dependencias entre proyectos que conforman un multiproyecto, deben ser declaradas en la misma sección que las dependencias de librerías de terceros, pero a diferencia de éstas, debe seguir su propia notación, explicada en detalle en la web oficial de Gradle, disponible en la URL https://docs.gradle.org/current/userguide/dependency_types.html#sub:project_dependencies.

A continuación se muestra un ejemplo de dependencias de librerías externas:

```
dependencies {
   compile project(':otherProjectModule')
}
```

build.gradle

2.7.3 Dependencias entre proyectos para la configuración de ears

Las dependencias entre proyectos, dentro del ámbito de la configuración de un subproyecto de tipo **ear**, deben ser declaradas en la misma sección que las dependencias de librerías de terceros, pero a diferencia de éstas, deben seguir su propia notación, explicada en detalle en la web oficial de Gradle, disponible en la URL https://docs.gradle.org/current/userguide/ear_plugin.html#sec:ear_customizing

Existen dos tipos de dependencias asociadas a la configuración de proyectos **ear**, aquellas dependencias de artefactos, que deben ser ubicadas en la carpeta definida por la propiedad **libDirName** y que serán definidas mediante la directiva **earlib**, y aquellas que deben ser ubicadas en el raíz del artefacto **ear** y que serán definidas mediante la directiva **deploy**.

Las dependencias declaradas, tanto de tipo earlib como de tipo deploy, deberán seguir la notaciones definidas en los puntos anteriores Dependencias de librerías de terceros o Dependencias entre proyectos, identificando las dependencias o bien por coordenadas, en el caso de dependencias de terceros, o bien con la directiva project, en caso de dependencias de artefactos generados por subproyectos internos.

En el caso de la definición de dependencias de artefactos generados por subproyectos internos, es necesario indicar tanto la ruta del subproyecto, a través de la opción **path**, como el identificador del **configuration** que contendrá el nombre del artefacto del que se depende, teniendo en cuenta que, si no se indica ningún **configuration**, Gradle utilizará el **configuration default**, que contendrá el nombre del artefacto principal del subproyecto.

Para obtener información de cómo definir de forma adecuada cada tipo de dependencias en función **configurations** definido, es recomendable acceder a la web https://docs.gradle.org/current/userguide/managing_dependency_configurations.html

A continuación se muestra un ejemplo de dependencias para la configuración de un ear:

```
dependencies {
    earlib ‘org.groovy:groovy:2.2.0@jar’
    earlib group: 'org.groovy', name: 'groovy', version: '2.2.0', ext: 'jar'
```

```
        earlib project(path: ':otherProjectModuleWeb')
        deploy project(path: ':otherProjectModuleWeb')

        // Incuirá el/los artefactos incluidos en el configurations archives
        deploy project(path: ':otherProjectModuleWeb', configuration: 'archives')
}
```

build.gradle

2.7.4 Configuración de acceso al repositorio de artefactos

Para que Gradle identifique y descargue aquellos artefactos necesarios para la compilación, o su empaquetado en los distribuibles y que de esta forma se encuentren accesibles durante la ejecución de la aplicación, es necesario que los proyectos definan los accesos a los repositorios, en los que se encuentran todas y cada una de las dependencias definidas, en la sección **repositories**.

A continuación se muestra un ejemplo de configuración de repositorio:

```
repositories {
        maven {
                url “http://repo.mycompany.com/maven2”
        }
}
```

build.gradle

Para conocer con más profundidad aquellos repositorios pre-configurados en Gradle, y los datos a definir necesarios para la conexión de aquellos que no viene configurados por defecto, podemos acceder a la URL

https://docs.gradle.org/current/userguide/declaring_repositories.html.

2.7.5 Configuración de acceso al repositorio de plugins

Para que Gradle identifique y descargue aquellos plugins definidos en los proyectos, y que de esta forma se encuentren accesibles durante la ejecución de la aplicación, existen dos posibilidades,

que el proyecto acceda directamente al repositorio *plugins.gradle.org*, en cuyo caso el proyecto no deberá realizar configuración alguna, ya que es el mecanismo por defecto de Gradle, o acceder a un repositorio *mirror*, de la organización en la que se encuentre, que permita el acceso diferido a los plugins expuestos en el repositorio *plugins.gradle.org*, en cuyo caso el proyecto deberá configurar el acceso al repositorio que actuará de espejo.

Para definir el acceso a un repositorio *mirror*, es necesario que se configure de forma adecuada la sección **pluginManagement** dentro del fichero **settings.gradle**.

A continuación se muestra un ejemplo de configuración de repositorio *mirror* de plugins:

```
pluginManagement {
    repositories {
        maven {
            url "http://repo.mycompany.com/maven2/plugins-gradle-mirror"
        }
    }
}
```

settings.gradle

Para conocer en profundidad las características de Gradle, en relación a la configuración de repositorios de tipo *mirror* de su repositorio oficial *plugins.gradle.org*, recomendamos la lectura en detalle de su documentación oficial disponible en las URL https://plugins.gradle.org/docs/mirroring y https://docs.gradle.org/current/userguide/plugins.html#sec:plugin_management.

2.8 Multiproyectos en Gradle

Gradle dispone de mecanismos y directrices sobre la configuración, estructura y construcción de proyectos que estén formados por varios módulos. Tal y como se indica en la documentación de Gradle, existe una gran cantidad de proyectos que no pueden o deben ser estructurados de forma monolítica, ya que su complejidad y dimensiones lo harían inmanejable y no permitirían ser mantenidos de forma factible.

Desde DGT aconsejamos hacer uso de estructuras multiproyectos en todos aquellos proyectos que generan varios artefactos de tipo **jar**, **war** o **ear**, ya que de esta forma, tanto la configuración, como la construcción podrá ser gestionada por ficheros **build**.gradle específicos, aumentando la versatilidad de la construcción y posibilitando la construcción completa del proyecto o de cada uno de los módulos por separado.

Para conocer en profundidad las características de Gradle en relación a multiproyectos recomendamos la lectura en detalle de la web:

https://docs.gradle.org/current/userguide/intro_multi_project_builds.html.

2.8.1 Estructura multiproyecto

La estructura de un proyecto con varios módulos debe seguir las indicaciones expuestas en la siguiente URL:

https://docs.gradle.org/current/userguide/intro_multi_project_builds.html#sec:structure_of_a_multiproject_build.

En ella se indica que cada proyecto debe contener un fichero **build.gradle** específico, además de disponer en el raíz del proyecto de los ficheros **build.gradle** y **settings.gradle**.

A continuación se indica un ejemplo de estructura:

```
Root project 'multiproject'
+--- Project ':project1'
                \---- build.gradle
+--- Project ': project2'
                \---- build.gradle
+--- Project ': project3'
                \---- build.gradle
+--- settings.gradle
\--- build.gradle
```

Como se puede ver en el ejemplo el proyecto principal debe contener un fichero **settings.gradle**, en el que se definirá el nombre del proyecto y se declararán los proyectos que lo

conforman, y un fichero **build.gradle** con la configuración global del proyecto. Además cada proyecto debe contener un fichero **build.gradle** con la configuración específica del proyecto al que pertenezca.

Es posible modificar los nombres de los ficheros Gradle de construcción de los subproyectos, al igual que su ubicación, pero en caso de realizar dicho cambio, es necesario que se indique de forma explícita el acceso y nombre del proyecto y su fichero de construcción. Desde DGT no aconsejamos la modificación de los nombres y ubicaciones estándares, pero en caso de ser necesario deberá ser aprobado por la oficina de calidad de DGT.

2.8.2 Construcción de multiproyectos

La ejecución de tareas en un proyecto múltiple se realiza de la misma manera que un proyecto individual, a través de la ejecución del comando **gradle** y el nombre de la tarea. Debido a la gran diversidad de módulos que compondrán un proyecto, es posible ejecutar tareas específicas de un módulo de forma independiente, ya sea cambiando la ubicación de la carpeta root a la hora de ejecutar gradle, o mediante la composición del nombre de la tarea a partir del nombre del módulo y el nombre de la tarea.

A continuación se muestra un ejemplo de comando de ejecución completa:

```
gradle build
    :project1:build
    :project2:build
    :project3:build
```

Ubicado en el raíz del proyecto principal

A continuación se muestra un ejemplo de composición de nombre de tarea:

```
gradle :project2:build
    :project2:build
```

Ubicado en el raíz del proyecto principal

A continuación se muestra un ejemplo ubicándonos en el raíz de un proyecto:

```
gradle build
    :build
```

Ubicado en el raíz del proyecto project1

Para obtener información completa de la construcción y ejecución de tareas de un proyecto multimódulo, deberemos acceder a la web:

https://docs.gradle.org/current/userguide/intro_multi_project_builds.html#sec:executing_a_multiproject_build.

2.8.3 Composite builds

Existe una forma adicional de definir proyectos que conforma un multiproyecto, realizando la composición de proyectos múltiples a través de la inclusión de ejecuciones de construcción de otros proyectos en el ciclo de vida del proyecto principal. Para obtener más información se puede acceder a la web https://docs.gradle.org/current/userguide/composite_builds.html

3 Convenciones y buenas prácticas

Con el objetivo de seguir estándares relacionados con la gestión, configuración y construcción de proyectos, desde el área de calidad de DGT proponemos la adaptación y utilización de todas las siguientes buenas prácticas que harán que todos los proyectos sigan una misma organización, facilitando la homogeneización de la configuración de todos los proyectos, y por lo tanto simplificando la automatización de los procesos de construcción y mantenimiento.

3.1 Identificación de proyectos, aplicaciones y artefactos

La identificación de proyectos debe ser gestiona siguiendo el estándar de proyectos Java, utilizando las propiedades **grupo**, **nombre** y **versión**, que se corresponden a las coordenadas de cualquier artefacto generado en tecnología Java.

- **grupo**: el nombre de dominio debe ser creado mediante subgrupos generados a partir de identificadores en minúscula y separados por puntos. Ejemplo: org.dgt.project
- **nombre**: el nombre del proyecto y/o artefacto deberá formarse con palabras en minúsculas separadas por guiones. Ejemplo: api-project
- **versión**: la versión debe seguir el estándar de versionado semántico de software. Ejemplo: 3.0.2 (ver apartado Convención de versionado)

Este grupo de propiedades, que identifican a un proyecto o artefacto, debe ser continuo en todas las fases del ciclo de vida del proyecto y existir en todos los artefactos y distribuibles de la aplicación, de esta forma la identificación será sencilla, rápida y coherente.

3.2 Convención de versionado

Uno de las propiedades de identificación de un proyecto más importante y compleja, es la versión, ya que puede ser utilizada tanto para identificar una distribución, como para gestionar el ciclo de vida de un proyecto, además de ofrecer información relevante a la hora de comparar distintas versiones de la misma aplicación.

Toda la información sobre la normativa de versionado establecida en la DGT se encuentra en el "Anexo 38 Normativa de Versionado" de la guía de desarrollo.

3.3 Convención de ubicación de ficheros

La herramienta Gradle, al igual que otras de su mismo ámbito, utilizan **plugins** para aumentar su funcionalidad y proporcionan mecanismos para aumentar la productividad de las construcciones de aplicaciones. Cada lenguaje de programación, como puede ser Java, y cada herramienta que proporciona una funcionalidad adicional a Gradle a través de un **plugin**, como puede ser **JaCoCo**, definen una serie de ubicaciones que deben contener los archivos con los que trabajarán, por lo que para garantizar el buen funcionamiento de Gradle y su correcta configuración, se deben seguir estrictamente las recomendaciones de cada una de las herramientas que se utilicen, así como de cada uno de los **plugins** asociados a herramientas.

Como ejemplo utilizaremos las especificaciones del plugin Java de Gradle, disponibles en la siguiente URL https://docs.gradle.org/current/userguide/java_plugin.html#sec:java_project_layout.

Directory	Meaning
src/main/java	Production Java source
src/main/resources	Production resources
src/test/java	Test Java source
src/test/resources	Test resources
src/sourceSet/java	Java source for the given source set
src/sourceSet/resources	Resources for the given source set

Tabla 1. Ubicación de los ficheros

Como vemos en la tabla, cada tipo de fichero dispone de una ubicación determinada, que cada proyecto deberá seguir obligatoriamente para todos los **plugins** utilizados siguiendo sus especificaciones, para que de esta forma tanto el proyecto, Gradle, sus **plugins** y la plataforma de Integración Continua de DGT trabajen de forma heterogénea y estandarizada.

4 Directrices sobre coordenadas en proyectos Gradle

La normativa de coordenadas definida en este documento tiene como objetivo que todos los proyectos desarrollados para la DGT sigan un mismo estándar que garantice la interoperabilidad entre proyectos, el correcto mantenimiento de los mismos y la homogeneidad de configuración Gradle incluida en todos sus ficheros **settings.gradle** y **build.gradle**.

A continuación, se definen los esquemas que deben seguir los valores asignados a los elementos group, name (nombre del proyecto principal en rootProject.name y nombre de carpeta en subproyectos) y version que definirán a todos los proyectos, módulos y artefactos generados bajo la normativa:

- **Grupo**:
    - Proyectos: es.trafico.acronimo
    - Proyectos con subsistemas: es.trafico.acronimo.subsistema
    - Proyectos del framework DGT: es.trafico.framework.acronimo

  Dónde acrónimo se corresponde con el ACRÓNIMO del proyecto y subsistema el SUBSISTEMA dentro del proyecto global, éste último se indicará en caso de que el proyecto haya sido dividido en subsistemas. En el caso de los proyectos asociados a componentes comunes del framework DGT el grupo deberá ser siempre es.trafico.framework.

  El grupo asignado al proyecto principal debe extenderse a todos sus subproyectos, haciendo que todos los proyectos que conforman el multiproyecto pertenezcan al mismo grupo.

  Todo grupo debe seguir las buenas prácticas y recomendaciones indicadas en el apartado Identificación de proyectos, aplicaciones y artefactos.

- **Nombre**:
    - Proyectos multiproyecto (proyecto principal): acronimo
    - Proyecto multiproyecto con subsistemas (proyecto principal): acronimo-subsistema
    - Subproyecto: nombre-subproyecto

  El nombre debe coincidir con el nombre del módulo/componente definido en la arquitectura del proyecto, siendo exactamente el nombre de la carpeta que contiene al fichero build.gradle que lo define.

  Este nombre será utilizado por Gradle para nombrar a los artefactos generados, añadiéndole la versión del proyecto separada por un guion, tal y como se indica en su especificación. Por ejemplo: nombre-subproyecto-1.0.0.jar. Aunque en ocasiones podrá modificarse el

nombre con el que se generará el artefacto a través de la propiedad archiveName, en función de las necesidades del proyecto y en casos puntuales.

