Escuela Superior de Ingeniería

ESCUELA SUPERIOR DE INGENIERÍA

MÁSTER EN INGENIERÍA INFORMÁTICA

RaViGEn. Generador de Video Radar desde información DIS para sistemas de adiestramiento marítimo.

AUTOR(A): Antonio García Alba

4 de septiembre de 2016

Escuela Superior de Ingeniería

ESCUELA SUPERIOR DE INGENIERÍA

MÁSTER EN INGENIERÍA INFORMÁTICA

RaViGEn. Generador de Vídeo Radar desde información DIS para sistemas de adiestramiento marítimo.

AUTOR(A): Antonio García Alba
DIRECTOR(A): Manuel Palomo Duarte
DIRECTOR(A): Juan Manuel Dodero Beardo

Cádiz, 4 de septiembre de 2016

Agradecimientos

A todo lo que pudo ser y a todo lo que será, a los que dejé por el camino y a los que me volveré a encontrar. A todos los profesores de la Escuela Superior de Ingeniería que con tanto esfuerzo y cariño nos han acogido en esta nueva etapa.

Resumen

Este Trabajo de Fin de Máster, en adelante TFM, se centra en el desarrollo de un producto, llamado RaViGEn, que permita la generación de vídeo radar simulado a partir de tráfico DIS para centros de adiestramiento de sistemas de mando y control.

La simulación de vídeo radar, o vídeo en crudo, es una necesidad histórica que no ha sido cubierta debido a la alta inversión necesaria para conseguir una simulación acorde a las necesidades de estos centros de adiestramiento. La integración en escenarios coordinados, haciendo uso del protocolo DIS, es un requisito *sine qua non* la simulación no es una herramienta útil.

Tomando como referencia los estándares de la industria y haciendo uso de tecnologías libres se construye un motor de simulación que permite generar cualquier tipo de señal de vídeo radar sobre una red TCP/IP con soporte UDP multicast.

La implemetación que ofrece RaViGEn permite resolver los siguientes retos:

- Evitar la necesidad, debido a su alto coste, de instalar sistemas reales que sean sensibles a entornos simulados en los centros de adiestramiento.
- Disponer de una solución software, abierta a la extensión, que sea capaz de simular diferentes formatos de vídeo radar digital.
- Ser capaces de integrar la señal de vídeo radar digital en una red convergente haciendo uso de un ancho de banda limitado.
- Ofrecer la capacidad de incorporar los distintos modelos de plataformas definidas en el estándar DIS consiguiendo que la firma radar presentada esté acorde con la naturaleza del contacto generado en la simulación.

El desarrollo de este proyecto ha seguido los principios Lean con el objetivo de maximinar el retorno de la inversión de las horas invertidas. Bajo estos principios se ha adaptado la metodología Scrum para el desarrollo, seguimiento y control del mismo.

El estudio del estándar DIS, el dominio de la representación georeferenciada de la información, la profundización en tecnologías de procesamiento paralelo y de la computación gráfica son aspectos claves en el éxito de este proyecto.

Junto al componente principal, RaViGEn Core, se han desarrollado dos componetes fundamentales en el éxito de este proyecto:

- RaViGEn Test Console es la aplicación encargada de presentar el vídeo radar para su validación visual.
- Platforms Simulator es una aplicación que nos permite ejecutar escenarios de simulación DIS para la validación de RaViGEn Core.

RaViGEn se distingue como un producto estable, con un núcleo sólido, que permite su fácil ampliación con nuevas características.

Palabras clave: simulación, dis, openmp, radar, adiestramiento

Índice general

Índice de figuras

Índice de tablas

Parte I

Prolegómeno

Capítulo 1

Introducción

A continuación, se describe la motivación del presente proyecto y su alcance. También se incluye un glosario de términos y la organización del resto de la presente documentación.

1.1. Motivación y Objeto

Vivimos un periodo de fomento del desarrollo de sistemas de mando y control marítimos por parte de empresas nacionales. Históricamente el Estado ha confiado en fiables sistemas de desarrollo foráneo [Infodefensa, 2009]. El alto coste de estos sistemas, la necesidad de inversión en la industria nacional, y la eliminación de la dependencia de otras naciones, son las motivaciones principales que impulsan este cambio de estrategia. Estar actualmente inmersos en este periodo de nacionalización, en pleno proceso de desarrollo de nuevos sistemas, provoca que los buques nacionales no dispongan de sistemas totalmente operativos.

Un sistema de mando y control marítimo es la integración de un conjunto de sensores y actuadores en la misma plataforma de operación para llevar a cabo una misión específica. Uno de los elementos principales de estos sistemas son los radares. Los radares ofrecen al sistema de integración dos fuentes principales de información: las trazas y el vídeo radar. En un sistema con un desempeño ideal el vídeo radar no es un recurso necesario para la operación. El sistema sería capaz de presentar la información fusionada de varias fuentes radar de alta calidad. En un sistema real, en estado de desarrollo y depuración, el vídeo radar es un componente esencial. Los operadores necesitan la información del radar en crudo, conocida como vídeo radar, para saber si el radar está reportando correctamente la información de contexto.

El adiestramiento es un aspecto fundamental para la correcta operación de un sistema de mando y control. El Estado apuesta por el uso del estándar Distributed Interactive Simulation (DIS) en sus sistemas de enseñanza. DIS es un estándar del IEEE orientado a la simulación tiempo real de plataformas y contexto operativo a través de múltiples estaciones de proceso usado tanto por organizaciones militares como por otras organizaciones relacionadas con el control de tráfico marítimo, la medicina y la exploración espacial.

Los sistemas de adiestramiento disponibles en la actualidad no contemplan la presentación de vídeo radar. La integración de la señal de vídeo radar en estos sistemas presenta varios retos:

- Evitar la necesidad, debido a su alto coste, de instalar sistemas reales que sean sensibles a entornos generados por simuladores en los centros de adiestramiento.

- Disponer de una solución software, abierta a la extensión, que sea capaz de simular diferentes formatos de vídeo radar digital.

- Ser capaces de integrar la señal de vídeo radar digital en una red convergente haciendo uso de un ancho de banda limitado.

El objetivo principal de este trabajo fin de máster es resolver los retos que presenta la integración y presentación de vídeo radar en entornos de simulación y adiestramiento distribuido DIS.

1.2. Alcance

El alcance de este Trabajo Fin de Máster se basa en reconocer las entidades reportadas por los motores de simulación que cumplen el estándar DIS, generando el tráfico necesario sobre tecnologías de difusión de red para que sea posible el envío de la señal digital de vídeo radar obtenida de modelos matemáticos sobre una red convergente.

Dentro de este proyecto se abordarán los siguientes aspectos:

- La investigación e integración de distintas tecnologías. Revisión del estándar DIS, análisis de alternativas de diseño e implementación de aplicaciones, servicios y utilidades, teniendo en cuenta el contexto de aplicación de la solución propuesta.

- El uso de técnicas ágiles para la planificación y el seguimiento del proyecto. Uso de Scrum y Lean para conseguir el mayor retorno de la inversión del tiempo usado en la elaboración de este proyecto.

- La aplicación de estándares de computación gráfica y de altas prestaciones cuando sea necesario. Resolver la necesidad de procesar gran cantidad de entidades y la generación de vídeo radar adecuada en los diferentes contextos de operación.

- El desarrollo de las pruebas necesarias para la validación del sistema. Aplicación de la ingeniería de pruebas para la consecución de un sistema fiable.

- La incorporación de nuevo tipo de tráfico a una red convergente. Aplicar técnicas de control de flujo para evitar la saturación de las redes existentes.