Todo nombre debe seguir las buenas prácticas y recomendaciones indicadas en el apartado Identificación de proyectos, aplicaciones y artefactos.

- Versión: el valor de la versión deberá cumplir siempre lo indicado en el apartado Convención de versionado.

A continuación se pueden ver algunos ejemplos de configuración en proyectos multiproyecto:

- Proyecto multiproyecto sin subsistemas (Acrónimo: ACRO)

```
Root project 'acro'
+--- \dgt-api'
                +---- build.gradle
+--- \dgt-ws-soap
                +---- build.gradle
+---\dgt-ws-rest
                +---- build.gradle
+--- settings.gradle
\--- build.gradle
```

${ROOT_PROJECT}

```
…
rootProject.name = ‘acro’

include ‘dgt-api’
include ‘dgt-ws-soap’
include ‘dgt-ws-rest’
```

acro/settings.gradle

```
//Información global de proyecto
ext {
  projectInfo = [
    …
```

```
    acronimo : 'ACRO',
    group: "es.trafico.${project.name}",
    name : "${project.name}",
    version : '1.0.0-SNAPSHOT',
…
 ]
}
…
//Configuración global para el proyecto y todos sus subproyectos
allprojects {
 //Identificación del proyecto
 group = projectInfo.group
 version = projectInfo.version
…
```

acro/build.gradle

- Proyecto multimódulo con subsistemas (Acrónimo: ACRO, subsistema: INTRA)

```
Root project 'acro-intra'
+--- \dgt-api'
                +---- build.gradle
+--- \dgt-ws-soap
                +---- build.gradle
+---\dgt-ws-rest
                +---- build.gradle
+--- settings.gradle
\--- build.gradle
```

${ROOT_PROJECT}

```
…
rootProject.name = ‘acro-intra’

include ‘dgt-api’
include ‘dgt-ws-soap’
include ‘dgt-ws-rest’
```

acro-intra/settings.gradle

```
//Información global de proyecto
ext {
  projectInfo = [
    …
    acronimo : 'ACRO,
    group: "es.trafico.${project.name.replaceAll('-','.')}",
    name : "${project.name}",
    version : '1.0.0-SNAPSHOT',
…
  ]
}
…
//Configuración global para el proyecto y todos sus subproyectos
allprojects {
  //Identificación del proyecto
  group = projectInfo.group
  version = projectInfo.version
…
```

acro-intra/build.gradle

- Proyecto multimódulo asociado al framework DGT (Acrónimo CXCP)

```
Root project 'cxcp'
+--- \pojo-cxcp'
                +---- build.gradle

+--- settings.gradle
\--- build.gradle
```

${ROOT_PROJECT}

```
…
rootProject.name = 'cxcp'

include 'pojo-cxcp'
```

cxcp/settings.gradle

```
//Información global de proyecto
ext {
  projectInfo = [
    …
    acronimo : 'CXCP',
    group: "es.trafico.framework",
    name : "${project.name}",
    version : '1.0.0-SNAPSHOT',
…
  ]
}
…
//Configuración global para el proyecto y todos sus subproyectos
allprojects {
  //Identificación del proyecto
  group = projectInfo.group
  version = projectInfo.version
…
```

cxcp/build.gradle

5 Directrices de configuración Gradle

A lo largo de este punto se definirá la normativa de configuración de proyectos gestionados por Gradle, a la que se deberá ajustar todos los proyectos desarrollados para y por DGT. Cada punto que conforma esta sección hace uso de las explicaciones realizadas sobre las características de Gradle indicadas en el capítulo 2 Gradle, y harán referencia a las buenas prácticas declaradas en el capítulo 3 Convenciones y buenas prácticas.

Es necesario tener en cuenta que existirán casos en los que la aplicación de algunos puntos, definidos como obligatorio, no puedan realizarse, en cuyo caso, los equipos de desarrollo se deberán poner en contacto con la oficina de calidad de DGT para solicitar un acuerdo de no utilización, indicando los motivos por lo que no es viable ajustarse a la normativa.

Cada punto expuesto en la presente normativa, busca ser un estándar para todos los proyecto, creando diseños y configuraciones aplicables a cualquier proyecto de forma genérica. Este diseño generalista posibilita su rediseño para aquellos proyectos en los que no tenga cabida, pero es necesario tener en cuenta que la filosofía de diseño plasmada debe respetarse.

Se marcarán con un **<obligatorio>** en rojo aquellas directrices de carácter obligatorio y con un **[opcional]** en azul, para aquellas indicaciones que puedan ser opcionales según el tipo de proyecto y sus necesidades.

5.1 Directrices de nomenclatura de ficheros gradle

Con el objetivo de garantizar que todos los proyectos puedan ser migrados de forma inmediata a una versión superior de Gradle, y no generar configuración extra en los proyectos que dificulten su mantenibilidad y construcción automatizada, es obligatorio que todos los ficheros de propietarios sigan la siguiente directriz:

<obligatorio> Los ficheros de configuración de Gradle deberán seguir la nomenclatura estándar de la herramienta, sin personalizar los ficheros **gradle.properties**, **build.gradle**, **settings.gradle**, etc., modificando su nombre, ni ubicación.

5.2 Directrices de configuración y contenido de ficheros settings.gradle

Con la finalidad de que todos los proyectos se ajusten a la configuración por defecto ofrecida por el núcleo de Gradle, y no disponer de configuración adicional en los proyectos que dificulten su mantenibilidad y legibilidad, es obligatorio que el contenido de los ficheros **settings.gradle** sigan las siguientes directrices:

- **<obligatorio>** El fichero debe contener el nombre del proyecto, utilizando para ello la propiedad Gradle rootProject.name, siguiendo las recomendaciones expuesta en el apartado Directrices sobre coordenadas en proyectos Gradle.

- **<obligatorio>** Los módulos que conforma proyectos múltiples, deben ser declarados en el fichero **settings.gradle** del proyecto principal. Esta declaración se realiza utilizando la sentencia **include** y el nombre del módulo o proyecto a incluir. Cada módulo debe encontrarse en una carpeta, situada en el raíz del proyecto, y cuyo nombre debe coincidir con el declarado en la sentencia **include**.
- **<obligatorio>** Todos los módulos que conformen el proyecto múltiple, deben indicarse en una sentencia **include** propia.
- **<obligatorio>** El fichero **settings.gradle** debe contener la configuración para que el gestor de plugins de Gradle acceda al repositorio *mirror* expuesto en el Artifactory de la DGT. Esta configuración debe realizarse a través de la sección **pluginManagement**, definiendo en su interior el repositorio de tipo **maven** que dará acceso a los plugins dentro de la sección **repositories,** utilizando para ello la propiedad de entorno definida en el fichero **gradle.properties** creada para tal efecto, artifactoryGradlePluginsUrl.

  A continuación se muestra un ejemplo:

```
Root project 'dgt-multiproject'
+--- \dgt-api'
                +---- build.gradle
+--- \dgt-ws-soap
                +---- build.gradle
+---\dgt-ws-rest
                +---- build.gradle
+--- settings.gradle
\--- build.gradle
```

${ROOT_PROJECT}

```
pluginManagement {
   repositories {
      maven {
         url "${artifactoryGradlePluginsUrl}"
      }
   }
}
```

```
rootProject.name = 'dgt-multiproject'

include 'dgt-api'
include 'dgt-ws-soap'
include 'dgt-ws-rest'
```

settings.gradle

5.3 Directrices de propiedades de entorno definidas en gradle.properties

Con el objetivo de garantizar que todos los proyectos utilizan la misma nomenclatura de propiedades, la misma ubicación y el mecanismo propuesto por Gradle para la configuración de variables dependientes de entorno, todos los proyectos deben seguir las siguientes directrices obligatorias y opcionales:

- **<obligatorio>** Todos los proyectos creados con tecnología **Java** deben contener las propiedades de proyecto que indiquen la ubicación del compilador en su versión necesaria. Es obligatorio que se siga la nomenclatura indicada, pero es opcional el número de propiedades configuradas, ya que todos los proyectos no deberán, ni podrán ser compilados en todas las versiones. Los entornos de Integración Continua tendrán todas las propiedades configuradas en su fichero **gradle.properties**.
    - pathJdk5=<ubicación ejecutable javac de la versión 1.5 de Oracle>
    - pathJdk5Ibm=<ubicación ejecutable javac de la versión 1.5 de IBM>
    - pathJdk6=<ubicación ejecutable javac de la versión 1.6 de Oracle>
    - pathJdk6Ibm=<ubicación ejecutable javac de la versión 1.6 de IBM>
    - pathJdk7=<ubicación ejecutable javac de la versión 1.7 de Oracle>
    - pathJdk7Ibm=<ubicación ejecutable javac de la versión 1.7 de IBM>
    - pathJdk8=<ubicación ejecutable javac de la versión 1.8 de Oracle>
    - pathJdk8Ibm=<ubicación ejecutable javac de la versión 1.8 de IBM>
- **<obligatorio>** Todos los proyectos deben incorporar las propiedades necesarias para la conexión con **Artifactory**, y de esta forma poder acceder correctamente al sistema en cualquier entorno de construcción. Para ello es indispensable que se incluyan, manteniendo su nomenclatura, las siguientes propiedades:

  - **artifactoryGradlePluginsUrl**=<url del repositorio mirror de plugins gradle del Artifactory DGT >
  - **artifactoryDependenciesUrl**=<url del repositorio central del Artifactory DGT>
  - **artifactoryUser**=<usuario para la conexión con Artifactory>
  - **artifactoryPassword**=<contraseña para la conexión con Artifactory>
- **[opcional]** Todos los proyectos que necesiten realizar firma de artefactos, deberán incluir la configuración del Keystore con el que realizar la firma en cualquier entorno de construcción sin necesidad de modificar la configuración de tareas.
  - **jarsignerWorkingDir**=<ubicación del de la herramienta jarsigner>
  - **jarsignerKeystore**=<ubicación de la keystore>
  - **jarsignerAlias**=<alias del certificado>
  - **jarsignerPassword**=<contraseña>
- Los valores asignados en todas las propiedades indicadas anteriormente serán sustituidos por los valores asignados en el fichero **gradle.properties** del entorno en el que se ejecute el proyecto, garantizando de esta forma la portabilidad y construcción entre entornos.

Aquellos proyectos que necesiten o deseen incorporar propiedades de configuración, a partir del fichero **gradle.properties**, deben seguir la notación **camelCase** en su definición.

A continuación se muestra un ejemplo:

```
#Configuración de compilación
pathJdk7=C:/herramientas/Java/jdk1.7.0_71
pathJdk8=C:/herramientas/Java/jdk1.8.0_191

#Configuración de acceso a artifactory
artifactoryGradlePluginsUrl=http://artifactory.trafico.es:8080/artifactory/dgt-gradle-plugins
artifactoryDependenciesUrl=http://artifactory.trafico.es:8080/artifactory/dgt-central-ic
artifactoryUser=ic-reader
artifactoryPassword=usrIC@reader

#Configuración de firma electrónica
jarsignerWorkingDir=C:/herramientas/Java/jdk1.8.0_191/bin
jarsignerKeystore=C:/firma-digital/keystore/firma.jks
jarsignerAlias=alias
jarsignerPassword=password
```

```
#Configuración de propia del proyecto
propiedadUno=valor-asignado
propDos=valor asignado
```

gradle.properties

5.4 Directrices de configuración de ficheros build.gradle

A lo largo de este punto se indicarán las directrices que se deben seguir para configurar el proyecto, como pueden ser las dependencias, los repositorios, las propiedades extras, entre otras, además de mostrar la guía diseño que se debe seguir para la generación de tareas y su configuración.

Cada punto de la normativa aparece siguiendo el orden que deberá aplicarse en la configuración de todos los proyectos para mantener una misma y de esta forma aumentar la legibilidad y mantenibilidad. El orden de secciones deberá ser el siguiente, dependiendo del tipo de proyecto:

- build.gradle de proyecto principal
  - **<obligatorio>** Aplicación de **plugins**
  - **<obligatorio>** Tareas por defecto
  - **<obligatorio>** Propiedades del objeto **ext** projectInfo
  - **<obligatorio>** Ejecución de fichero gradle asociados al entorno (environmentBuildConfig.gradle)
  - **<obligatorio>** Declaración de la descripción del proyecto
  - **<obligatorio>** Sección **allprojects**
    - Declaración del grupo del proyecto
    - Declaración de la versión del proyecto
    - Configuración de accesos a repositorios
  - **<obligatorio>** Configuración de la tarea **wrapper**
- build.gradle de sub-proyecto de tipo Java
  - **<obligatorio>** Aplicación de **plugins**
  - **<obligatorio>** Declaración de la descripción del sub-proyecto
  - **<obligatorio>** Configuración de compatibilidad de versionado Java
  - **<obligatorio>** Configuración de fork para la compilación de fuentes Java
  - **<obligatorio>** Aplicación de fichero gradle de dependencias y declaración de las mismas

- **<obligatorio>** Aplicación del fichero gradle asociado al fichero manifest y su configuración dentro del plugin jar
- **<obligatorio>** Configuración del plugin jar
- **<obligatorio>** Aplicación del fichero gradle asociado a la generación de artefactos javadoc
- **<obligatorio>** Aplicación del fichero gradle asociado a la generación de artefactos sources
- **<obligatorio>** Aplicación del fichero asociado a la firma de artefactos

- build.gradle de sub-proyecto de tipo War
  - **<obligatorio>** Aplicación de **plugins**
  - **<obligatorio>** Declaración de la descripción del sub-proyecto
  - **<obligatorio>** Aplicación de fichero gradle de dependencias y declaración de las mismas
  - **<obligatorio>** Aplicación del fichero gradle asociado al fichero manifest y su configuración dentro del plugin war
  - **<obligatorio>** Configuración del plugin war

5.4.1 Directrices de aplicación de plugins

Con el fin de homogeneizar todas las configuraciones de los proyectos, y de esta forma aumentar la mantenibilidad de los proyectos, es indispensable que todos los proyectos sigan la siguiente directriz:

- **<obligatorio>** Dentro del orden de configuración de ficheros **build.gradle**, la aplicación de **plugins** debe tomar la primera posición de forma obligatoria.
- **<obligatorio>** Es obligatorio aplicar los **plugins** con la notación creada en las nuevas versiones de Gradle, **plugins DSL**. (plugins {…})

A continuación se muestra un ejemplo:

```
//Aplicación de plugins
plugins {
    ...
    id 'java'
    id 'jacoco'
    id 'project-report'
    ...
}
```

build.gradle

5.4.1.1 Plugin java

Con el objetivo de garantizar la estandarización y la mantenibilidad de configuración Gradle dentro de los proyectos, es obligatorio que todo proyecto/módulo que utilice el **plugin java**, siga las indicaciones y buenas prácticas que a continuación se indican, definidas en la documentación oficial del **plugin java**, disponible en la web https://docs.gradle.org/current/userguide/java_plugin.html.

- **<obligatorio>** Todo proyecto se debe ajustar a la convención de ubicación de código productivo y código de pruebas, sin realizar modificaciones de su configuración por defecto. https://docs.gradle.org/current/userguide/java_plugin.html#source_sets
- **<obligatorio>** Todos los proyectos deben ajustarse a la convención de ubicación de ficheros según tipología, sin realizar modificaciones a la configuración por defecto. https://docs.gradle.org/current/userguide/java_plugin.html#sec:java_project_layout
- **<obligatorio>** Todo proyecto debe ajustarse a la convención de propiedades y sus valores por defecto, sin realizar modificaciones que alteren el comportamiento del plugin. https://docs.gradle.org/current/userguide/java_plugin.html#sec:java_convention_properties
- **<obligatorio>** Todos los proyectos deben ajustarse y no realizar modificaciones de valores por defecto de las propiedades asociadas a código. https://docs.gradle.org/current/userguide/java_plugin.html#sec:source_set_properties

5.4.1.2 Plugin war

Con el objetivo de garantizar la estandarización y la mantenibilidad de configuración Gradle dentro de los proyectos, es obligatorio que todo proyecto/módulo que utilice el **plugin war**, siga las indicaciones y buenas prácticas que a continuación se indican, definidas en la documentación oficial del **plugin war**, disponible en la web https://docs.gradle.org/current/userguide/war_plugin.html

- **<obligatorio>** Todo proyecto se debe ajustar a la convención de ubicación de ficheros asociados a la generación de artefactos de tipo war. https://docs.gradle.org/current/userguide/war_plugin.html#sec:war_project_layout
- **<obligatorio>** Todo proyecto debe ajustarse a la convención de propiedades y sus valores por defecto, sin realizar modificaciones que alteren el comportamiento del plugin. https://docs.gradle.org/current/userguide/war_plugin.html#sec:war_convention_properties

- **<obligatorio>** Todos los proyectos deberán ajustarse a la configuración por defecto, sin realizar modificaciones de su comportamiento injustificadamente. https://docs.gradle.org/current/userguide/war_plugin.html#sec:war_default_settings

5.4.1.3 Plugin ear

Con el objetivo de garantizar la estandarización y la mantenibilidad de configuración Gradle dentro de los proyectos, es obligatorio que todo proyecto/módulo que utilice el **plugin ear**, siga las indicaciones y buenas prácticas que a continuación se indican, definidas en la documentación oficial del **plugin ear**, disponible en la web https://docs.gradle.org/current/userguide/ear_plugin.html.

- **<obligatorio>** Todo proyecto se debe ajustar a la convención de ubicación de ficheros asociados a la generación de artefactos de tipo ear. https://docs.gradle.org/current/userguide/ear_plugin.html#sec:ear_project_layout
- **<obligatorio>** Todo proyecto debe ajustarse a la convención de propiedades y sus valores por defecto, sin realizar modificaciones que alteren el comportamiento del plugin. https://docs.gradle.org/current/userguide/ear_plugin.html#sec:ear_convention_properties
- **<obligatorio>** Todos los proyectos deberán ajustarse a la configuración por defecto, modificando sólo aquellas propiedades y opciones que sean indispensables para la correcta configuración del plugin y la generación del artefacto ear. https://docs.gradle.org/current/userguide/ear_plugin.html#sec:ear_customizing

5.4.1.4 Plugin jacoco

Con el objetivo de garantizar la estandarización y la mantenibilidad de configuración Gradle dentro de los proyectos, y garantizar la correcta integración entre los informes generados por **Jacoco** y herramientas como SonarQube, es obligatorio que todos los proyectos/módulos que contengan código java, incorporen la aplicación del **plugin jacoco**, siguiendo las indicaciones y buenas prácticas que a continuación se indican, definidas en la documentación oficial del **plugin jacoco**, disponible en la web https://docs.gradle.org/current/userguide/jacoco_plugin.html.

- **<obligatorio>** Todo proyecto debe ajustarse a la configuración por defecto de la propiedad **resportsDir**, sin realizar modificaciones de su valor. https://docs.gradle.org/current/userguide/jacoco_plugin.html#sec:configuring_the_jacoco_plugin

5.4.1.5 Plugin project-report

Con el objetivo de garantizar la estandarización y la mantenibilidad de configuración Gradle dentro de los proyectos, y garantizar la correcta integración entre los informes generados y la explotación automatizada de los mismos, es obligatorio que todos los proyectos/módulos incorporen la aplicación del **plugin project-report**, siguiendo las indicaciones y buenas prácticas que a continuación se indican, definidas en la documentación oficial del disponible en la web https://docs.gradle.org/4.4/userguide/project_reports_plugin.html.

- **<obligatorio>** Todo proyecto debe ajustarse a la convención de propiedades y sus valores por defecto, sin realizar modificaciones que alteren el comportamiento del plugin. https://docs.gradle.org/4.4/userguide/project_reports_plugin.html#sec:project_reports_convention_properties

5.4.1.6 Plugin sonarqube

Con el fin de que todos los proyectos sigan el mismo mecanismo de integración con **SonarQube** y que la automatización de los análisis sea lo menos intrusiva en los proyectos posible, es obligatorio que todos los proyectos (no teniendo que ser configurada en los módulos), incorporen la aplicación del **plugin org.sonarqube** en la última versión indicada a continuación, sin hacer configuraciones adicionales, ya que toda su configuración se aplicará mediante ficheros **gradle** inyectados al proyecto en el entorno de Integración Continua.

- **<obligatorio>** Todo proyecto deberá aplicar la última versión disponible y compatible con la instalación de **SonarQube** de la plataforma de Integración Continua, siendo esta la versión '2.7'.

A continuación, se muestra un ejemplo:

```
//Aplicación de plugins
plugins {
    ...
    id 'org.sonarqube' version '2.7'
    ...
}
```

build.gradle

5.4.1.7 Plugin DependencyCheck

Para todos aquellos proyectos que quieran incorporar el análisis de dependencias externas en el análisis de SonarQube, los proyectos pueden incorporar la aplicación del **plugin org.owasp.dependencycheck**, sin hacer configuraciones adicionales, ya que toda su configuración se aplicará mediante ficheros **gradle** inyectados al proyecto en el entorno de Integración Continua.

- **[opcional]** Todo proyecto que desee hacer uso de la herramienta DependencyCheck.

A continuación, se muestra un ejemplo:

```
//Aplicación de plugins
plugins {
    ...
    id 'org.owasp.dependencycheck' version '6.1.5'
    ...
}
```

build.gradle

5.4.2 Directrices de configuración de propiedades en el objeto ext

Para poder disponer de acceso a toda la información del proyecto de forma rápida y estructurada, además de centralizar la mayoría de las propiedades con la que se configurarán los plugins y su comportamiento, es necesario que todas los ficheros build.gradle de proyectos principales, contengan la declaración del objeto ext y las siguientes secciones de propiedades, justo después de la aplicación de plugins definida en el punto anterior.

5.4.2.1 Propiedades de información del proyecto

Todos los proyectos deben seguir para la definición del objeto `ext` las siguientes directrices asociadas a propiedades de información global del proyecto:

- **<obligatorio>** Todos los proyectos deben declarar una propiedad de tipo `map` llamada projectInfo.
- **<obligatorio>** La propiedad projectInfo debe contener las siguientes propiedades debidamente cumplimentadas:

- **<obligatorio> description**: contendrá una descripción breve de la aplicación
- **<obligatorio> organization**: contendrá el literal 'Dirección General de Tráfico'
- **<obligatorio> developmentVendor**: contendrá el nombre de la empresa adjudicataria de la implementación
- **<obligatorio> acronimo**: contendrá el acrónimo del proyecto
- **<obligatorio> group**: contendrá el grupo al que pertenece la aplicación, siguiendo la convención definida. Se debe utilizar para su composición la propiedad de proyecto **project.name**, que contendrá el valor, asignado en el fichero **settings.gradle**, de la propiedad rootProject.name.
- **<obligatorio> name**: contendrá el nombre de la aplicación, siguiendo la convención definida en el punto Identificación de proyectos, aplicaciones y artefactos y la normativa definida. Se debe asignar la propiedad de proyecto project.name, que contendrá el valor, asignado en el fichero **settings.gradle**, de la propiedad rootProject.name.
- **<obligatorio> version**: contendrá la versión de la aplicación, las convenciones definidas en los puntos Identificación de proyectos, aplicaciones y artefactos y Convención de versionado.
- **<obligatorio> encoding**: contendrá el tipo de **encoding** utilizado por el proyecto, por defecto y si no hay motivo para utilizar otro debe ser 'UTF-8'
- **<obligatorio> sourceJdkVersion**: contendrá la versión Java utilizada para la implementación de la aplicación.
- **<obligatorio> targetJdkVersion**: contendrá la versión Java con la que se deberá realizar la compilación.

A continuación, se muestra un ejemplo:

```
//Información del proyecto
ext {
    ...
    projectInfo = [
        description : """Proyecto de ejemplo de la normativa Gradle del DGT""",
        organization : 'Dirección General de Tráfico',
        developmentVendor : 'Altran',
        acronimo : 'PROY',
        group: "es.trafico.${project.name}",
        name : "${project.name}",
        version : '1.0.0-SNAPSHOT',
        encoding : 'UTF-8',
        sourceJdkVersion : '1.8',
```

```
            targetJdkVersion : '1.8'
        ]
        ...
}
```

build.gradle

5.4.2.2 Propiedades de información de la construcción

Todos los proyectos deben seguir para la definición del objeto **ext** las siguientes directrices asociadas a propiedades de información de construcción del proyecto, que serán utilizadas durante la generación de ficheros **Manifest**:

- **<obligatorio>** Todos los proyectos deben incluir, justo después de la sección anterior, la aplicación del fichero **gradle** llamado environmentBuildConfig.gradle, ubicado en la carpeta gradle-env-conf, situada en la raíz del proyecto principal.
- **<obligatorio>** Todos los proyectos deben declarar una propiedad de tipo **map** llamada projectBuildInfo, en la sección de definición del objeto **ext**, dentro del fichero environmentBuildConfig.gradle.
- La propiedad projectBuildInfo debe contener las siguientes propiedades:
  - **<obligatorio>** **buildDate**: contendrá el script que devolverá la hora y fecha de la ejecución (new Date().format("yyyy-MM-dd'T'HH:mm:ss"))
  - **<obligatorio>** **pathJavac**: contendrá la ubicación del compilador Javac, asociada a la versión que se indique en la propiedad projectInfo.targetJdkVersion, utilizando para su configuración la propiedad pathJdkX que se haya incluido en el fichero gradle.properties. (${pathJdkX}/bin/javac)
  - **<obligatorio>** **buildGradleGroovy**: contendrá el script que devolverá la versión de Gradle y Groovy utilizada durante su ejecución ("gradle-${getGradle().getGradleVersion()}, groovy-${GroovySystem.getVersion()}")

Para poder disponer de información adicional en las construcciones que se realicen en entornos de Integración Continua, la sección del objeto ext deberá contener las siguientes propiedades:

- **<obligatorio>** **isCiServer**: contendrá el script que retornará si la ejecución se ha lanzado sobre un servidor de Integración Continua (System.getenv().containsKey("CI_SERVER")). En estos entornos existirá una variable llamada CI_SERVER, haciendo que la propiedad isCiServer tenga el valor true, mientras que en entornos locales tomará el valor false.
- **<obligatorio>** **buildNumber**: contendrá el número de compilación asignado por el entorno de integración continua, para ello se deberá incluir el script que dependiendo del entorno en el que se esté ejecutando el proyecto, la generará o no, asignándole el valor de la variable de entorno BUILD_NUMBER.

```
(if (isCiServer) {
   projectBuildInfo += [
    buildNumber : System.getenv("BUILD_NUMBER")
   ]
  })
```

A continuación, se muestra un ejemplo:

```
//Información del proyecto
ext {
        ...
        projectInfo = [
                ...
        ]
        ...
}
apply from: "gradle-env-conf/environmentBuildConfig.gradle"
```

build.gradle

```
//Información sobre la construcción del proyecto
ext {
        projectBuildInfo = [
                buildDate : new Date().format("yyyy-MM-dd'T'HH:mm:ss"),
                pathJavac : "${pathJdk8}/bin/javac",
                buildGradleGroovy: “gradle-${getGradle().getGradleVersion()}, groovy-
                ${GroovySystem.getVersion()}
        ]

        isCiServer = System.getenv().containsKey("CI_SERVER")

   if (isCiServer) {
                projectBuildInfo += [
                        buildNumber : System.getenv("BUILD_NUMBER")
```

```
            ]
        }
}
```

${ROOT_PROJECT}/gradle-env-conf/environmentBuildConfig.gradle

El fichero environmentBuildConfig.gradle será utilizado para inyectar la aplicación de scripts plugins con los que incluir funcionalidades y automatizaciones requeridas por los entornos de DGT. Por ejemplo se incluirán funcionalidades para la generación de propiedades de proyecto asociadas a versionado con las que poder auditar la versión y el seguimiento de convenciones, o la inyección de propiedades necesarias para integración con la herramienta de análisis de código SonarQube.

A continuación, se muestra un ejemplo:

```
//Información sobre la construcción del proyecto
ext {
        projectBuildInfo = [
                ...
        ]

        isCiServer = System.getenv().containsKey("CI_SERVER")
   ...
}
apply from: "gradle-env-conf/gradleAudit.gradle"
apply from: "gradle-env-conf/versioning.gradle"
apply from: "gradle-env-conf/printProjectInfo.gradle"
...
```

${ROOT_PROJECT}/gradle-env-conf/environmentBuildConfig.gradle

Estas líneas no deben incluirse en el fichero environmentBuildConfig.gradle del proyecto ya que serán incluidas de forma automatizada por el entorno de Integración Continua.