- La capacidad de extensión del sistema. Como aspecto transversal conseguir un producto de vida longeva que permita ser una solución ampliable a nuevos requisitos y necesidades.

1.3. Glosario de Términos

Esta sección contiene una lista ordenada alfabéticamente de los principales términos, acrónimos y abreviaturas específicos del dominio del problema.

DIS Distributed Interactive Simulation es un estándar IEEE para el desarrollo de juegos de guerra a través de múltiples plataformas, hospedado en múltiples ordenadores. Es ampliamente usado, especialmente en organizaciones militares, agencias espaciales y médicas. Está recogido en la serie IEEE 1278 bajo el control de la SISO.

Scrum El proceso Scrum se basa en ciclos de desarrollos cortos en los que los requisitos pueden ser variables y el proceso está orientado al desarrollo de prototipos incrementales y funcionales.

SISO Simulation Interoperability Standards Organization es una organización dedicada a la promoción de la operatividad y el reuso para el beneficio de las distintas comunidades dedicadas a la simulación.

Revolución Ciclo de barrido de 360 grados realizado por un radar.

Radio Precisión máxima angular que contiene la información de detacción del radar.

Retorno Valor de intensidad de en un radio a un distancia concreta representado en una celda.

Celda Precisión máxima que relaciona la precisión en distancia con la precisión angular. Define la unidad, en tamaño, mínima de detección.

Rango Alcance o distancia máxima de operatividad del radar.

1.4. Organización del documento

Esta sección contiene una descripción de los contenidos de la presente memoria.

La primera parte de la memoria del TFM consta de una introducción y una planificación del proyecto detallando los aspectos más relevantes del desarrollo del mismo. La introducción presenta a modo de resumen una breve descripción del contexto del proyecto y la motivación para su desarrollo, destacando la definición del alcance previsto. La planificación incluye los plazos, los entregables, los recursos y la metodología de ingeniería de software a emplear.

La segunda parte de la memoria se compone de la documentación del desarrollo siguiendo la metodología de ingeniería asociada al desarrollo de software. Con apartados de requisitos, análisis, diseño, construcción y pruebas adaptados al uso de la metodología seleccionada para ejecutar este TFM.

En la tercera y última parte están recogidas las conclusiones así como los manuales necesarios para el manejo de la aplicación resultado del desarrollo.

Capítulo 2

Planificación

2.1. Metodología de desarrollo

El proyecto es gestionado aplicando los métodos de desarrollo ágil propuestos y practicados en el currículo del Máster de Ingeniería Informática, principalmente haciendo uso de Scrum como metodología y Lean filosofía general. Orientamos la gestión del proyecto a la consecución de valor frente al seguimiento de un plan innegociable. Aunque el proyecto queda bien definido en su alcance y objetivo, el cambio es bien recibido, dentro de los márgenes que establece la normativa de desarrollo del Trabajo Fin de Máster (TFM), tal como establecen los principios de Scrum.

El Manifiesto Ágil [Firmantes Manifiesto Ágil, 2001] sirve como guía de orientación a conseguir el mayor retorno de la inversión posible en el desarrollo de este proyecto:

> Estamos descubriendo formas mejores de desarrollar software tanto por nuestra propia experiencia como ayudando a terceros. A través de este trabajo hemos aprendido a valorar:

Individuos e interacciones	sobre	procesos y herramientas.
Software funcionando	sobre	documentación extensiva
Colaboración con el cliente	sobre	negociación contractual
Respuesta ante el cambio	sobre	seguir un plan

Tabla 2.1: Manifiesto Ágil

> Esto es, aunque valoramos los elementos de la derecha, valoramos más los de la izquierda.

El desarrollo del proyecto es iterativo e incremental basado en tres pilares fundamentales:

1. Transparencia. Las personas involucradas en el proyecto tienen visibilidad total del progreso, planificación, cambio...

2. Inspección. Se comprueba a tiempo que el producto evoluciona acorde con los objetivos. El proyecto será revisado en cada reunión de retrospectiva.

3. Adaptación. El proceso se ajusta para minimizar la desviación de los objetivos. La priorización y modificación del *backlog* permite reconducir el desarrollo del producto en caso de que sea necesario.

La elección de Scrum sobre Kanban es debida a que no se necesita gestionar un pool de trabajos sobre un equipo. No es un requisito necesario llevar al estado de completado el mayor número de tareas posibles optimizando el tiempo de iniciación de cada uno de los items. Al ser un proyecto individual, autogestionado, sumado a la dependencia entre las actividades, no se establece el valor del *Work In Progress* (WIP) a priori. Será ajustado según las necesidades de las actividades desarrolladas. Se establece un plan de iteraciones cortas que nos lleven a la entrega de un producto completo al final de cada una de ellas evaluando y optimizando el proceso siempre que sea necesario. Los roles definidos por Scrum serán ocupados en su totalidad por el autor del TFM colaborando con el director del TFM en las labores de *Product Owner*. La herramienta *Taiga* permite renombrar el rol de los integrantes del equipo de trabajo usando el alias de *Alumno*. Este alias será usado para identificar los trabajos realizados para superar el TFM por el alumno.

La aplicación de los principios Lean es fundamental en el desarrollo de este proyecto. Nos centraremos en todas aquellas tareas que aporten valor al cliente. En este caso tipificamos el *cliente* como todo aquello necesario para superar este TFM eliminando el desperdicio entregando un software que funciona, con un diseño flexible preparado para el cambio y una organización colaborativa donde la comunicación tiene gran importancia. La decisiones serán retrasadas hasta el último momento creando un producto de calidad orientado a cumplir el objetivo del mismo. Se evita el trabajo parcialmente completado, el intercambio continuo de tareas aplicando una filosofía de cero defectos.

Al no usar ninguna metodología de manera rígida, enfoque que iría en contra de la metodología ágil, podemos considerar que usamos un proceso Ad Hoc que nos permite cumplir con los requisitos establecidos por la normativa de desarrollo de los TFMs y hacer un seguimiento cercano y real del TFM centrando el foco en aquellos aspectos que realmente aporten valor a la superación de los créditos asociados a esta actividad.

El proceso Scrum se basa en ciclos de desarrollo cortos en los que los requisitos pueden ser variables, como puede verse en la figura 2.1. El proceso está orientado al desarrollo de prototipos incrementales que contengan un mínimo de funcionalidad que nos permitan tener un producto operativo para que pueda ser evaluado por el cliente.

Se parte de una lista priorizada de los requisitos conocida como Product Backlog. Cada requisito, o historia de usuario, tendrá un nombre, una prioridad, una estimación inicial, un solicitante, una justificación y las indicaciones necesarias para saber cuando el requisito es aceptado. El lenguaje de redacción tendrá que ser no técnico.

La lista de historias de usuario serán repartidas por Sprints que están definidos por un nombre y un intervalo de fechas. Las fechas quieren delimitar el número de horas de trabajo invertidas en un Sprint. En nuestro caso, debido a la naturaleza de la inversión de tiempo en la realización del TFM, el intervalo de fechas será cambiado por un número de horas de trabajo.

En los siguientes apartados se describe como resolver los distintos procesos de la gestión de proyectos.

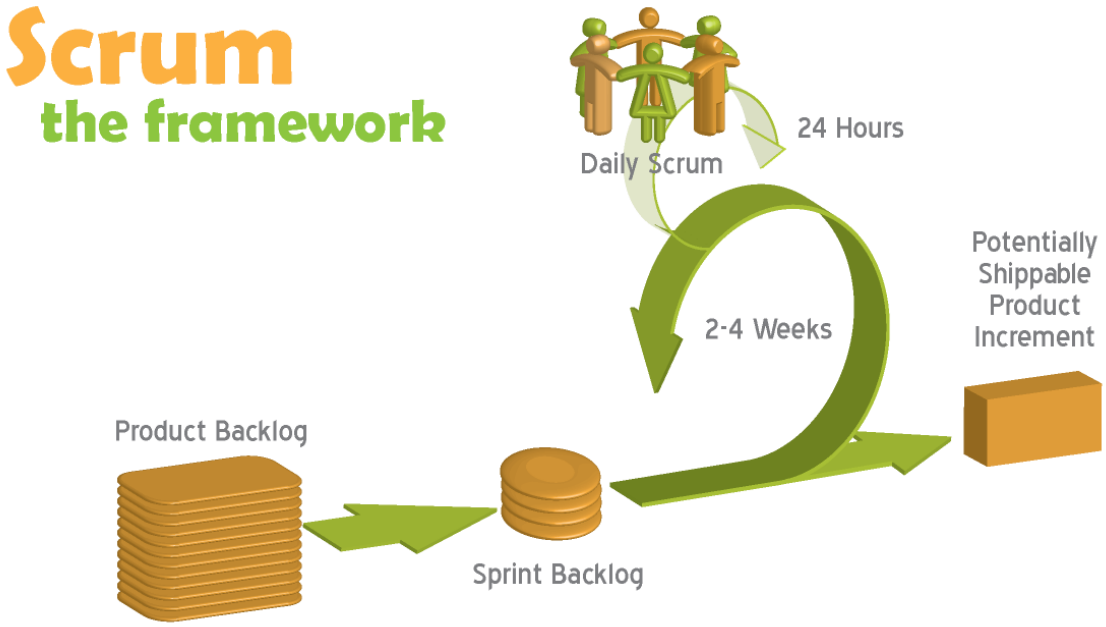


Figura 2.1: Ciclo general de Scrum

2.1.1. Iniciación del proyecto

El proyecto se inicia con la aprobación del alcance y objeto por parte de la Comisión de Trabajos Fin de Máster.

Para dar a conocer correctamente el alcance del proyecto se usará un subconjunto de las distintas técnicas de iniciación de proyectos propuestas en el *Inception Desk* recogidas en la publicación Agile Samurai [Ramusson, 2010].

1. ¿Por qué estamos aquí? (Why are we here?). Esta técnica permite motivar a los integrantes del proyecto mostrando claramente cual es el objetivo del proyecto.
2. Elevator Pitch. Aplicando este método conseguiremos explicar el proyecto de manera breve y clara los aspectos fundamentales del proyecto
3. NOT list. En este caso se trata de destacar aquellas cosas que no serán cubiertas por el proyecto.
4. Show the solution. Haciendo uso de esta técnica se pretende realizar una aproximación al diseño que nos permita recoger en un plano general, a alto nivel, cada uno de los componentes y unidades de proceso en las que se divide el proyecto.
5. What keeps us up at night. Haciendo uso de esta técnica se ponen de manifiesto cuales son las preocupaciones principales referentes al proyecto. Permite detectar riesgos en una etapa temprana.
6. Size it up. Esta casilla del Inception Desk permite estimar qué esfuerzo es necesario para abordar el proyecto.

7. Be clear on what's going to give. Con esta técnica se pondrá de manifiesto que la realización del proyecto siguiendo técnicas ágiles nos permite ser flexibles en el alcance del mismo, fijando la calidad y el tiempo del mismo, dentro de los términos que permite la normativa de la Universidad de Cádiz en la realización del TFM.

8. Show what it's going to take. Se define el equipo y el coste que tendrá el proyecto completando la planificación en términos monetarios.

Haciendo uso de todas estas técnicas se pretende estar fuera de toda duda en cuanto al alcance del proyecto. La planificación y el diseño quedarán definidos y serán el punto de partida necesario para hacer una estimación de coste del proyecto.

El desarrollo de estas técnicas se encuentra en el apéndice A *Iniciación del proyecto*.

2.1.2. Desarrollo de la planificación

El coste de las funcionalidades del sistema está delimitado por el creditaje definido para el desarrollo del presente TFM limitado en 300 horas de trabajo. Las actividades no serán organizadas en torno a un presupuesto monetario concreto. Será el tiempo de dedicación de las mismas las que definan el coste del proyecto.

Los requisitos, repartidos en tareas, serán modelados mediante *Historias de Usuario*. Las Historias de usuario sserán autocontenidas y validadas por el *Product Owner* en cada Sprint. La aproximación del concepto de *Epic* y *Feature* usado en el este proyecto es el propuesta por Dean Leffinegwell en su obra *Agile Requirements* [Leffinegwell, 2010] . Las *Historias de Usuario* formarán el *Product Backlog* del proyecto agrupadas en *Features* que a su vez serán reunidas en *Epics*. En definitiva, existen tres niveles de agrupación para los requisitos del proyecto, desde el más abstracto -la Epic-, hasta el más concreto, la *Historia de Usuario*. Las *Historias de Usuario* a su vez contendrán tareas que describan las actividades a realizar para completar el trabajo que describen. Estas tareas serán definidas en el *Sprint Meeting* que se realizan de cada *Sprint*.

Se generará una *Epic* para la elaboración y gestión de la documentación del proyecto que, aunque no aporta un valor al proyecto final, es fundamental para la superación del TFM buscando la gestión integral de la elaboración de este trabajo.

Se definirá el tamaño de cada historia de usuario usando *Scrum Poker*. Toda historia de usuario que obtenga un valor de más de 20 puntos deberá ser dividida en varias historias de usuario. De cada historia de usuario se definirá el *Who*, *Wants*, *Why/So that* y *Success Criteria* definidos en la metodología Scrum. Una vez conocidos todos estos detalles se priorizarán y se subirán a la herramienta *Taiga* tal como se refleja en la figura 2.2.

Se realiza una adaptación en la metodología ágil. La priorización servirá para secuenciar las actividades dentro de cada *Sprint* en vez de para priorizar las historias de usuario en los *Sprints*. La metodología ágil acepta este tipo de cambios motivados por ser el desarrollo unipersonal del sistema. En nuestros proyecto las *Epics* podrán durar varios *Sprints*.

La priorización de las tareas resuelve la dependencia entre ellas y su secuencialidad. Al ser un proyecto de desarrollo unipersonal no es necesario establecer el grafo de dependencias que resuel-

Figura 2.2: Historia de Usuario en Taiga.

va un posible conflicto entre los integrantes del equipo.

Del mismo modo los roles son usados de una manera particular. Aunque los principios ágiles nos recomiendan la orientación total de las historias de usuarios a los clientes, en este caso, necesitamos que los perfiles de alumno y desarrollador tengan peso en la planificación de las actividades. Existen historias de usuario ligadas a estos dos roles que tienen que ser superadas para poder superar la evaluación del TFM. Deben quedar reflejadas estando dentro del seguimiento y control del proyecto. Por este motivo la definición de usuario usada en este proyecto dentro de las historias no es la conceptualmente aceptada. En este caso definimos valor como toda aquella actividad que colabore a la consecución de una evaluación positiva en el examen de TFM. El usuario que recibe este valor es el alumno, que en este caso es desarrollador y usuario final, por lo que tendremos historias de usuario de desarrollador por los roles que sigue el alumno.

Los criterios de aceptación de las historias de usuario se agrupan en los siguientes tipos:

1. Realización de análisis de alternativas.

2. Desarrollo de pruebas unitarias y de integración.

3. Validación por parte del director del TFM.

Las historias de usuario irán pasando a estado de completadas, reflejando el concepto de *terminado*, una vez hayan superado el criterio de aceptación correspondiente.