5.4.3 Directrices sobre las coordenadas y descripción del proyecto

Todo proyecto que se ajuste a estas directrices, deberá disponer en su configuración Gradle, de las siguientes propiedades:

- **<obligatorio>** **description**: contendrá una breve descripción del proyecto o módulo, debe componerse con la propiedad del objeto **ext** projectInfo.description (véase el apartado de
- Propiedades de información del proyecto), quedando de la siguiente forma description = projectInfo.description.
- **<obligatorio>** **group**: contendrá el identificador de grupo del proyecto, al que pertenecerán todos los módulos que conformen la aplicación, por lo que deberá ser aplicado en la sección **allprojects**. Para su asignación debe utilizarse la propiedad del objeto **ext** projectInfo.group, quedando de la siguiente forma group = projectInfo.group
- **<obligatorio>** **version**: contendrá la versión del proyecto, y a su vez la de todos los módulos que conformen la aplicación, por lo que deberá ser aplicado en la sección **allprojects**. Para su asignación debe utilizarse la propiedad del objeto ext projectInfo.version, quedando de la siguiente forma version = projectInfo.version

  A continuación, se muestra un ejemplo:

```
//Información del proyecto
ext {
	...
	projectInfo = [
		...
	]
	...
}
apply from: "gradle-env-conf/environmentBuildConfig.gradle"
...

description = projectInfo.description

allprojects {
	group = projectInfo.group
	version = projectInfo.version
	...
}
...
```

build.gradle

5.4.4 Directrices de configuración de repositorios de artefactos

Con el objetivo de realizar una configuración estándar y parametrizada para que todos los proyectos realizados en DGT, utilizando **Gradle** en su configuración, es necesario que todos los proyectos realicen la definición de conexión con el repositorio de artefactos **Artifactory** siguiendo las siguientes directrices obligatorias:

- **<obligatorio>** sección **repositories**: contendrá la configuración de conexión con los repositorios de tipo **maven**, utilizando para ello las propiedades de entorno definidas en el fichero **gradle.properties** creadas para tal efecto, artifactoryDependenciesUrl, artifactoryUser y artifactoryPassword.

Todo proyecto principal debe incorporar dicha configuración dentro de la sección **allproject**, para que todos los módulos dispongan de acceso a los repositorios que necesite.

En caso de que un proyecto necesite acceso a librerías existentes en los repositorios Jcenter o Maven Central, ya que el repositorio central de la DGT dispone de conexión cacheada sobre estos repositorios externos. Por ello el uso de estos repositorios (jcenter, maven2) no será permitido, salvo excepciones que deberán ser solicitadas la Oficina de Calidad (Véase apartado 1.6 Resolución de dudas).

A continuación se muestra un ejemplo:

```
...

allprojects {
    ...
    repositories {
        maven {
            url "${artifactoryDependenciesUrl}"
            credentials {
                username "${artifactoryUser}"
                password "${artifactoryPassword}"
            }
        }
    }
```

```
}
...
```

build.gradle

El ejemplo anterior puede ser copiado y pegado para realizar la configuración en cualquier proyecto ya que la parametrización realizada hace de su diseño sea genérico y estándar.

5.4.5 Directrices de definición de dependencias

Con el objetivo de realizar una configuración legible y estructurada, la configuración y definición de dependencias de un proyecto debe seguir las siguientes directrices de diseño:

- **<obligatorio> dependencies.gradle**: es necesario crear un fichero llamado **dependencies.gradle**, ubicado en la carpeta gradle-project-conf, cuyo contenido incluirá la declaración de las propiedades de tipo map versions y libraries, declaradas dentro de la sección del objeto ext. Dichas propiedades albergarán las coordenadas de las dependencias del proyecto y su versión.
- **<obligatorio> libraries**: contendrá la asignación de todas las coordenadas de las dependencias del proyecto, teniendo cada dependencia un nombre asociado al artifactId de la misma, como por ejemplo libraries.junit. Cada dependencia debe ser generada mediante la utilización del mecanismo de notación de cadena de caracteres, véase https://docs.gradle.org/current/userguide/dependency_types.html#sub:module_dependencies, quedando de la siguiente forma libraries.junit = ” junit:junit:${versions.junit}”
- **<obligatorio> versions**: contendrá las versiones de todas las dependencias, para ello deberá contener un par artifactId y su versión, quedando de la siguiente forma, versions.junit = "4.12". Cada versión será utilizada en la definición de una dependencia en la propiedad libraries.
- **<obligatorio>** sección **dependencies**: la declaración del tipo de dependencia, junto a las coordenadas de las mismas, deberán realizarse en la sección **dependencies** del proyecto, haciendo uso de las propiedades de tipo **map**, libraries y versiones, para lo que con anterioridad se tendrá que aplicar el fichero **gradle** que las contiene.

A continuación, se muestra un ejemplo:

```
//Dependencias del proyecto
...
ext {
    libraries = [:]
    versions = [:]
```

```
}

versions += [
        ...
        junit = "4.12"
        slf4jApi = "1.7.21"
        ...
]

libraries += [
        ...
        junit : "junit:junit:${versions.junit}"
        slf4jApi = "org.slf4j:slf4j-api:${versions.slf4jApi}"
        ...
]
...
```

${ROOT_PROJECT}/gradle-project-conf/dependencies.gradle

```
//dependencias del proyecto
...
apply from: “gradle-project-conf/dependencies.gradle”
dependencies {
        ...
        compile libraries.slf4jApi
        ...
        testCompile libraries.junit
        ...
}
...
```

build.gradle

5.4.6 Directrices de generación de artefactos JAR

A lo largo de esta sección se definirán las bases de diseño con las que realizar la extensión de funcionalidades, y configuración a utilizar durante la generación de artefactos de tipo JAR. El diseño

creado sigue la misma estructura y filosofía utilizada para la configuración de dependencias, es decir, un fichero **gradle** ubicado en la carpeta **gradle-project-conf** y la aplicación de dicho fichero en el **build.gradle**.

5.4.6.1 Directrices de configuración de compilación

Para poder compilar el proyecto con la JDK asociada al código generado y la máquina virtual sobre la que deberá ser ejecutada, es necesario que todo proyecto configure de la siguiente manera la sección **compileJava** y las propiedades de compatibilidad Java:

- **<obligatorio>** **sourceCompatibility**: contendrá la versión Java con la que se ha implementado el código fuente, debe componerse a partir de la propiedad del objeto ext destinada a tal efecto, quedando de la siguiente forma sourceCompatibility = projectInfo.sourceJdkVersion
- **<obligatorio>** **targetCompatibility**: contendrá la versión Java con la que se ha implementado el código fuente, debe componerse a partir de la propiedad del objeto ext destinada a tal efecto, quedando de la siguiente forma targetCompatibility = projectInfo.targetJdkVersion
- **<obligatorio>** sección **compileJava**: contendrá la activación de un fork Java, options.fork = true, la asignación del acceso a la herramienta javac, options.forkOptions.executable = projectBuildInfo.pathJavac, y la configuración del conjunto de caracteres utilizado para la implementación del proyecto a través de la propiedad del objeto ext creada para dicha tarea , quedando de la siguiente forma, options.encoding = projectInfo.encoding

A continuación, se muestra un ejemplo:

```
...
sourceCompatibility = projectInfo.sourceJdkVersion
targetCompatibility = projectInfo.targetJdkVersion

compileJava {
	options.fork = true
	options.forkOptions.executable = projectBuildInfo.pathJavac
	options.encoding = projectInfo.encoding
}
...
```

build.gradle

5.4.6.2 Directrices de configuración de fichero Manifest

Cada artefacto de tipo **JAR** generado por los proyectos desarrollado en DGT, deberán contener un fichero **MANIFEST.MF**, cuyo contenido será generado automáticamente durante el ensamblado del proyecto. La información a incluir dentro del proceso automatizado deberá indicarse en un fichero **gradle** llamado **manifest.gradle**, ubicado dentro de la carpeta **gradle-project-conf** y cuyo contenido debe seguir las siguientes directrices:

- **<obligatorio>** **projectManifest**: esta propiedad debe definirse dentro del objeto ext, con el objetivo de poder ser utilizado por todas las tareas que generen artefactos durante la construcción del proyecto. Contendrán un objeto de tipo **manifest** y cada uno de los atributos que se definen a continuación:
- **<obligatorio>** **Created-By**: contendrá las versiones de **gradle** y **groovy** con las que se haya construido el proyecto, para ello se deberá asignar el valor de la propiedad projectBuildInfo.buildGradleGroovy.
- **<obligatorio>** **Source-Compatibility**: contendrá la versión Java del código fuente, para ello se deberá asignar el valor de la propiedad projectInfo.sourceJdkVersion.
- **<obligatorio>** **Target-Compatibility**: contendrá la versión Java sobre la que debe ejecutarse, para ello se deberá asignar el valor de la propiedad projectInfo.targetJdkVersion.
- **<obligatorio>** **Built-By**: contendrá la organización propietaria del artefacto, para ello se deberá asignar el valor de la propiedad projectInfo.organization.
- **<obligatorio>** **Build-Time**: Contendrá la fecha y hora en la que se ha construido el artefacto, para ello se deberá asignar el valor de la propiedad projectBuildInfo.buildDate.
- **<obligatorio>** **Specification-Title**: contendrá la información necesaria para identificar la especificación sobre la que se ha diseñado el artefacto, para ello se deberá asignar el valor de las propiedades projectInfo.group, projectInfo.name, junto a las propiedades name y description del proyecto.
- **<obligatorio>** **Specification-Version**: contendrá la versión de la especificación sobre la que se ha diseñado el artefacto, para ello se deberá asignar el valor de la propiedad projectInfo.version.
- **<obligatorio>** **Specification-Vendor**: contendrá la empresa encargada de la definición de la especificación con la que se ha diseñado el artefacto, para ello se deberá asignar el valor de la propiedad projectInfo.developmentVendor.
- **<obligatorio>** **Implementation-Title**: contendrá la información necesaria para identificar la implementación de la especificación sobre la que se ha diseñado el artefacto, para ello se deberá asignar el valor de las propiedades projectInfo.acronimo, projectInfo.group y projectInfo.name, junto a las propiedades name y description del proyecto.

- **<obligatorio> Implementation-Version**: contendrá la versión de la implementación de la especificación sobre la que se ha diseñado el artefacto, para ello se deberá asignar el valor de la propiedad projectInfo.version.
- **<obligatorio> Implementation-Vendor**: contendrá la empresa encargada de la implementación de la especificación con la que se ha diseñado el artefacto, para ello se deberá asignar el valor de la propiedad projectInfo.developmentVendor.
- **<obligatorio> Build-Number**: contendrá el número de construcción generado por el entorno de Integración Continua, esta propiedad sólo se incluirá en caso de que la ejecución se realice dentro de dicho contexto, por lo que estará asociada a la propiedad extra isCiServer, para ello se deberá incluir el siguiente código:

```
If (isCiServer) {
        projectManifest.getAttributes().put('Build-Number', '${projectBuildInfo.buildNumber }')
}
```

- La aplicación del fichero **manifest.gradle**, se realizará justo antes de ser usado en la sección **jar**, y estará disponible a partir de ese momento para la generación de los artefactos **javadoc** y **sources**, cuya normativa se definirá en los siguientes puntos.

  A continuación, se muestra un ejemplo:

```
//Información a incluir en el fichero manifest
ext {
  projectManifest = manifest {
     attributes('Created-By' : "${projectBuildInfo.buildGradleGroovy}",
              'Source-Compatibility' : "${projectInfo.sourceJdkVersion}",
              'Target-Compatibility' : "${projectInfo.targetJdkVersion}",
              'Built-By' : "${projectInfo.organization}",
              'Build-Time' : "${projectBuildInfo.buildDate}",
              'Specification-Title' :  "${projectInfo.group} ::
                                  ${projectInfo.name} ::
                                   $name :: $description",
              'Specification-Version' : "${projectInfo.version}",
              'Specification-Vendor' : "${projectInfo.developmentVendor}",
            'Implementation-Title' : "${projectInfo.group} ::
                                  ${projectInfo.name} ::
                                   $name :: $description",
              'Implementation-Version' : "${projectInfo.version}",
```

```
            'Implementation-Vendor' : "${projectInfo.developmentVendor}"
            )
  }

  if (isCiServer) {
    projectManifest.getAttributes().put('Build-Number',
                                        "${projectBuildInfo.buildNumber}")
  }
}
```

${ROOT_PROJECT}/gradle-project-conf/manifest.gradle

```
//Configuración de construcción
...
apply from: “gradle-project-conf/manifest.gradle”
jar {
        ...
        manifest = project.manifest {
    from projectManifest
  }
        ...
}
...
```

build.gradle

El ejemplo anterior puede ser copiado y pegado para realizar la configuración en cualquier proyecto ya que la parametrización realizada hace de su diseño sea genérico y estándar.

5.4.6.3 Directrices de generación de artefactos javadoc

Para aumentar la información de los artefactos generados y que su distribución se ajuste a los estándares de proyectos Java, todos proyecto, que durante su construcción, genere ficheros de tipo **JAR**, deberá generar su artefacto **javadoc** asociado. Para ello es necesario ejecutar con anterioridad la generación de documentación del código que lo compone, mediante la configuración y ejecución del **task generateJavadoc**, justo después se deberá configurar y ejecutar la tarea que creará el artefacto de tipo JAR con clasificado como **javadoc**, llamada **generateJavadocJar**. Estas tareas deben incluirse

en un fichero **gradle** llamado **generateJavadocJar.gradle**, ubicándolo en la carpeta **gradle-project-conf**.

Todas las configuraciones y definición de tareas necesarias para la generación de artefactos de tipo **javadoc**, se detallan a continuación como directrices que deberán seguir aquellos proyectos que lo necesiten:

- **[opcional]** tarea **generateJavadoc**: esta tarea deberá contener una descripción, asignado un grupo del ciclo de vida del **plugin** Java, la configuración necesaria para ejecutar la generación del **javadoc** y la asignación de una propiedad con la que poder ser saltada. Esta tarea deberá ser asignada al grupo '**Documentation'** y ser asociada a la propiedad creada con el prefijo **skip** seguido del nombre de la tarea.
- **[opcional]** tarea **generateJavadocJar**: esta tarea deberá contener una descripción, asignado un grupo del ciclo de vida del **plugin** Java, la configuración necesaria para crear el artefacto y la asignación de una propiedad con la que poder ser saltada. Esta tarea deberá ser asignada al grupo '**Build'** y ser asociada a la propiedad creada con el prefijo **skip** seguido del nombre de la tarea.
- **[opcional]** **archivado** de artefactos: el artefacto generado por la tarea **generateJavadocJar**, deberá ser archivado utilizando para ello su inclusión en la sección **artifacts**.

A continuación, se muestra el contenido del fichero **generateJavadocJar.gradle**, que podrá ser copiado y pegado en cualquier proyecto debido a su parametrización y diseño genérico.

```
task generateJavadoc (type : Javadoc) {
  group = 'Documentation'
  description = "[Create-by: ${projectInfo.name}] - Generates Javadoc API
                    documentation for the main source code"
  source = sourceSets.main.allJava
  classpath = sourceSets.main.compileClasspath
  failOnError = false
}
generateJavadoc.onlyIf { !project.hasProperty('skipGenerateJavadocJar') }

task generateJavadocJar(type: Jar, dependsOn: generateJavadoc) {
  group = 'Build'
  description = "[Create-by: ${projectInfo.name}] - Assembles a jar archive containing
```

```
                    the javadoc."
  classifier = 'javadoc'
  from generateJavadoc.destinationDir
  manifest = project.manifest {
    from projectManifest
  }
}
generateJavadocJar.onlyIf { !project.hasProperty('skipGenerateJavadocJar') }

artifacts {
  archives generateJavadocJar
}
```

${ROOT_PROJECT}/gradle-project-conf/generateJavadocJar.gradle

```
//Configuración de construcción

apply from: “gradle-project-conf/manifest.gradle”
jar {
        ...
        manifest = project.manifest {
    from projectManifest
  }
        ...
}
...
apply from: “gradle-project-conf/generateJavadocJar.gradle”
...
```

build.gradle

5.4.6.4 Directrices de generación de artefactos sources

Para aumentar la información de los artefactos generados y que su distribución se ajuste a los estándares de proyectos Java, todos proyecto, que durante su construcción, genere ficheros de tipo JAR, deberá generar su artefacto **sources** asociado. Para ello es configurar y ejecutar la tarea que creará el artefacto de tipo JAR con clasificado como **sources**, llamada **generateSourcesJar**. Esta tarea

debe incluirse en un fichero **gradle** llamado **generateSourcesJar.gradle**, ubicándolo en la carpeta **gradle-project-conf**.

La configuración y definición de esta tarea necesaria para la generación de artefactos de tipo **sources**, se detalla a continuación como directriz que deberán seguir aquellos proyectos que lo necesiten:

- **[opcional]** tarea **generateSourcesJar**: esta tarea deberá contener una descripción, asignado un grupo del ciclo de vida del **plugin** Java, la configuración necesaria para crear el artefacto y la asignación de una propiedad con la que poder ser saltada. Esta tarea deberá ser asignada al grupo '**Build'** y ser asociada a la propiedad creada con el prefijo **skip** seguido del nombre de la tarea.
- **[opcional]** **archivado** de artefactos: el artefacto generado por la tarea **generateSourcesJar**, deberá ser archivado utilizando para ello su inclusión en la sección **artifacts**.

A continuación, se muestra el contenido del fichero **generateSourcesJar.gradle**, que podrá ser copiado y pegado en cualquier proyecto debido a su parametrización y diseño genérico.

```
task generateSourcesJar(type: Jar) {
  group = 'Build'
  description = "[Create-by: ${projectInfo.name}] - Assembles a jar archive containing
                    the sources."
  classifier = 'sources'
  from sourceSets.main.allSource
  manifest = project.manifest {
    from projectManifest
  }
}
generateSourcesJar.onlyIf { !project.hasProperty('skipGenerateSourcesJar') }

artifacts {
  archives generateSourcesJar
}
```

${ROOT_PROJECT}/gradle-project-conf/generateSourcesJar.gradle

```
//Configuración de construcción

apply from: “gradle-project-conf/manifest.gradle”
jar {
        ...
        manifest = project.manifest {
    from projectManifest
  }
        ...
}
...
apply from: “gradle-project-conf/generateJavadocJar.gradle”
apply from: “gradle-project-conf/generateSourcesJar.gradle”
...
```

build.gradle

5.4.6.5 Directrices de publicación de artefactos en Artifactory

Para poder publicar los artefactos generados, todo proyecto, que durante su construcción, genere ficheros de tipo JAR, y tengan que ser publicados en Artifactory deberá configurar la tarea que especificará los parámetros de publicación, llamada publishArtifactoryJar. Esta tarea debe incluirse en un fichero **gradle** llamado **publishArtifactoryJar.gradle**, ubicándolo en la carpeta **gradle-project-conf**.

A continuación, se muestra el contenido del fichero **publishArtifactoryJar.gradle**, que podrá ser copiado y pegado en cualquier proyecto debido a su parametrización y diseño genérico:

```
apply plugin: 'maven-publish'

publishing {
    publications {
        mavenJava(MavenPublication) {
            from components.java
            artifact generateSourcesJar
            artifact generateJavadocJar
        }
```

```
    }
    repositories {
        maven {
            url "${artifactoryPublishUrl}"
            credentials {
                username "${artifactoryPublishUser}"
                password "${artifactoryPublishPassword}"
            }
        }
    }
}
```

${ROOT_PROJECT}/gradle-project-conf/publishArtifactoryJar.gradle

```
        ...
}
...
apply from: “gradle-project-conf/generateJavadocJar.gradle”
apply from: “gradle-project-conf/generateSourcesJar.gradle”
apply from: “gradle-project-conf/publishArtifactoryJar.gradle”
...
```

build.gradle

En caso de que algún proyecto necesite poder establecer la publicación o no de los artefactos en base a una propiedad interna, lo puede realizar de esta forma:

Definiendo en el fichero gradle.properties una propiedad con la que “activar/desactivar” la publicación de artefactos:

```
…
artifactoryPublishEnabled=true/false
…
```

Incluyendo el siguiente código a continuación del bloque publishing en el fichero publishArtifactoryJar.gradle

```
apply plugin: 'maven-publish'

publishing {
…
}

model {
	tasks.generatePomFileForMavenJavaPublication {
		enabled = "${artifactoryPublishEnabled}".toBoolean()
	}
	tasks.publishMavenJavaPublicationToMavenRepository {
		enabled = "${artifactoryPublishEnabled}".toBoolean()
	}
}

tasks.publish.enabled = "${artifactoryPublishEnabled}".toBoolean()
```

${ROOT_PROJECT}/gradle-project-conf/publishArtifactoryJar.gradle

5.4.6.6 Directrices de firma de artefactos

Para aquellos proyectos que tengan como requisito la firma de artefactos de tipo **JAR**, se ha definido el diseño de un fichero llamado **signJar.gradle**, que contiene las tareas de firma y verificado, asociadas al ciclo de vida de construcción del **plugin Java**, por lo que cualquier proyecto que necesite firmar artefactos, únicamente deberá incluir el fichero en la carpeta **gradle-project-conf** y realizar la aplicación del mismo en su fichero **build.gradle**.

La configuración y definición de estas tareas necesarias para la firma de artefactos, se detalla a continuación como directriz que deberán seguir aquellos proyectos que lo necesiten:

- **[opcional]** tarea **signJar**: esta tarea deberá contener una descripción, asignado un grupo del ciclo de vida del **plugin** Java, la configuración necesaria para ejecutar el firmado y la asignación de una propiedad con la que poder ser saltada. Esta tarea deberá ser asignada al grupo '**Build'** y ser asociada a la propiedad creada con el prefijo **skip** seguido del nombre de la tarea.

- **[opcional]** tarea **verifySignedJar**: esta tarea deberá contener una descripción, asignado un grupo del ciclo de vida del **plugin** Java, la configuración necesaria para ejecutar el firmado y la asignación de una propiedad con la que poder ser saltada. Esta tarea deberá ser asignada al grupo '**Build'** y ser asociada a la propiedad creada con el prefijo **skip** seguido del nombre de la tarea.
- **[opcional]** asociar a la tarea **build**: la tarea **verifySignedJar** debe ser asociada a la ejecución de la tarea **build** para desencadenar la ejecución por dependencia de la firma y verificado durante la construcción del artefacto.

A continuación, se muestra el contenido del fichero **signJar.gradle**, que podrá ser copiado y pegado en cualquier proyecto debido a su parametrización y diseño genérico.

```
task signJar (type: Exec) {
  group = 'Build'
  description = "[Create-by: ${projectInfo.name}] - Sign a jar archive."
  workingDir jarsignerJdkDir
  executable = jarsigner
  args = '-keystore', jarsignerkeyStore
  args = '-storepass, jarsignerStorePass
  args jar.archivePath
  args jarsignerAlias
}
signJar.onlyIf { !project.hasProperty('skipSignJar') }

task verifySignedJar (type: Exec, dependsOn: signJar) {
  group = 'Build'
  description = "[Create-by: ${projectInfo.name}] – Verify a jar archive."
  workingDir jarsignerJdkDir
  executable = jarsigner
  args = '-verify
  args jar.archivePath
}
signJar.onlyIf { !project.hasProperty('skipVerifySignedJar') }

build.dependsOn verifySignedJar
```

${ROOT_PROJECT}/gradle-project-conf/generateSourcesJar.gradle

```
//Configuración de construcción

apply from: “gradle-project-conf/manifest.gradle”
jar {
        ...
        manifest = project.manifest {
    from projectManifest
  }
        ...
}
...
apply from: “gradle-project-conf/generateJavadocJar.gradle”
apply from: “gradle-project-conf/generateSourcesJar.gradle”
...
apply from: “gradle-project-conf/signJar.gradle”
```

build.gradle

5.4.6.7 Directrices sobre la nomenclatura de artefactos generados

Todos los artefactos generados de tipo **JAR** deben seguir la convención de nombres, definida por defecto en la documentación de ensamblado **Jar**, asociado al **plugin Java**. A continuación se indica la notación que deberán cumplir todos los artefactos generados de este tipo:

- **<obligatorio>** Esta notación debe respetarse de forma obligatoria: [baseName]-[appendix]-[version]-[classifier].[extension]
- **<obligatorio>** Gradle utiliza propiedades de proyecto para formar el nombre de artefactos, ${baseName}-${appendix}-${version}-${classifier}.${extension}, que deberá respetarse siempre, generando los artefactos a partir de las propiedades y no definiendo un literal en la propiedad **archiveName**.

Para obtener más información recomendamos la lectura de la documentación oficial y de la sección dedicada a este tema en la URL

https://docs.gradle.org/current/dsl/org.gradle.api.tasks.bundling.Jar.html#org.gradle.api.tasks.bundling.Jar:archiveName

5.4.6.8 Directrices sobre la nomenclatura de artefactos generados (ejb)

Todos los artefactos generados de tipo ejb deben seguir las directrices de nombrado diseñadas por DGT, ya que por requisitos internos, el nombre de los artefactos generados no puede incluir el versionado. A continuación se indica la notación que deberán cumplir todos los artefactos generados de este tipo:

- **<obligatorio>** Esta notación debe respetarse de forma obligatoria: [name].[extension]
- **<obligatorio>** Gradle utiliza propiedades de proyecto para formar el nombre de artefactos, ${baseName}-${appendix}-${version}-${classifier}.${extension}, por lo que será necesario modificar la asignación del nombre que realiza por defecto. Para ello se deberá asignar el valor deseado a la propiedad archiveName, siguiendo la notación [name].[extension], dónde [extension] se corresponderá con el valor por defecto de la propiedad jar.extension, y [name] se corresponderá con el nombre que llevará el artefacto generado, pudiendo utilizar para su composición, la propiedad baseName en caso de que el nombre del proyecto/módulo coincida con el nombre del artefacto a generar (jar.archiveName = "${baseName}.${jar.extension}"), o un literal en caso de que el nombre del proyecto/módulo no deba ser el utilizado para su generación (jar.archiveName = "nombre-jar.${extension}")

5.4.7 Directrices de generación de artefactos WAR

A lo largo de esta sección se definirán las bases de diseño con las que realizar la extensión de funcionalidades, y configuración a utilizar durante la generación de artefactos de tipo **WAR**. El diseño creado sigue la misma estructura y filosofía utilizada para la configuración de dependencias, es decir, un fichero **gradle** ubicado en la carpeta **gradle-project-conf** y la aplicación de dicho fichero en el **build.gradle**.

5.4.7.1 Directrices de configuración de fichero Manifest

Cada artefacto de tipo **WAR** generado por los proyectos desarrollado en DGT, deberán contener un fichero **MANIFEST.MF**, cuyo contenido será generado automáticamente durante el

ensamblado del proyecto. La información a incluir dentro del proceso automatizado deberá indicarse en un fichero gradle llamado **manifest.gradle**, ubicado dentro de la carpeta **gradle-project-conf** y cuyo **contenido** debe seguir las siguientes directrices:

- **<obligatorio> projectManifest**: esta propiedad debe definirse dentro del objeto ext, con el objetivo de poder ser utilizado por todas las tareas que generen artefactos durante la construcción del proyecto. Contendrán un objeto de tipo **manifest** y cada uno de los atributos que se definen a continuación:
  - **<obligatorio> Created-By**: contendrá las versiones de **gradle** y **groovy** con las que se haya construido el proyecto, para ello se deberá asignar el valor de la propiedad projectBuildInfo.buildGradleGroovy.
  - **<obligatorio> Source-Compatibility**: contendrá la versión Java del código fuente, para ello se deberá asignar el valor de la propiedad projectInfo.sourceJdkVersion.
  - **<obligatorio> Target-Compatibility**: contendrá la versión Java sobre la que debe ejecutarse, para ello se deberá asignar el valor de la propiedad projectInfo.targetJdkVersion.
  - **<obligatorio> Built-By**: contendrá la organización propietaria del artefacto, para ello se deberá asignar el valor de la propiedad projectInfo.organization.
  - **<obligatorio> Build-Time**: Contendrá la fecha y hora en la que se ha construido el artefacto, para ello se deberá asignar el valor de la propiedad projectBuildInfo.buildDate.
  - **<obligatorio> Specification-Title: contendrá la información necesaria para identificar la especificación sobre** la que se ha diseñado el artefacto, para ello se deberá asignar el valor de las propiedades projectInfo.group, projectInfo.name, junto a las propiedades name y description del proyecto.
  - **<obligatorio> Specification-Version**: contendrá la versión de la especificación sobre la que se ha diseñado el artefacto, para ello se deberá asignar el valor de la propiedad projectInfo.version.
  - **<obligatorio> Specification-Vendor**: contendrá la empresa encargada de la definición de la especificación con la que se ha diseñado el artefacto, para ello se deberá asignar el valor de la propiedad projectInfo.developmentVendor.
  - **<obligatorio> Implementation-Title**: contendrá la información necesaria para identificar la implementación de la especificación sobre la que se ha diseñado el artefacto, para ello se deberá asignar el valor de las propiedades projectInfo.acronimo, projectInfo.group, projectInfo.name, junto a las propiedades name y description del proyecto.