2.2. Planificación del proyecto

2.2.1. Planificación de actividades y calendario

En la tabla 2.2 puede verse la planificación del proyecto por sprints y releases. Cada uno de los sprints tiene un lema, reflejado en la columna del mismo nombre, que marca el objetivo principal de cada una de los sprints. Podemos distinguir las diferentes fases en las que se ha dividido del proyecto. Desde el sprint 4 al 6 las actividades principales han sido la implementación y pruebas del producto.

El final de la primera release está marcado como *entregable* al estimarse que en la fecha reflejada deberemos de tener un producto funcionando listo para pasar a un nivel superior de depuración y pruebas.

En la gráfica de *burn down* de la figura 2.3 podemos ver el reflejo de la estimación realizada en la planificación. Incluimos una captura de la gráfica de *burn down* de la plataforma *Taiga* reflejada en la figura 2.4 sobre la cual hemos realizado el seguimiento de la ejecución del proyecto. Esta figura no presenta todos los puntos del proyecto completados porque la redacción de esta memoria está incluida como historia de usuario en las actividades a controlar en la herramienta.

Para realizar la estimación de puntos se ha usado una evaluación de expertos. La metodología usada es *Scrum Poker* recogida en *Agile Samurai* [Ramusson, 2010] con algunas modificaciones. En vez de repetir el proceso de estimación hasta conseguir un valor con el que el equipo esté de acuerdo hemos ofrecido un valor optimista, un valor pesimista y un valor de estimación realista. Para obtener la cifra resultante se ha dado el doble de paso a la estimación realista que a la pesimista y optimista y se ha aproximado al valor más cercano de *Scrum Poker* por exceso.

Planificación detallada

Las actividades de análisis de requisitos, diseño, implementación, pruebas, documentación y finalización recogidas en los sprints de la tabla 2.2 se dividen en historias de usuario incluidas en la tabla 2.3

En la tabla 2.3 se reflejan las historias de usuario indicando el sprint al que están asignadas así como el tamaño de las mismas. La columna de priorización justifica la asignación de una determinada historia de usuario a un sprint concreto.

Existe un tipo de historias de usuario marcadas como tareas *Spike*. Según la referencia [Ramusson, 2010] las historias de usuario de tipo *Spike* son un tipo de tarea habilitadora cuyo objetivo es reducir el riesgo de la solución técnica propuesta. Son tareas que, a priori, no aportan un valor directo al usuario pero que son necesarias para garantizar el éxito del proyecto.

Las actividades propias de las historias de usuario *Spikes* son el análisis de alternativas, la validación de diseños, la investigación y el prototipado.

Leyenda de la tabla 2.3

T Tipo de tarea

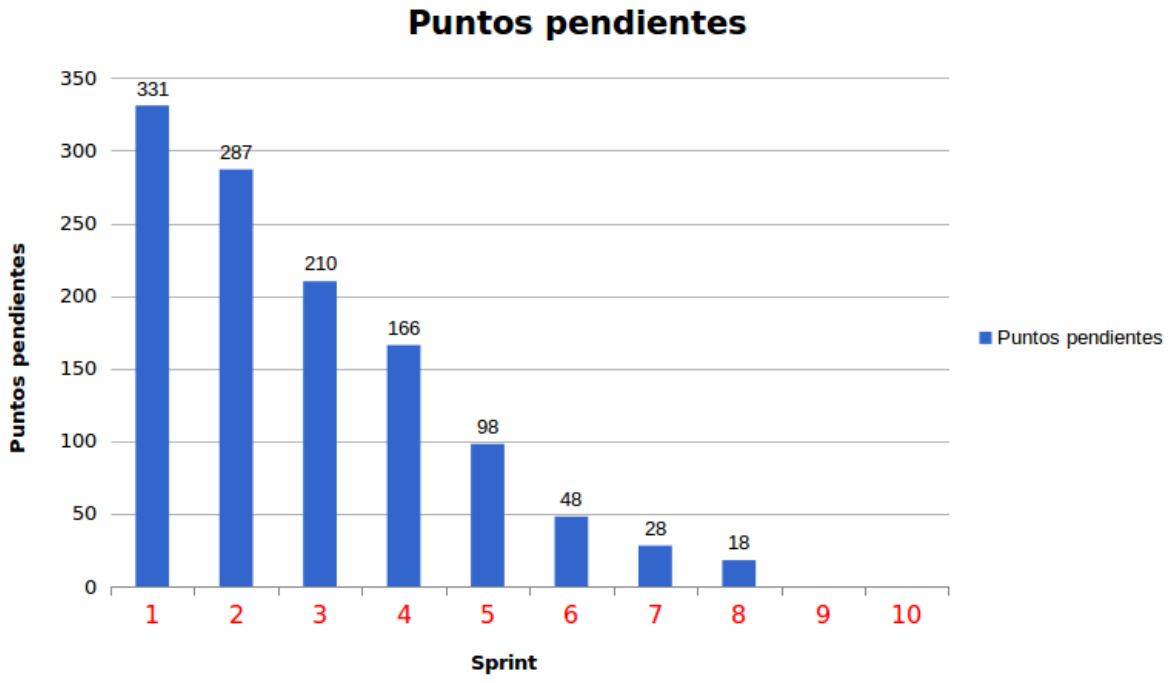


Figura 2.3: Burn down estimado

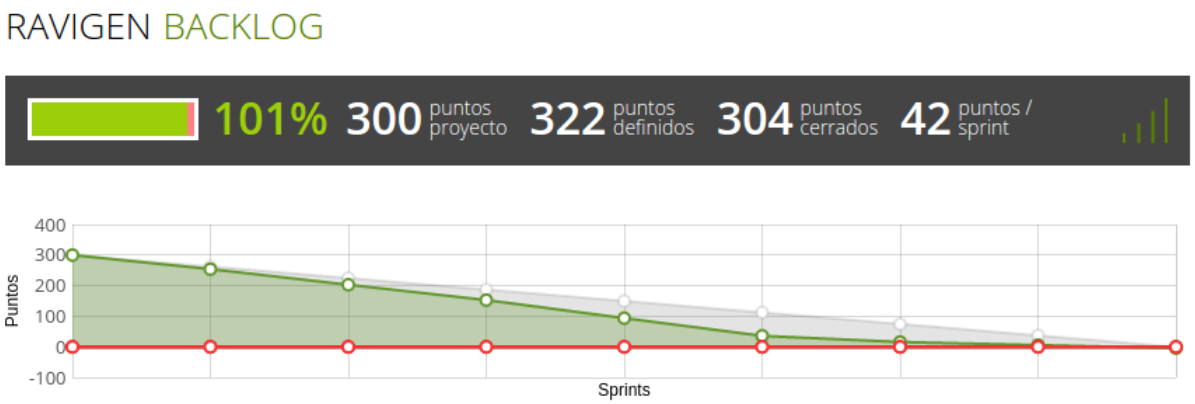


Figura 2.4: Burn down real. Plataforma Taiga

Release	Release Name	Sprint	Lema	Puntos por Sprint	Puntos acumulados	Puntos pendientes	Estimación	Reales	Notas
0	N/A	1	Planficación	44	44	331	01/05/2016	05/05/2016	
0	N/A	2	Análisis	77	121	287	15/05/2016	19/05/2016	
1	Early generator	3	Diseño	44	165	210	01/06/2016	10/06/2016	
1	Early generator	4	Recepción	68	233	166	15/06/2016	25/06/2016	
1	Early generator	5	Envío	50	283	98	15/07/2016	29/07/2016	Releseable
2	Full generator	6	Control	20	303	48	30/07/2016	15/08/2016	
2	Full generator	7	Manuales	10	313	28	15/08/2016	26/08/2016	
2	Full generator	8	Documentación	18	331	18	26/08/2016	09/01/2016	
			Total	331					

Tabla 2.2: Planificación por Sprints y Releases

S Tamaño medido en puntos

P Prioridad relativa

Sp Sprint asignado

K Tarea de tipo Spike

2.2.2. Cierre del Proyecto

En el último Sprint, tras -al menos- 300 horas de trabajo, el tutor del TFM, realizando las tareas propias de *Product Owner*, revisará el trabajo realizado para conceder, si así lo considera, la autorización para presentar el proyecto. La revisión constará de:

1. Lectura de la documentación.
2. Revisión de la presentación.
3. Demostración que ilustre el trabajo realizado.

Una vez recibida la confirmación por parte del director del TFM se procederá a preparar los entregables y a completar las gestiones solicitadas por la normativa de entrega de TFM vigente.