- **<obligatorio> Implementation-Version**: contendrá la versión de la implementación de la especificación sobre la que se ha diseñado el artefacto, para ello se deberá asignar el valor de la propiedad projectInfo.version.
- **<obligatorio> Implementation-Vendor**: contendrá la empresa encargada de la implementación de la especificación con la que se ha diseñado el artefacto, para ello se deberá asignar el valor de la propiedad projectInfo.developmentVendor.
- **<obligatorio> Build-Number**: contendrá el número de construcción generado por el entorno de Integración Continua, esta propiedad sólo se incluirá en caso de que la ejecución se realice dentro de dicho contexto, por lo que estará asociada a la propiedad extra isCiServer, para ello se deberá incluir el siguiente código:
  - If (isCiServer) {projectManifest.getAttributes().put('Build-Number', '${projectBuildInfo.buildNumber }')}

La aplicación del fichero **manifest.gradle**, se realizará justo antes de ser usado en la sección **war**.

A continuación, se muestra un ejemplo:

```
//Información a incluir en el fichero manifest
ext {
  projectManifest = manifest {
    attributes('Created-By' : "${projectBuildInfo.buildGradleGroovy}",
            'Source-Compatibility' : "${projectInfo.sourceJdkVersion}",
            'Target-Compatibility' : "${projectInfo.targetJdkVersion}",
            'Built-By' : "${projectInfo.organization}",
            'Build-Time' : "${projectBuildInfo.buildDate}",
            'Specification-Title' : "${projectInfo.group} ::
                                     ${projectInfo.name} ::
                                     $name :: $description",
            'Specification-Version' : "${projectInfo.version}",
            'Specification-Vendor' : "${projectInfo.developmentVendor}",
            'Implementation-Title' : "${projectInfo.group} ::
                                      ${projectInfo.name} ::
                                      $name :: $description",
            'Implementation-Version' : "${projectInfo.version}",
            'Implementation-Vendor' : "${projectInfo.developmentVendor}"
            )
```

```
  }

  if (isCiServer) {
    projectManifest.getAttributes().put('Build-Number',
                                        "${projectBuildInfo.buildNumber}")
  }
}
```

${ROOT_PROJECT}/gradle-project-conf/manifest.gradle

```
//Configuración de construcción
...
war.archiveName = "nombre-war.${war.extension }"

apply from: “gradle-project-conf/manifest.gradle”
war {
        ...
        manifest = project.manifest {
                        from projectManifest
                }
        ...
}
...
```

build.gradle

El ejemplo anterior puede ser copiado y pegado para realizar la configuración en cualquier proyecto ya que la parametrización realizada hace de su diseño sea genérico y estándar.

5.4.7.2 Directrices sobre la nomenclatura de artefactos generados

Todos los artefactos generados de tipo WAR deben seguir las directrices de nombrado diseñadas por DGT, ya que por requisitos internos, el nombre de los artefactos generados no puede incluir el versionado. A continuación se indica la notación que deberán cumplir todos los artefactos generados de este tipo:

- **<obligatorio>** Esta notación debe respetarse de forma obligatoria: [name].[extension]

- **<obligatorio>** Gradle utiliza propiedades de proyecto para formar el nombre de artefactos, ${baseName}-${appendix}-${version}-${classifier}.${extension}, por lo que será necesario modificar la asignación del nombre que realiza por defecto. Para ello se deberá asignar el valor deseado a la propiedad archiveName, siguiendo la notación [name].[extension], dónde [extension] se corresponderá con el valor por defecto de la propiedad war.extension, y [name] se corresponderá con el nombre que llevará el artefacto generado, pudiendo utilizar para su composición, la propiedad baseName en caso de que el nombre del proyecto/módulo coincida con el nombre del artefacto a generar (war.archiveName = "${baseName}.${war.extension}"), o un literal en caso de que el nombre del proyecto/módulo no deba ser el utilizado para su generación (war.archiveName = "nombre-war.${extension}")

Para obtener más información recomendamos la lectura de la documentación oficial y de la sección dedicada a este tema en la URL https://docs.gradle.org/current/dsl/org.gradle.api.tasks.bundling.War.html#org.gradle.api.tasks.bundling.War:archiveName

5.4.7.3 Directrices de generación de fichero de estáticos

Aquellos proyectos que contengan contenido estático destinado al despliegue fuera del artefacto war, deberán generar un fichero de tipo comprimido mediante la funcionalidad prediseñada **generateStaticFilesZip**. Esta tarea debe incluirse en un fichero **gradle** llamado **generatStaticFilesZip.gradle**, ubicándolo en la carpeta **gradle-project-conf**.

La configuración y definición de esta tarea necesaria para la generación de ficheros de tipo **zip**, se detalla a continuación como directriz que deberán seguir aquellos proyectos que lo necesiten:

- **[opcional]** tarea **generateStaticFilesZip**: esta tarea deberá contener una descripción, asignado un grupo del ciclo de vida del **plugin** Java, la configuración necesaria para crear el comprimido y la asignación de una propiedad con la que poder ser saltada. Esta tarea deberá ser asignada al grupo '**Build'** y ser asociada a la propiedad creada con el prefijo **skip** seguido del nombre de la tarea.
- **[opcional]** **archivado** de artefactos: el artefacto generado por la tarea **generateStaticFilesZip**, deberá ser archivado utilizando para ello su inclusión en la sección **artifacts**.

A continuación, se muestra el contenido del fichero **generateStaticFilesZip.gradle**, que podrá ser copiado y pegado en cualquier proyecto debido a su parametrización y diseño genérico, debiendo configurarse tanto la ruta de la carpeta raíz, como la ubicación específica de los ficheros a incluir por medio de los parámetros **from** e **include** respectivamente.

```
task generateStaticFilesZip (type: Zip) {
  group = 'Build'
  description = "[Create-by: ${projectInfo.name}] - Generate static web content ZIP file."
  archiveName = "static-${war.baseName}.zip"
  from "src/main/webapp"
  include "images/*.*"
  include "styles/**/*.*"
}
generateStaticFilesZip.onlyIf { !project.hasProperty('skipGenerateStaticFilesZip') }

artifacts {
  archives generateStaticFilesZip
}
```

${ROOT_PROJECT}/gradle-project-conf/generateStaticFilesZip.gradle

```
//Aplicación de plugins
plugins {
        ...
     id ‘war’
     id ‘java
        ...
}

//Configuración de construcción
apply from: “gradle-project-conf/manifest.gradle”
war {
        ...
    exclude("images")
    exclude("styles")
        ...
```

```
}
...
apply from: “gradle-project-conf/generateStaticFilesZip.gradle”
...
```

build.gradle

Como se puede ver en el ejemplo, aquellos ficheros estáticos incluidos en el archivo zip a generar, deberán ser excluidos durante la generación del artefacto war.

5.4.7.4 Directrices de gestión de artefactos generados

Aquellos proyectos que generen ficheros distribuibles junto al artefacto de tipo war, como por ejemplo ficheros de distribución de estáticos mediante la tarea **generateStaticFilesZip**, incluirán en su nivel de dependencia (*configurations*) “archives” todos aquellos ficheros que generen, con el objetivo de disponer del conjunto completo de artefactos generados por el proyecto.

Este mecanismo de gestión de artefactos que posee Gradle, mediante niveles de dependencia, puede provocar una configuración errónea si no se aplica correctamente, como por ejemplo la definición de dependencias entre proyectos de tipo ear y war, pudiendo incluirse artefactos que no deban ser desplegados de forma interna en el artefacto ear. Por ello, es necesario que todos los proyectos de tipo war, que generen artefactos de distribución de artefactos, sigan las siguientes directrices para que la gestión de artefactos y dependencias no causen comportamientos inadecuados:

- **[opcional]** el nivel de dependencias (*configurations*) “warArchive”, debe ser utilizada para disponer en exclusiva del artefacto desplegable de tipo war, por ejemplo acro-dgt-ws.war, para ello únicamente la generación de artefacto de tipo war debe ser incluido en el nivel “warArchive”.

A continuación, se muestra un ejemplo de configuración con el que incluir los artefactos war en el nivel de dependencia “warArchive”.

```
//Aplicación de plugins
plugins {
```

```
        ...
      id 'war'
      id 'java
        ...
}

//Niveles de dependencia
configurations {
   warArchive
}

//Configuración de construcción
apply from: "gradle-project-conf/manifest.gradle"
war {
        ...
    exclude("images")
    exclude("styles")
        ...
}
artifacts {
   warArchive war
}
...
apply from: "gradle-project-conf/generateStaticFilesZip.gradle"
...
```

build.gradle

Como se puede ver en el ejemplo, aquellos ficheros generados de tipo war serán incluidos en el nivel de dependencia "warArchive", pudiendo ser utilizado para definir la dependencia en proyectos de tipo ear.

5.4.8 Directrices de generación de artefactos EAR

5.4.8.1 Directrices de configuración de fichero Manifest

Cada artefacto de tipo **EAR** generado por los proyectos desarrollado en DGT, deberán contener un fichero **MANIFEST.MF**, cuyo contenido será generado automáticamente durante el ensamblado del proyecto. La información a incluir dentro del proceso automatizado deberá indicarse en un fichero gradle llamado **manifest.gradle**, ubicado dentro de la carpeta **gradle-project-conf** y cuyo **contenido** debe seguir las siguientes directrices:

- **<obligatorio> projectManifest**: esta propiedad debe definirse dentro del objeto ext, con el objetivo de poder ser utilizado por todas las tareas que generen artefactos durante la construcción del proyecto. Contendrán un objeto de tipo **manifest** y cada uno de los atributos que se definen a continuación:
  - **<obligatorio> Created-By**: contendrá las versiones de **gradle** y **groovy** con las que se haya construido el proyecto, para ello se deberá asignar el valor de la propiedad projectBuildInfo.buildGradleGroovy.
  - **<obligatorio> Source-Compatibility**: contendrá la versión Java del código fuente, para ello se deberá asignar el valor de la propiedad projectInfo.sourceJdkVersion.
  - **<obligatorio> Target-Compatibility**: contendrá la versión Java sobre la que debe ejecutarse, para ello se deberá asignar el valor de la propiedad projectInfo.targetJdkVersion.
  - **<obligatorio> Built-By**: contendrá la organización propietaria del artefacto, para ello se deberá asignar el valor de la propiedad projectInfo.organization.
  - **<obligatorio> Build-Time**: Contendrá la fecha y hora en la que se ha construido el artefacto, para ello se deberá asignar el valor de la propiedad projectBuildInfo.buildDate.
  - **<obligatorio> Specification-Title: contendrá la información necesaria para identificar la especificación sobre** la que se ha diseñado el artefacto, para ello se deberá asignar el valor de las propiedades projectInfo.group, projectInfo.name, junto a las propiedades name y description del proyecto.
  - **<obligatorio> Specification-Version**: contendrá la versión de la especificación sobre la que se ha diseñado el artefacto, para ello se deberá asignar el valor de la propiedad projectInfo.version.

- **<obligatorio> Specification-Vendor**: contendrá la empresa encargada de la definición de la especificación con la que se ha diseñado el artefacto, para ello se deberá asignar el valor de la propiedad projectInfo.developmentVendor.
- **<obligatorio> Implementation-Title**: contendrá la información necesaria para identificar la implementación de la especificación sobre la que se ha diseñado el artefacto, para ello se deberá asignar el valor de las propiedades projectInfo.acronimo, projectInfo.group, projectInfo.name, junto a las propiedades name y description del proyecto.
- **<obligatorio> Implementation-Version**: contendrá la versión de la implementación de la especificación sobre la que se ha diseñado el artefacto, para ello se deberá asignar el valor de la propiedad projectInfo.version.
- **<obligatorio> Implementation-Vendor**: contendrá la empresa encargada de la implementación de la especificación con la que se ha diseñado el artefacto, para ello se deberá asignar el valor de la propiedad projectInfo.developmentVendor.
- **<obligatorio> Build-Number**: contendrá el número de construcción generado por el entorno de Integración Continua, esta propiedad sólo se incluirá en caso de que la ejecución se realice dentro de dicho contexto, por lo que estará asociada a la propiedad extra isCiServer, para ello se deberá incluir el siguiente código:
  - If (isCiServer) {projectManifest.getAttributes().put('Build-Number', '${projectBuildInfo.buildNumber }')}

La aplicación del fichero **manifest.gradle**, se realizará justo antes de ser usado en la sección **ear**.

A continuación, se muestra un ejemplo:

```
//Información a incluir en el fichero manifest
ext {
  projectManifest = manifest {
    attributes('Created-By' : "${projectBuildInfo.buildGradleGroovy}",
               'Source-Compatibility' : "${projectInfo.sourceJdkVersion}",
               'Target-Compatibility' : "${projectInfo.targetJdkVersion}",
               'Built-By' : "${projectInfo.organization}",
               'Build-Time' : "${projectBuildInfo.buildDate}",
             'Specification-Title' : "${projectInfo.group} ::
                                      ${projectInfo.name} ::
                                       $name :: $description",
```

```
                'Specification-Version' : "${projectInfo.version}",
                'Specification-Vendor' : "${projectInfo.developmentVendor}",
              'Implementation-Title' : "${projectInfo.group} ::
                                  ${projectInfo.name} ::
                                   $name :: $description",
                'Implementation-Version' : "${projectInfo.version}",
                'Implementation-Vendor' : "${projectInfo.developmentVendor}"
                )
  }

  if (isCiServer) {
    projectManifest.getAttributes().put('Build-Number',
                                  "${projectBuildInfo.buildNumber}")
  }
}
```

${ROOT_PROJECT}/gradle-project-conf/manifest.gradle

```
//Configuración de construcción
…
ear {
        ...
        manifest = project.manifest {
    from projectManifest
  }
        ...
}
...
```

build.gradle

El ejemplo anterior puede ser copiado y pegado para realizar la configuración en cualquier proyecto ya que la parametrización realizada hace de su diseño sea genérico y estándar.

5.4.8.2 Directrices sobre la nomenclatura de artefactos generados

Todos los artefactos generados de tipo **EAR** deben seguir la convención de nombres, definida por defecto en la documentación de ensamblado Jar, asociado al plugin Java, ya que el plugin Ear

hereda esta configuración de este plugin. A continuación se indica la notación que deberán cumplir todos los artefactos generados de este tipo:

- **<obligatorio>** Esta notación debe respetarse de forma obligatoria: [baseName]-[appendix]-[version]-[classifier].[extension]
- **<obligatorio>** Gradle utiliza propiedades de proyecto para formar el nombre de artefactos, ${baseName}-${appendix}-${version}-${classifier}.${extension}, que deberá respetarse siempre, generando los artefactos a partir de las propiedades y no definiendo un literal en la propiedad **archiveName**.