2.3. Organización

En los siguientes apartados se define la relación de roles involucradas en el proyecto y de cómo se estructuran las relaciones entre las mismas para ejecutar el proyecto.

2.3.1. Roles y responsabilidades

Debido a la naturaleza del TFM, se ha representado todos los roles propios del desarrollo ágil de software en diferentes momentos del proceso para así lograr un producto de calidad y siguiendo procesos ágiles de ingeniería, lo cual ofrece un valor añadido al proyecto obteniendo el mayor retorno de la inversión posible.

Product Owner

Representante del dueño del proyecto y usuarios finales del software. Se focaliza en la parte de negocio y el es responsable de la utilidad del proyecto. Traslada la visión del proyecto al equipo, formaliza las prestaciones en historias a incorporar en el *Product Backlog* y las reprioriza de forma regular para conseguir los objetivos a corto plazo.

Scrum Master

Lidera al equipo guiándolo dentro de los procesos de la metodología. Gestiona la resolución de impedimentos del proyecto.

Team

Personal con el conocimiento técnico necesario que desarrolla el proyecto de manera conjunta llevando a cabo las historias a las que se comprometen al inicio de cada sprint. Dentro de las labores del equipo está la realización de las pruebas del producto.

Epic	Feature	Historia de Usuario	T	S	P	Sp	Como	Quiero	Por qué	Criterio de aceptación
Gestión / Documentación	Documentación acorde reglamentación TFM	Definición de Alcance y Objeto		3	1	1	Alumno	Definir el alcance y objetivo del proyecto	Para cumplimentar la documentación necesaria para la asignación de TFM	La asignación de TFM es aprobada por el tribunal competente
Gestión / Documentación	Documentación acorde reglamentación TFM	Plan de desarrollo		8	2	1	Alumno	Cumplir con la ISO 16326 de gestión de proyectos	Para superar los criterios establecidos por el reglamento de TFM	Se elabora un 'Plan de Desarrollo' que cubre todos los apartados del 'Project Management Plan'
Gestión / Documentación	Documentación acorde reglamentación TFM	Plan de pruebas		3	3	2	Alumno	Desarrollar un plan de pruebas	Para garantizar la calidad del producto creado	Se define como hacer las pruebas unitarias y de integración de todos los elementos
Gestión / Documentación	Documentación acorde reglamentación TFM	Planificación		8	2	1	Alumno	Organizar el trabajo a realizar de manera ágil	Para tener una guía y organización del trabajo a realizar	Se completa la planificación en la plataforma Taiga
Gestión / Documentación	Documentación acorde reglamentación TFM	Creación de historias de usuario		10	1	1	Alumno	Dividir, estimar y establecer el criterio de aceptación de los trabajos	Para tener una guía y estimación que acote el trabajo a realizar	Se dan de alta todas las historias de usuario en la plataforma Taiga y se comunica al director del TFM

Gestión / Documentación	Documentación acorde reglamentación TFM	Iniciación del proyecto		10	2	1	Alumno	Delimitar y explicar correctamente de qué se trata el proyecto	Para poder poner el foco en los asuntos fundamentales, revisar los riegos y que el director del TFM y el tribunal de evaluación tenga toda la información necesaria	Elaborar el material siguiendo el Inception Desk propuesto en Agile Samurai
Gestión / Documentación	Documentación acorde reglamentación TFM	Presentación para defensa del proyecto		10	2	8	Alumno	Realizar la presentación del proyecto acorde a la normativa y la rúbrica	Para superar la evaluación el TFM con éxito	Revisar la presentación con el director del TFM
Gestión / Documentación	Documentación acorde reglamentación TFM	Generación de documentación		8	1	8	Alumno	Generar y revisar la documentación acorde a la normativa y la rúbrica	Para superar la evaluación el TFM con éxito	Revisar la presentación con el director del TFM y depositar las copias necesarias en Rodin y en la secretaría
Gestión / Documentación	Documentación acorde reglamentación TFM	Manual de uso		5	1	7	Usuario	Tener una guía de uso de la aplicación	Para poder validar y probar la aplicación correctamente	Se elabora un manual de uso
Gestión / Documentación	Documentación acorde reglamentación TFM	Manual de instalación		5	2	7	Usuario	Tener una guía de instalación	Para poder validar y probar la aplicación correctamente	Se elabora un manual de instalación

Gestión / Documentación	Documentación acorde reglamentación TFM	Adaptación de metodologías ágiles al desarrollo del TFM	K	5	3	1	Alumno	Saber como adaptar el uso de metodologías ágiles a la realización del TFM	Para superar la evaluación el TFM con éxito	Apartado o anexo donde se describa las principales consideraciones del uso de metodologías ágiles en el TFM
RaViGEn Core (RC)	Diseño adaptable	Configuración del entorno de desarrollo y pruebas		10	1	2	Alumno	Implementar y ejecutar el sistema, manteniendo un control de configuración	Para poder desarrollar RaViGEn	El entorno de desarrollo es completo y existe un repositorio del sistema
RaViGEn Core (RC)	Recepción de información DIS	Análisis de estándar DIS	K	15	2	2	Usuario	Generar video radar a partir de las PDUs DIS más relevantes	Para que la señal de video radar sea lo más fidedigna posible	Se elabora un estudio identificando qué PDUs se tendrán en cuenta en la generación, cómo se hará y que objetivo se persigue.
RaViGEn Core (RC)	Recepción de información DIS	Análisis de alternativas de implementaciones DIS	K	10	3	3	Desarrollador	Seleccionar el/los framework(s) de implementación DIS para el proyecto	Para poder recibir las distintas PDUs DIS	Se redacta un documento de análisis de alternativa que recoge los puntos fundamentales de la elección del framework

RaViGEn Core (RC)	Diseño adaptable	Análisis de formatos de video radar	K	10	3	2	Desarrollador	Conocer qué formatos de video radar existen	Para poder realizar el mejor modelo de datos posible	Se realiza una análisis de los formatos existentes que sirva para poder realizar el modelado de datos
RaViGEn Core (RC)	Diseño adaptable	Diseño del Sistema		10	2	2	Desarrollador	Definir el diseño general del sistema	Para tener una visión general que nos permita cumplir con los objetivos	Se realiza un documento de diseño de sistema
RaViGEn Core (RC)	Diseño adaptable	Diseño de RaViGEn Core		8	1	3	Desarrollador	Definir el diseño de RaViGEn Core	Para orientar la implementación del componente principal que nos permita cumplir con los objetivos	Se realiza un documento de diseño de la aplicación
RaViGEn Core (RC)	Diseño adaptable	Diseño del almacenamiento y modelo de datos		8	2	3	Desarrollador	Definir la base de datos de RaViGEn	Para que sea suficiente para la explotación y la depuración del sistema	Se realiza un modelo de datos del sistema
RaViGEn Core (RC)	Recepcion de video radar	Implementación recepción y procesamiento de datos según el estándar DIS		15	1	4	Desarrollador	Recibir datos DIS y tratarlos como indica el estándar	Para poder analizarlos para la generación de video radar asociado	Se realiza una prueba unitaria de la recepción de datos DIS

RaViGEn Core (RC)	Diseño adaptable	Análisis de alternativas para el almacenamiento de datos	K	8	1	4	Desarrollador	Definir qué solución tecnológica se usa para almacenar los datos en RaViGEn	Para que sea suficiente para la explotación y la depuración del sistema	Se selecciona un método para el almacenamiento de datos temporal y persistente
RaViGEn Core (RC)	Diseño adaptable	Modelado de datos para firma radar	K	20	2	4	Usuario	Obtener un video radar fidedigno	Poder adiestrar a las dotaciones	Se elabora un documento de modelado de datos radar
RaViGEn Core (RC)	Generación de video radar	Implementación procesamiento video radarmultiformato		20	1	5	Usuario	Obtener un video radar fidedigno	Poder adiestrar a las dotaciones	Se realiza una prueba unitaria del procesamiento de video radar del sistema en varios formatos
RaViGEn Core (RC)	Generación de video radar	Implementación envío video radarmultiformato		20	2	5	Usuario	Recibir video radar en varios formatos	Poder adiestrar a las dotaciones	Se realizan las pruebas unitarias del envío de video radar del sistema en varios formatos
RaViGEn Core (RC)	Generación de video radar	Control de flujo de envío de datos		10	2	6	Usuario	No saturar la red con envío de video radar	Para que pueda convivir con el resto de sistemas de adiestramiento	Se realizan las pruebas unitarias del control de flujo de envío de datos
RaViGEn Test Console (RTC)	Control y presentación información	Configuración del entorno de desarrollo y pruebas		8	1	2	Desarrollador	Poder probar el sistema	Para que pueda ser validado	El sistema se encuentra operativo y se documenta las decisiones tomadas

RaViGEn Test Console (RTC)	Control y presentación información	Analisis de alternativas de frameworks de presentación de datos geolocalizados y video radar	K	15	3	2	Desarrollador	Poder presentar los distintos ítems de RaViGEn para explotación y pruebas	Para poder validar el generador de video radar	Se documenta la elección del framework para la presentación de entidades DIS y video radar
RaViGEn Test Console (RTC)	Control y presentación información	Diseño de RaViGEn Test Console		10	3	3	Desarrollador	Definir el diseño de RaViGEn Test Console	Para tener una visión que nos permita cumplir con los objetivos	Se realiza un documento de diseño de la aplicación
RaViGEn Test Console (RTC)	Control y presentación información	Elección de entorno de desarrollo	K	3	1	2	Desarrollador	Tener un entorno de desarrollo que permita la elaboración del proyecto	Para usar las herramientas adecuadas	Se documenta la elección de entorno de desarrollo
RaViGEn Test Console (RTC)	Control y presentación información	Implementación presentación video radar		15	3	4	Usuario	Visualizar la presentación de video radar	Comprobar que la solución es la esperada	Se realizan las pruebas unitarias de presentación de video radar
RaViGEn Test Console (RTC)	Control y presentación información	Implementación presentación datos de depuración		10	3	5	Desarrollador	Visualizar la presentación de video radar y de datos DIS	Comprobar que la solución es la esperada	Se realizan las pruebas unitarias de presentación de video radar y datos DIS para depuración

RaViGEn Test Console (RTC)	Control y presentación información	Diseño HMI RaViGEn Test Console		8	3	3	Desarrollador	Realizar el diseño en mockup de la aplicación RTC	Para que pueda ser evaluada	Se realiza un documento de diseño IHM de la aplicación
RaViGEn Test Console (RTC)	Control y presentación información	Implementación controles RTC		10	1	6	Desarrollador	Poder seleccionar qué presenta RTC así como controlar el flujo de video radar	Para cumplir con los objetivos del sistema	Se realizan las pruebas unitarias de presentación de video radar

Tabla 2.3: Historias de Usuario por Epic y Feature.

2.3.2. Recursos y herramientas

Equipo de desarrollo:

- Intel Quad Core q6600
- 4 GiB RAM
- Tarjeta gráfica Nvidia 9500 GT
- Ubuntu 16.04 LTS 64 bits

Herramientas

- Control de versiones
  - Cliente Git 2.7.4. Repositorio en bitbucket.org.
- Diseño
  - Editor de imágenes Aseprite 1.0.9
- Desarrollo C++
  - Qt 4.8.7
  - QtCreator 3.0.1
  - GNU/GCC 5.4.0
  - OpenMP
  - KDIS Framework 2.9.0
  - Geographical Lib 1.46
  - SQLite 3
- Desarrollo Java
  - Eclipse 3.8
  - OpenJDK 8
  - NASA WorldWind SDK 2.0
- Ofimática
  - Kile
  - Tex Live 2015
  - LibreOffice 5

2.4. Costes

En este aparado se ofrece una visión general del coste de los elementos de Scrum punto y *sprint* sirviendo de ejercicio sobre el coste del proyecto desarrollado pudiendo calcular con estos valores el coste de cada una de las fases del mismo. La relación de precios tomada como referencia es [CCOO, 2010].

Concepto	Cantidad	Comentarios
Puntos por Sprint	37,50 puntos	Velocidad del equipo (media)
p/m por Sprint	0,25	Person-Months
Sprints	8,00	
p/m total	2	Person-Months
Capacidad	300,00 puntos	12 créditos
Meses	2,00	Al 100 % de dedicación
Puntos por mes	150,00	Según creditaje
Estimación total	331,00 puntos	
p/m necesarios	2,2	Person-Months
Puntos por mes necesarios	165,50	A tiempo completo
Margen	-15,50 puntos	Sobretrabajo
Coste salario / hora	20,00 €	
Coste salario / año	34000,00 €	
Equipo	1 persona	
Coste equipo total año	34000,00 €	
Coste equipo año	37513,33 €	Costes sociales
Dedicación	1,103	10 % de horas extras
Coste equipo sprint	781,53 €	
Coste equipo mes	3126,11 €	
Coste equipo punto	20,84 €	
Coste equipo proyecto	6252,22 €	
Horas mes	141,67 h	
Horas año	1700,00 h	
Costes indirectos	375,13 €	
Coste asesoría	937,83 €	
Coste total	7565,19 €	

Tabla 2.4: Coste proyecto

2.5. Riesgos

Gestión de riesgos

Las acciones correctoras y los cambios resultantes de la evaluación de riesgos serán propuestos mediante el mecanismo de *Peticiones* dentro de la plataforma *Taiga* donde se gestiona el proyecto. Las peticiones serán clasificadas como *Bug* o *Mejora* según la naturaleza de la misma. En cada reunión de planificación del Sprint se valorarán las *Peticiones* y, si el equipo de gestión lo considera, pasarán a formar parte del *Backlog* del proyecto como acción correctora que mitigue el riesgo identificado.

Presentamos los riesgos en orden de prioridad:

1. Alcance excesivo del proyecto. El estándar DIS tiene una extensión significativa. El tener que analizar este estándar para conseguir un simulador de vídeo radar de una calidad mínima puede exceder fácilmente el alcance propuesto para la realización de un TFM.

2. No disponibilidad de equipo de desarrollo con suficiente potencia. El desarrollo con Java y OpenGL, procesando gran cantidad de información por segundo aconseja la mejora del equipo de desarrollo propio.

3. Calidad del producto. Los problemas propios del desarrollo, los problemas no esperados y que el resultado no sea de una calidad suficiente son riesgos inherentes al problema que se pretende solucionar en este proyecto.