Para obtener más información recomendamos la lectura de la documentación oficial y de la sección dedicada a este tema en la URL https://docs.gradle.org/current/dsl/org.gradle.api.tasks.bundling.Jar.html#org.gradle.api.tasks.bundling.Jar:archiveName

5.4.8.3 Directrices sobre definición de dependencias de artefactos internos de tipo war

Aquellos artefactos de tipo **EAR** generados por los proyectos desarrollado en DGT, siguiendo las directrices indicadas en el punto **Directrices de gestión de artefactos generados**, deberán definir las dependencias de artefactos de tipo war mediante el nivel de dependencias "warArchive".

- **[opcional]** el nivel de dependencias (*configurations*) "warArchive", debe ser utilizada para definir las dependencias entre el artefacto de tipo ear y aquellos artefactos de tipo war que deban ser desplegados cómo módulo del ear.

A continuación, se muestra un ejemplo de configuración con el que definir las dependencias de artefactos war a través del nivel de dependencia "warArchive".

```
//Aplicación de plugins
plugins {
    ...
  id 'ear
```

```
        ...
}

apply from: "gradle-project-conf/dependencies.gradle"

dependencies {
        earlib project(':acro-dgt-dao')
         …
        deploy project(path: ':acro-dgt-ws', configuration: 'warArchive')
        …
        earlib libraries.log4j
        …
}

//Configuración de construcción
apply from: "gradle-project-conf/manifest.gradle"
ear {
        ...
        ...
}
...
```

build.gradle

5.4.9 Directrices de generación de artefactos asociados con configuración, recursos y otros distribuibles

Todos los proyectos, desarrollados según las directrices descritas en el presente documento, que contenga ficheros de configuración, recursos y cualquier tipo de fichero que deban ser distribuidos junto a los artefactos desplegables, deberá contener un subproyecto llamado "conf".

El subproyecto "conf" deberá aplicar el plugin distribution, utilizando su funcionalidad para generar los distribuibles necesarios en función de las necesidades propias del proyecto. Para obtener más información recomendamos la lectura de la documentación oficial del plugin disponible en la web https://docs.gradle.org/current/userguide/distribution_plugin.html

Todas las configuraciones necesarias para la generación de artefactos distribuibles asociados a recursos, configuración u otros tipos, se detallan a continuación como directrices que deberán seguir aquellos proyectos que lo necesiten:

- **[opcional]** el proyecto deberá contener la ruta **src/conf/dist** en la que se incluirán todos los ficheros de configuración, habilitándose con ello las tareas confDistTar y confDistZip dentro del proyecto, con los que se podrá generar los artefactos de distribución de ficheros de configuración. Dentro del directio src/conf/dist se encontrará todo el árbol de directorios necesarios.
- **[opcional]** el proyecto deberá contener la ruta **src/rec/dist** en la que se incluirán todos los recursos, habilitándose con ello las tareas recDistTar y recDistZip dentro del proyecto, con los que se podrá generar los artefactos de distribución de recursos. Dentro del directio src/rec/dist se encontrará todo el árbol de directorios necesarios.
- **[opcional]** si además de los ficheros de configuración y recursos, el proyecto incluye otro tipo de ficheros que deban ser distribuidos como entregables, deberá contener la ruta **src/xxxxx/dist** en la que se incluirán todos los ficheros que formen parte de ese entregable, habilitándose con ello las tareas xxxxxDistTar y xxxxxDistZip dentro del proyecto, con los que se podrá generar los artefactos de distribución de otro tipo de ficheros.
- **[opcional]** sección **distributions**: contendrá las secciones de configuración en la que se deberá configurar el nombre de los distribuibles, siguiendo la nomenclatura "conf-${projectInfo.name}" para el artefacto asociado con los ficheros de configuración, y "rec-${projectInfo.name}" para el artefacto asociado con los recursos.
- **[opcional]** siguiendo las directrices de la DGT, los ficheros de distribución deberán ser de tipo ZIP, por lo que las tareas asociadas con el tipo TAR deberán ser deshabilitadas, utilizando para ello la propiedad enable, asignándole el valor false.

A continuación, se muestra varios ejemplos de contenidos del fichero **build.gradle**.

Ejemplo dónde el fichero de configuración y el fichero de recursos son un único zip, en este caso la configuración contendrá las carpetas de los entornos dentro del mismo zip.

```
//Configuración de plugins
plugins {
  id 'distribution'
}
```

```
//Descripción del proyecto
description = """Archivos de configuración del proyecto."""

distributions {
    conf {
        baseName = "conf-${projectInfo.name} "
    }
    rec {
        baseName = "rec-${projectInfo.name}"
    }
}

tasks.confDistTar.enabled = false
tasks.recDistTar.enabled = false
```

${ROOT_PROJECT}/conf/build.gradle

Ejemplo dónde existe un fichero de configuración por cada entorno y el fichero de recursos es un único zip.

```
//Configuración de plugins
plugins {
  id ‘distribution’
}

//Descripción del proyecto
description = """Archivos de configuración del proyecto."""

distributions {
    confDesa {
        baseName = "conf-desa-${projectInfo.name} "
        contents {
                    from 'src/conf/dist/desa'
        }
    }
    confPre {
        baseName = "conf-pre-${projectInfo.name} "
        contents {
                    from 'src/conf/dist/pre'
        }
    }
    confPro {
        baseName = "conf-pro-${projectInfo.name} "
        contents {
                    from 'src/conf/dist/pro'
        }
    }
    rec {
        baseName = "rec-${projectInfo.name}"
    }
}
```

```
tasks.confDesaDistTar.enabled = false
tasks.confPreDistTar.enabled = false
tasks.confProDistTar.enabled = false
tasks.recDistTar.enabled = false
```

${ROOT_PROJECT}/conf/build.gradle

5.4.10 Directrices de ampliación de funcionalidad

Cualquier proyecto, desarrollado ajustado a las directrices descritas en el presente documento, podrá realizar tareas y funcionalidades con las que dar soporte a todas y cada una de sus necesidades específicas, aprovechando para ello la potencia ofrecida por Gradle mediante su lenguaje de programación DSL y la aplicación de scripts plugins.

Cada tarea creada para una finalidad concreta dentro de un proyecto deberá seguir la línea de diseño mostrada a lo largo de este documento, ajustándose para ello a cada una de las directrices detalladas a continuación:

- **<obligatorio>** Cada funcionalidad deberá implementarse en un fichero gradle, con un nombre lo suficientemente descriptivo, siguiendo la notación camelCase.
- **<obligatorio>** Cada fichero gradle generado deberá incluirse en la carpeta gradle-project-conf.
- **<obligatorio>** La aplicación del fichero gradle se realizará justo antes de su utilización, para de esta forma aumentar la legibilidad y comprensión de la configuración realizada.
- **<obligatorio>** Cada tarea incluida en los ficheros de funcionalidad de proyecto, deberán de tener un nombre descriptivo, utilizando la notación camelCase. (task name)
- **<obligatorio>** Cada tarea creada deberá contener debidamente indicado el tipo Gradle del que hereda. (type)
- **<obligatorio>** Cada tarea deberá incluir el grupo al que pertenece. (group)
- **<obligatorio>** Cada tarea deberá incluir una descripción a modo de documentación. (description)
- **<obligatorio>** Cada tarea debe contener una implementación que se ajuste a las buenas prácticas indicadas por gradle en la documentación oficial del tipo de tarea de que hereda.

- **<obligatorio>** Cada tarea deberá estar correctamente configurada, para que se pueda saltar durante la construcción del proyecto, mediante una propiedad cuyo nombre estará formado por el prefijo skip y el nombre de la tarea. (skip${taskName})
- **<obligatorio>** Aquellas tareas que dependan de otras y deban ser ejecutadas de forma conjunta estarán debidamente enlazadas mediante dependencia gradle. (dependsOn)
- **<obligatorio>** Aquellas tareas que deban incluirse en el ciclo de vida de construcción gradle, deberá estar debidamente configurado mediante la dependencia de la tarea del plugin que la lanzará. (task dependsOn)
- **<obligatorio>** Cada tarea deberá ser parametrizada para que su uso pueda ser replicado en todos los casos que sea necesario y de esta forma no duplicar código dentro del proyecto.
- **<obligatorio>** Cada tarea que necesite información disponible en el objeto ext, deberá utilizarlos a partir de las propiedades existentes, con el objetivo de no duplicar información ni establecer valores a fuego.
- **<obligatorio>** Cada propiedad creada y utilizada en las tareas creadas, deberán ser incluidas en la configuración global del proyecto mediante el proyecto ext, siempre que sea posible y su uso se pueda hacer extensivo por otras tareas o funcionalidades.

6 Ejemplos de configuración de proyecto multimódulo

A lo largo de este apartado se muestra un ejemplo de configuración de proyecto multimódulo siguiendo las directrices que se describen en el presente documento.

El proyecto ejemplo está formado por dos subproyectos o módulos, uno de tipo jar y otro de tipo war. Ambos están definido en el fichero settings.gradle del proyecto principal, de manera que la construcción se realizará de forma conjunta al lanzar la tarea build sobre el fichero build.gradle del proyecto padre.

Cada subproyecto contiene un fichero build.gradle con la configuración propia del proyecto, y la carpeta gradle-project-conf, donde se ubicarán todos los ficheros gradle necesarios para su construcción, en el caso de proyecto de tipo jar, contendrá además del fichero dependencies.gradle y manifest.gradle, los scripts de tareas necesarios para poder construir ficheros javadoc y sources, además de la funcionalidad de firma de artefactos.

6.1 Estructura del proyecto

```
Root project 'acro-dgt-multiproject'
+--- settings.gradle
+--- gradle.properties
\--- build.gradle
+--- \gradle-env-conf
        +---- environmentBuildConfig.gradle
+--- \conf
        +---- build.gradle
        +----\src\
            +---- \conf\
                +---- \dist
            +---- \rec\
                +---- dist
+--- \acro-dgt-dao
        +---- build.gradle
        +----\gradle-project-conf\
                +---- dependencies.gradle
                +---- manifest.gradle
                +---- generateJavadocJar.gradle
                +---- generateSourcesJar.gradle
                +---- signJar.gradle
+--- \acro-dgt-ws
        +---- build.gradle
        +----\gradle-project-conf\
                +---- dependencies.gradle
                +---- manifest.gradle
                +---- generateStaticFilesZip.gradle
+--- \acro-dgt-app
        +---- build.gradle
        +----\gradle-project-conf\
                +---- dependencies.gradle
                +---- manifest.gradle
```

acro-dgt-multiproject

6.2 Configuración de proyecto principal

6.2.1 Configuración del fichero settings.gradle

```
pluginManagement {
    repositories {
        maven {
            url "${artifactoryGradlePluginsUrl}"
        }
    }
}

rootProject.name = 'acro-dgt-multiproject'

include 'conf'
include 'acro-dgt-dao'
include 'acro-dgt-ws'
include 'acro-dgt-app'
```

acro-dgt-multiproject/settings.gradle

6.2.2 Configuración del fichero gradle.properties

```
#Configuración de compilación
pathJdk7=C:/herramientas/Java/jdk1.7.0_71
pathJdk8=C:/herramientas/Java/jdk1.8.0_191

#Configuración de acceso a artifactory
artifactoryGradlePluginsUrl=http://artifactory.trafico.es:8080/artifactory/dgt-gradle-plugins
artifactoryDependenciesUrl=http://artifactory.trafico.es:8080/artifactory/dgt-central-ic
artifactoryUser=ic-reader
artifactoryPassword=usrIC@reader
```

```
#Configuración de firma electrónica
jarsignerWorkingDir=C:/herramientas/Java/jdk1.8.0_191/bin
jarsignerKeystore=C:/firma-digital/keystore/firma.jks
jarsignerAlias=alias
jarsignerPassword=password
```

acro-dgt-multiproject/gradle.properties

6.2.3 Configuración del fichero build.gradle

```
plugins {
  id 'project-report'
  id 'org.sonarqube' version '2.7
}

defaultTasks 'clean', 'build', 'projectReport'

//Información global de proyecto
ext {
  projectInfo = [
    description : """Proyecto de ejemplo de la normativa de configuración gradle""",
    organization : Dirección General de Tráfico',
    developmentVendor : 'Altran',
    acronimo : 'ACRO',
    group: "es.trafico.${project.name}",
    name : "${project.name}",
    version : '1.0.0-SNAPSHOT',
    encoding : 'UTF-8',
    sourceJdkVersion : '1.8',
    targetJdkVersion : '1.8'
  ]
}
//Configuración de información de construcción del proyecto
apply from: "gradle-env-conf/environmentBuildConfig.gradle"

//Descrpción del proyecto
```

```
description = projectInfo.description

//Configuración global para el proyecto y todos sus componentes
allprojects {
  //Identificación del proyecto
  group = projectInfo.group
  version = projectInfo.version

  //Configuración de acceso a los repositorios de artefactos
  repositories {
                maven {
                        url "${artifactoryDependenciesUrl}"
                        credentials {
                                username "${artifactoryUser}"
                                password "${artifactoryPassword}"
                        }
                }
        }
}

wrapper {
    gradleVersion = '5.2.1'
}
```

acro-dgt-multiproject/build.gradle

6.2.4 Configuración del fichero environmentBuildConfig.gradle

```
//Información sobre la construcción del proyecto proyecto
ext {
        projectBuildInfo = [
                buildDate : new Date().format("yyyy-MM-dd'T'HH:mm:ss"),
                pathJavac : "${pathJdk8}/bin/javac",
                buildGradleGroovy: "gradle-${getGradle().getGradleVersion()}, groovy-
                ${GroovySystem.getVersion()}
        ]
```

```
        isCiServer = System.getenv().containsKey("CI_SERVER")

    if (isCiServer) {
                projectBuildInfo += [
                        buildNumber : System.getenv("BUILD_NUMBER")
                ]
        }
}
```

acro-dgt -multiproject/gradle-env-conf/environmentBuildConfig.gradle

6.3 Configuración de subproyecto conf

6.3.1 Configuración de fichero build.gradle

```
//Configuración de plugins
plugins {
  id 'distribution'
}

//Descrpción del proyecto
description = """Archivos de configuración del proyecto."""

distributions {
    conf {
        baseName = "conf-${projectInfo.name} "
    }
    rec {
        baseName = "rec-${projectInfo.name}"
    }
}

tasks.confDistTar.enabled = false
tasks.recDistTar.enabled = false
```