En el mismo orden se presentan las acciones mitigadoras ejercidas:

1. Se seleccionan las metodologías ágiles de desarrollo como elemento vehiculador de este proyecto. Haciendo uso de estas metodologías debemos revisar el alcance del trabajo realizado y, sobre todo, el restante que nos permite decidir seguir con el proyecto o no. Además la eliminación de cualquier tipo de desperdicio o *waste* que propone la metodología *Lean* consigue que el retorno de la inversión se maximice.

2. Se adquiere un equipo económico que permite el desarollo, ejecución y pruebas del proyecto.

3. Evaluar cada sprint el resultado del producto para conseguir las menores desviaciones posibles.

2.6. Aseguramiento de la Calidad

2.6.1. Ejecución. Monitorización y Control

Se define cada Sprint como 37,5 horas de trabajo. Esta definición de Sprint no está acorde con la ofrecida por la biliografía. Tal como se expone en la metodología es una defincón adaptada. Un Sprint es una unidad organizativa de una duración determinada. Debido a la dedicación parcial en el desarrollo del proyecto se expone la necesidad, para facilitar el uso de la metodología, de definir el Sprint en horas. Definir el Sprint como una semana de trabajo nos llevaría a errores en el seguimiento y control del proyecto debido a la irregularidad en la dedicación de horas al proyecto que puede compaginar el alumno.

El primer Sprint recogerá las tareas de planificación del proyecto y el último las tareas propias del cierre y documentación final del mismo.

Para el seguimiento y monitorización del proyecto usaremos la herramienta online para la gestión de proyectos ágiles *Taiga*. El *Scrum meeting* es sustituido por la revisión de la plataforma de manera periódica de la plataforma y la comunicación por parte del alumno de tareas completadas, posibles retrasos e impedimentos.

Al finalizar cada Sprint se realizará una reunión de tipo *Sprint Review (demo)* que permita al tutor del proyecto controlar y validar el avance del mismo. El *Sprint Planning Meeting* será llevado a cabo de manera virtual entre tutor y alumno. En esta reunión se dividiran las historias de usuario en tantas tareas como sean necesarias. El criterio de aceptación será establecido a nivel de historia de usuario.

2.6.2. Lecciones Aprendidas

Todas aquellas lecciones aprendidas relacionadas con cualquier fase del desarrollo del proyecto serán recogidas en la sección "Lecciones Aprendidas" de la herramienta Taiga.

Las lecciones serán recogidas y revisadas en la reunión de retrospectiva de cada Sprint.

Parte II

Desarrollo

Capítulo 3

Requisitos del Sistema

3.1. Situación actual

Se parte de un desarrollo nuevo, con una idea innovadora, que define la situación actual del producto como el punto de partida del mismo.

La solución tecnológica planteada tiene como objetivo cubrir una necesidad de los centros de adiestramiento de los sistemas de mando y control. Estos sistemas son principalmente de propiedad pública. Debido a esta peculiaridad es complicado la modernización de los sistemas planteando la reestructuración de alguno de los servicios.

La necesidad cubierta por RaViGEn podría ser resuelta haciendo una gran inversión. Una nueva red de alta capacidad y un simulador que implemente el estándar DIS por tipo de radar (aéreo, superficie, según banda de escaneo, capacidad de filtrado...) podrían sustituir a este producto. Sólo la implantación de una nueva red LAN para diez equipos podría supera el coste del desarrollo completo de este producto.

Existen simuladores de vídeo radar o vídeo en crudo. La principal referencia en este entorno es la empresa *Cambridge Pixel* que ofrece una amplia gama de productos asociadas con los radares y la vigilancia. La característica fundamental y diferenciadora de RaViGEn es la integración de su simulación al contexto de un escenario coordinado mediante el uso del protocolo DIS.

3.1.1. Entorno Tecnológico

La Escuela de Especialidades Antonio de Escaño es el centro de adiestramiento de sistemas de mando y control de referencia en la Armada Española. Usaremos este centro para contextualizar la necesidad de la herramienta RaViGEn. Este centro dispone de una sala específica de adiestramiento para sistemas de mando y control. Está compuesta por:

- Un puesto de instructor compuesto de dos workstations de arquitectura x86. Una para control del ejercicio y otra para poder instruirse en el mismo ejercicio.
- Dieciséis puestos de adiestramiento con una workstations de arquitectura x86 y un panel táctil.

- Tres servidores de proceso IBM HS-22 donde se ejecuta el sistema de adiestramiento a un veinte por ciento de su capacidad.
- Sistema Operativo Red Hat Linux 5.2 x64
- Red LAN 100 mbps. Switch Enterasys.

La mayoría de los sistemas de simulación de entornos de mando y control están basado en plataformas GNU/Linux. Para hacer más sencilla la implementación de RaViGEn y su exportación a diferentes sistemas y plataformas se ha decidido usar el framework Qt4.

Qt es un framework multipropósito y multiplataforma de desarrollo orientado a objetos que ofrece un amplio conjunto de capacidades para resolver cualquier tipo de problema: gráfico, de conexión de red, de lectura de ficheros...

Nos apoyamos en las librerías KDIS y GeographicalLib para la gestión del tráfico DIS y para realizar los cálculos geodésicos necesarios para el resultado de la simulación realizada por RaViGEn.

Para la aplicación de visualización de vídeo radar, para poder analizar la simulación producida por RaViGEn en el ámbito de las pruebas y la depuración, se ha optado por el framework WorldWind de la NASA. Escrito en Java permite manejar cómoda y rápidamente las particularidades de la presentación de datos geolocalizados con un rendimiento aceptable.

3.1.2. Fortalezas y Debilidades

El sistema de adiestramiento trata de instruir a las dotaciones en sistemas con soluciones propietarias. El uso de estándares abiertos proporcionaría, con una inversión controlada, mayores capacidades a estos centros. RaViGEn trata ser un primer paso, que pueda marcar tendencia aprovechando los formatos y estándares abiertos, para solucionar un problema de alto coste manteniendo la arquitectura actual de los centros de formación.

Como debilidad principal del producto, en esta primera versión, podemos considerar que no es un simulador completo que permita su integración total en herramientas de simulación de terceros.

3.2. Necesidades de Negocio

3.2.1. Objetivos de Negocio

El objetivo de negocio a alcanzar con RaViGEn consiste en la implantación, de manera integrada, de un sistema de simulación de vídeo radar, configurable y extensible, que permita a los centros de adiestramiento inyectar señal radar en un ejercicio coordinado mediante el estándar DIS.

El coste de la implantación será sólo el propio del desarrollo y despliegue de la herramienta ya que, por definición, se adapta a la arquitectura y configuración disponible en los distintos centros de adiestramiento.

3.2.2. Procesos de Negocio

Los centros de adiestramiento actuales permiten a los operadores de los sistemas de mando y control adiestrarse en una simulación coordinada donde los distintos sistemas y sensores son simulados. Por motivos económicos, debido a la necesidad de equipos e implantación de nueva infraestructura, la señal de vídeo radar no es simulada.

RaViGEn permite que el adiestramiento pueda utilizar la misma metodología de enseñanza con el añadido de disponer de la señal de vídeo radar sin tener que hacer la inversión en nuevos equipos e infraestructura. RaViGEn se alimenta del mismo escenario simulado y coordinado que se reproduce en la actualidad para producir señal de vídeo radar presentable por el sistema.

3.3. Objetivos del Sistema

Los objetivos principales del producto RaViGEn son los definidos por su componente central, llamado *RaViGEn Core*, y son distribuidos por epics, features e historias de usuario según la granularidad de los mismos:

- **OBJ_01 Generar señal de vídeo radar a partir de información DIS (Antonio García). Prioridad Alta.** Disponer de una solución software, abierta a la extensión, que sea capaz de simular diferentes formatos de vídeo radar digital a partir de información proporcionada haciendo uso del estándar DIS.