acro-dgt-multiproject/conf/build.gradle

6.4 Configuración de subproyecto Java

6.4.1 Configuración de fichero build.gradle

```
//Configuración de plugins
plugins {
  id 'java'
  id 'jacoco'
  id 'project-report'
}

//Descrpción del proyecto
description = """Librería de acceso a datos del proyecto"""



//Configuración de versionado JDK para la compilación y ejecución
sourceCompatibility = projectInfo.sourceJdkVersion
targetCompatibility = projectInfo.targetJdkVersion

compileJava {
        options.fork = true
        options.forkOptions.executable = projectBuildInfo.pathJavac
        options.encoding = projectInfo.encoding
}



//Configuración de dependencias
apply from: "gradle-project-conf/dependencies.gradle"
dependencies {
  compile libraries.slf4j_api
  compile libraries.hibernateCore
  compile libraries.hibernateEntitymanager
  compile libraries.hibernate
```

```
  testCompile libraries.junit
}


//Configuración específica del empaquetado jar
apply from: "gradle-project-conf/manifest.gradle"
jar {
  manifest = project.manifest {
    from projectManifest
  }
}

//Configuración específica del empaquetado jar asociado a javadoc
apply from: "gradle-project-conf/generateJavadocJar.gradle"
//Configuración específica del empaquetado jar asociado a sources
apply from: "gradle-project-conf/generateSourcesJar.gradle"
//Configuración específica para la publicación de los componentes generados
apply from: "gradle-project-conf/publishArtifactoryJar.gradle"

//Configuración específica para el firmado de jar
apply from: "gradle-project-conf/signJar.gradle"
```

acro-dgt-multiproject/acro-dgt-dao/build.gradle

6.4.2 Configuración de fichero dependencies.gradle

```
//Dependencias del proyecto
ext {
	libraries = [:]
	versions = [:]
}

versions += [
	junit : "4.12",
```

```
        slf4jApi : "1.7.21",
        hibernateCore : "3.5.6-Final",
        hibernateEntitymanager : "3.5.6-Final",
        hibernate : "3.5.4-Final"
]

libraries += [
        junit : "junit:junit:${versions.junit}",
        slf4jApi : "org.slf4j:slf4j-api:${versions.slf4jApi}",
        hibernateCore : "org.hibernate:hibernate-core:${versions.hibernateCore}",
        hibernateEntitymanager : " org.hibernate:hibernate-entitymanager:${versions.hibernateEntitymanager}",
        hibernate : "org.hibernate:hibernate:${versions.hibernate}"
]
```

acro-dgt-multiproject/acro-dgt-dao/gradle-project-conf/dependencies.gradle

6.4.3 Configuración de fichero manifest.gradle

```
//Información a incluir en el fichero manifest
ext {
  projectManifest = manifest {
     attributes('Created-By' : "${projectBuildInfo.buildGradleGroovy}",
                'Source-Compatibility' : "${projectInfo.sourceJdkVersion}",
                'Target-Compatibility' : "${projectInfo.targetJdkVersion}",
                'Built-By' : "${projectInfo.organization}",
                'Build-Time' : "${projectBuildInfo.buildDate}",
                'Specification-Title' : "${projectInfo.group} :: ${projectInfo.name} ::
                                $name :: $description",
                'Specification-Version' : "${projectInfo.version}",
                'Specification-Vendor' : "${projectInfo.developmentVendor}",
                'Implementation-Title' : "${projectInfo.group} :: ${projectInfo.name} ::
                                $name :: $description",
                'Implementation-Version' : "${projectInfo.version}",
                'Implementation-Vendor' : "${projectInfo.developmentVendor}"
                )
  }
```

```
  if (isCiServer) {
    projectManifest.getAttributes().put('Build-Number',
                                        "${projectBuildInfo.buildNumber}")
  }
}
```

acro-dgt-multiproject/acro-dgt-dao/gradle-project-conf/manifest.gradle

6.4.4 Configuración de fichero generateJavadocJar.gradle

```
task generateJavadoc (type : Javadoc) {
  group = 'Documentation'
  description = "[Create-by: ${projectInfo.name}] - Generates Javadoc API
                 documentation for the main source code"
  source = sourceSets.main.allJava
  classpath = sourceSets.main.compileClasspath
  failOnError = false
}
generateJavadoc.onlyIf { !project.hasProperty('skipGenerateJavadocJar') }

task generateJavadocJar(type: Jar, dependsOn: generateJavadoc) {
  group = 'Build'
  description = "[Create-by: ${projectInfo.name}] - Assembles a jar archive containing
                 the javadoc."
  classifier = 'javadoc'
  from generateJavadoc.destinationDir
  manifest = project.manifest {
    from projectManifest
  }
}
generateJavadocJar.onlyIf { !project.hasProperty('skipGenerateJavadocJar') }

artifacts {
  archives generateJavadocJar
}
```

acro-dgt-multiproject/acro-dgt-dao/gradle-project-conf/generateJavadocJar.gradle

6.4.5 Configuración de fichero generateSourcesJar.gradle

```
task generateSourcesJar(type: Jar) {
  group = 'Build'
  description = "[Create-by: ${projectInfo.name}] - Assembles a jar archive containing
                        the sources."
  classifier = 'sources'
  from sourceSets.main.allSource
  manifest = project.manifest {
    from projectManifest
  }
}
generateSourcesJar.onlyIf { !project.hasProperty('skipGenerateSourcesJar') }

artifacts {
  archives generateSourcesJar
}
```

acro-dgt-multiproject/acro-dgt-dao/gradle-project-conf/generateSourcesJar.gradle

6.4.6 Configuración de fichero publishArtifactoryJar.gradle

```
apply plugin: 'maven-publish'

publishing {
    publications {
        mavenJava(MavenPublication) {
            from components.java
            artifact generateSourcesJar
            artifact generateJavadocJar
        }
    }
    repositories {
        maven {
            url "${artifactoryPublishUrl}"
            credentials {
```

```
            username "${artifactoryPublishUser}"
            password "${artifactoryPublishPassword}"
        }
    }
  }
}
```

acro-dgt-multiproject/acro-dgt-dao/gradle-project-conf/publishArtifactoryJar.gradle

6.4.7 Configuración de fichero signJar.gradle

```
task signJar (type: Exec) {
  group = 'Build'
  description = "[Create-by: ${projectInfo.name}] - Sign a jar archive."
  workingDir jarsignerJdkDir
  executable = jarsigner
  args = '-keystore', jarsignerkeyStore
  args = '-storepass, jarsignerStorePass
  args jar.archivePath
  args jarsignerAlias
}
signJar.onlyIf { !project.hasProperty('skipSignJar') }

task verifySignedJar (type: Exec, dependsOn: signJar) {
  group = 'Build'
  description = "[Create-by: ${projectInfo.name}] – Verify a jar archive."
  workingDir jarsignerJdkDir
  executable = jarsigner
  args = '-verify
  args jar.archivePath
}
signJar.onlyIf { !project.hasProperty('skipVerifySignedJar') }

build.dependsOn verifySignedJar
```

acro-dgt-multiproject/acro-dgt-dao/gradle-project-conf/generateSourcesJar.gradle

6.5 Configuración de subproyecto War

6.5.1 Configuración de fichero build.gradle

```
//Configuración de plugins
plugins {
  id ‘war’
  id 'project-report'
  id 'java'
  id 'jacoco'
}

//Niveles de dependencia
configurations {
    warArchive
}

//Descrpción del proyecto
description = """Aplicación ws del proyecto"""

//Configuración de dependencias
apply from: "gradle-project-conf/dependencies.gradle"
dependencies {
  compile libraries.springCore
  compile libraries.jaxbApi
  compile libraries.jaxbImpl
  compile libraries.commonsHttpclient
  compile libraries.slf4jLog4j12
  compile libraries.jta
  compile libraries.jodaTime

  compile project(':acro-dgt-dao')
}
```

```
//Configuración específica del empaquetado war
war.archiveName = "${war.baseName}.${war.extension}"

apply from: "gradle-project-conf/manifest.gradle"
war {
  …
  exclude("images")
  exclude("styles")
  …
  manifest = project.manifest {
    from projectManifest
  }
}
artifacts {
    warArchive war
}

apply from: “gradle-project-conf/generateStaticFilesZip.gradle”
```

acro-dgt-multiproject/acro-dgt-ws/build.gradle

6.5.2 Configuración de fichero dependencies.gradle

```
//Dependencias del proyecto
ext {
        libraries = [:]
        versions = [:]
}

versions += [
        springCore : "4.2.4.RELEASE",
        jaxbApi : "2.2",
        jaxbImpl : "2.2",
        commonsHttpclient : "3.1",
```

```
        slf4jLog4j12 : "1.6.1",
        jta : "1.1",
        jodaTime : "2.3"

]

libraries += [
        springCore : "org.springframework:spring-core:${versions.springCore}",
        jaxbApi : "javax.xml.bind:jaxb-api:${versions.jaxbApi}",
        jaxbImpl : "com.sun.xml.bind:jaxb-impl:${versions.jaxbImpl}",
        commonsHttpclient : "commons-httpclient:commons-httpclient:${versions.commonsHttpclient}",
        slf4jLog4j12 : "org.slf4j:slf4j-log4j12:${versions.slf4jLog4j12}",
        jta : "javax.transaction:jta:${versions.jta}",
        jodaTime : "joda-time:joda-time:${versions.jodaTime}"

]
```

acro-dgt-multiproject/acro-dgt-ws/gradle-project-conf/dependencies.gradle

6.5.3 Configuración de fichero manifest.gradle

```
//Información a incluir en el fichero manifest
ext {
  projectManifest = manifest {
     attributes('Created-By' : "${projectBuildInfo.buildGradleGroovy}",
                'Source-Compatibility' : "${projectInfo.sourceJdkVersion}",
                'Target-Compatibility' : "${projectInfo.targetJdkVersion}",
                'Built-By' : "${projectInfo.organization}",
                'Build-Time' : "${projectBuildInfo.buildDate}",
                'Specification-Title' : "${projectInfo.group} :: ${projectInfo.name} ::
                                $name :: $description",
                'Specification-Version' : "${projectInfo.version}",
                'Specification-Vendor' : "${projectInfo.developmentVendor}",
                'Implementation-Title' : "${projectInfo.group} :: ${projectInfo.name} ::
                                $name :: $description",
                'Implementation-Version' : "${projectInfo.version}",
                'Implementation-Vendor' : "${projectInfo.developmentVendor}"
```

```
        )
    }

    if (isCiServer) {
      projectManifest.getAttributes().put('Build-Number',
                                          "${projectBuildInfo.buildNumber}")
    }
}
```

acro-dgt-multiproject/acro-dgt-ws/gradle-project-conf/manifest.gradle

6.5.4 Configuración de fichero generateStaticFilesZip.gradle

```
task generateStaticFilesZip (type: Zip) {
  group = 'Build'
  description = "[Create-by: ${projectInfo.name}] - Generate static web content ZIP file."
  archiveName = "static-${war.baseName}.zip"
  from "src/main/webapp"
  include "images/*.*"
  include "styles/**/*.*"
}
generateStaticFilesZip.onlyIf { !project.hasProperty('skipGenerateStaticFilesZip') }

artifacts {
  archives generateStaticFilesZip
}
```

${ROOT_PROJECT}/gradle-project-conf/generateStaticFilesZip.gradle

6.6 Configuración de subproyecto Ear

6.6.1 Configuración de fichero build.gradle

```
//Configuración de plugins
plugins {
  id ‘ear
  id 'project-report'
```

```
  id 'java'
}

//Descrpción del proyecto
description = """Aplicación desplegable del proyecto"""

//Configuración de dependencias
apply from: "gradle-project-conf/dependencies.gradle"
dependencies {
    deploy project(path: ':acro-dgt-ws', configuration: 'warArchive')
    ....
    earlib libraries.gson
    earlib libraries.commonsCodec
    earlib libraries.commonsLang
    ...
}

apply from: "gradle-project-conf/manifest.gradle"
ear {
    manifest = project.manifest {
        from projectManifest
    }
    deploymentDescriptor {
        fileName = "application.xml"
        displayName = "${project.name}"
        description = "Empaquetado en un archivo ear del proyecto."
        webModule("acro-dgt-ws.war", "/WEB")
    }
}
```

acro-dgt-multiproject/acro-dgt-app/build.gradle

6.6.2 Configuración de fichero dependencies.gradle

```
//Dependencias del proyecto
ext {
        libraries = [:]
```

```
        versions = [:]
}

versions += [
        gson : "2.8.0",
        commonsCodec: "1.10",
        commonsLang: "2.4"
]

libraries += [
        gson : "com.google.code.gson:gson:${versions.gson}",
        commonsCode : "commons-codec:commons-codec:${versions.commonsCodec}",
        commonsLang : "commons-lang:commons-lang:${versions.commonsLang}"

]
```

acro-dgt-multiproject/acro-dgt-app/gradle-project-conf/dependencies.gradle

6.6.3 Configuración de fichero manifest.gradle

```
//Información a incluir en el fichero manifest
ext {
  projectManifest = manifest {
     attributes('Created-By' : "${projectBuildInfo.buildGradleGroovy}",
                'Source-Compatibility' : "${projectInfo.sourceJdkVersion}",
                'Target-Compatibility' : "${projectInfo.targetJdkVersion}",
                'Built-By' : "${projectInfo.organization}",
                'Build-Time' : "${projectBuildInfo.buildDate}",
                'Specification-Title' : "${projectInfo.group} :: ${projectInfo.name} ::
                                $name :: $description",
                'Specification-Version' : "${projectInfo.version}",
                'Specification-Vendor' : "${projectInfo.developmentVendor}",
                'Implementation-Title' : "${projectInfo.group} :: ${projectInfo.name} ::
                                $name :: $description",
                'Implementation-Version' : "${projectInfo.version}",
                'Implementation-Vendor' : "${projectInfo.developmentVendor}"
                )
```

```
  }

  if (isCiServer) {
    projectManifest.getAttributes().put('Build-Number',
                                        "${projectBuildInfo.buildNumber}")
  }
}
```

acro-dgt-multiproject/acro-dgt-app/gradle-project-conf/manifest.gradle

6.7 Artefactos generados

```
Root project 'acro-dgt-multiproject'
+--- \conf
        +---- \build
                +---- \ distributions
                        +---- conf-acro-dgt-multiproject-1.0.0-SNAPSHOT.jar
                        +---- rec-acro-dgt-multiproject-1.0.0-SNAPSHOT.jar
+--- \acro-dgt-dao
        +---- \build
                +---- \libs
                        +---- acro-dgt-dao-1.0.0-SNAPSHOT.jar
                        +---- acro-dgt-dao-1.0.0-SNAPSHOT-javadoc.jar
                        +---- acro-dgt-dao-1.0.0-SNAPSHOT-sources.jar
+--- \acro-dgt-ws
        +---- \build
                +---- \libs
                        +---- acro-dgt-ws.war
                +---- \distributions
                        +---- static-acro-dgt-ws.zip
+--- \acro-dgt-app
        +---- \build
                +---- \libs
                        +---- acro-dgt-app-1.0.0-SNAPSHOT.ear
```

acro-dgt-multiproject