- **OBJ_02 Control de flujo (Antonio García). Prioridad Alta.** Ser capaces de integrar la señal de vídeo radar digital en una red convergente haciendo uso de un ancho de banda limitado.

- **OBJ_03 Extensibilidad (Antonio García). Prioridad Media.** Ofrecer la capacidad de incorporar los distintos modelos de plataformas definidas en el estándar DIS consiguiendo que la firma radar presentada esté acorde con la naturaleza del contacto generado en la simulación.

- **OBJ_04 Realismo (Antonio García). Prioridad Media.** Ser capaces de incluir ruido en el vídeo radar generado.

- **OBJ_05 Tipología radar (Antonio García). Prioridad Media.** Poder generar vídeo de radares de superficie.

- **OBJ_06 Ofrecer un mecanismo de configuración (Antonio García). Prioridad Media.** Disponer de un API para poder realizar la integración de la configuración del producto.

3.4. Catálogo de Requisitos

Esta sección contiene la descripción del conjunto de requisitos específicos del sistema a desarrollar.

3.4.1. Requisitos funcionales

Los requisitos funcionales se agrupan, principalmente, en dos epics: RaViGEn Core y RaViGEn Test Console. Estas epics se dividen en funcionalidades entre las que destacamos: Recepción de información DIS, Generación de vídeo radar y Control y presentación de la información.

Los requisitos son reflejados en las historias de usuario de la tabla 2.3. La trazabilidad sobre los objetivos de los requisitos funcionales es la siguiente:

OBJ_01 Generar señal de vídeo radar a partir de información DIS

Requisito	Historias de usuario
RQ-01-F Procesamiento del estándar DIS	Análisis de estándar DIS Análisis de alternativas de implementaciones DIS Diseño del Sistema Diseño de RaViGEn Core Diseño de RaViGEn Test Console Implementación recepción y procesamiento de datos según el estándar DIS Análisis de alternativas para el almacenamiento de datos
RQ-02-F Generación de vídeo radar a partir de datos DIS	Análisis de formatos de video radar Diseño del Sistema Diseño de RaViGEn Core Diseño de RaViGEn Test Console Diseño del almacenamiento y modelo de datos Implementación recepción y procesamiento de datos según el estándar DIS Análisis de alternativas para el almacenamiento de datos Modelado de datos para firma radar
RQ-03-F Presentar información DIS y de vídeo radar	Diseño del Sistema Diseño de RaViGEn Test Console Diseño del almacenamiento y modelo de datos Implementación recepción y procesamiento de datos según el estándar DIS Análisis de alternativas para el almacenamiento de datos Modelado de datos para firma radar Implementación procesamiento vídeo radar multiformato

Requisito	Historias de usuario
	Analisis de alternativas de frameworks de presentación de datos geolocalizados y video radar Diseño de RaViGEn Test Console Implementación presentación vídeo radar Implementación presentación datos de depuración Diseño HMI RaViGEn Test Console Implementación controles RTC

OBJ_02 Control de flujo

Requisito	Historias de usuario
RQ-05-F Seleccionar cota de consumo de red	Control de flujo de envío de datos Diseño del Sistema Diseño de RaViGEn Core Test Console Diseño HMI RaViGEn Test Console Implementación controles RTC

OBJ_04 Realismo

Requisito	Historias de usuario
RQ-07-F Seleccionar cantidad y tipo de ruido	Implementación envío video radarmultiformato Diseño del Sistema Diseño de RaViGEn Core Diseño de RaViGEn Test Console Diseño HMI RaViGEn Test Console
RQ-08-F Procesar las emisiones electromagnéticas simuladas	Análisis de estándar DIS Diseño del Sistema Diseño de RaViGEn Core Implementación envío vídeo radarmultiformato

OBJ_05 Tipología radar

Requisito	Historias de usuario
RQ-10-F Seleccionar tipo de radar	Diseño del Sistema

Requisito	Historias de usuario
	Diseño de RaViGEn Core
	Diseño de RaViGEn Test Console
	Implementación controles RTC

OBJ_06 Ofrecer un mecanismo de configuración

Requisito	Historias de usuario
RQ-11-F Presentar vídeo radar	
	Diseño del Sistema
	Diseño de RaViGEn Core
	Diseño de RaViGEn Test Console
	Implementación controles RTC
RQ-12-F Presentar datos DIS	
	Diseño del Sistema
	Diseño de RaViGEn Core
	Diseño de RaViGEn Test Console
	Diseño HMI de RaViGEn Test Console
	Implementación controles RTC
RQ-13-F Presentar controles de visualización	
	Diseño del Sistema
	Diseño de RaViGEn Core
	Diseño de RaViGEn Test Console
	Diseño HMI de RaViGEn Test Console
	Implementación controles RTC

3.4.2. Requisitos no funcionales

Los requisitos no funcionales son reflejados en las historias de usuario recogidas en la tabla 2.3 .

Al igual que los requisitos funcionales, los requisitos no funcionales se agrupan, principalmente, en dos epics: RaViGEn Core y RaViGEn Test Console. Estas epics se dividen en funcionalidades entre las que destacamos el diseño adaptable. Además la historia de usuario *Control de flujo de envío de datos* es parte fundamental de este proyecto y de la solución que al cliente ofrece RaViGEn.

La trazabilidad sobre los objetivos de los requisitos no funcionales es la siguiente:

OBJ_02 Control de flujo

Requisito	Historias de usuario
RQ-04-NF Flujo de datos enviados por red	
	Control de flujo de envío de datos
	Diseño del Sistema

Requisito	Historias de usuario
	Diseño de RaViGEn Core Implementación envío vídeo radarmultiformato

OBJ_03 Extensibilidad

Requisito	Historias de usuario
RQ-06-NF Aceptar nuevos modelos de datos de entidades	Análisis de estándar DIS Análisis de formatos de video radar Diseño del Sistema Diseño de RaViGEn Core Diseño del almacenamiento y modelo de datos Modelado de datos para firma radar Implementación procesamiento video radarmultiformato Implementación envío video radarmultiformato

OBJ_05 Tipología radar

Requisito	Historias de usuario
RQ-09-NF Aceptar nuevos formatos de vídeo	Diseño del Sistema Diseño de RaViGEn Core

3.4.3. Requisitos de información

En esta sección se describen los requisitos de gestión de información que el sistema debe gestionar.

La fuente principal de información es el estándar DIS [NATO Standardization Agency, 1996] donde se describen todas los mensajes a tratar, procesar y almacenar como entrada de RaViGEn. En el apéndice B se hace un estudio pormenorizado de cada una de las unidades de datos a procesar por el producto que presenta este proyecto. Como conclusión destacar la influencia de la Entity State PDU en la simulación de vídeo radar que nos permite conocer la posición, orientación y categorización de cualquier plataforma recogida por el estándar.

La salida producida por RaViGEn se ajusta los formatos de vídeo radar ofrecidos por la industria. En el apéndice C se ofrece una análisis de los formatos comunes en la industria. Como conclusión destacar la abstracción de estos formatos dentro de RaViGEn para ser compatible con cualquier salida deseada.

3.5. Alternativas de Solución

3.5.1. OpenMP

Seleccionamos OpenMP para paralelizar secciones de manera sencilla. En RaViGEn existen numerosas secciones sobre conjuntos de datos que no dependen una iteración sobre la siguiente. Este diseño facilita el uso de esta tecnología.

El aumento del rendimiento conseguido con el uso de esta tecnología es relevante. El que esta tecnología mejore la eficiencia del sistema permite procesar un mayor número de entidades. RaViGEn podrá tratar un mayor número de plataformas dentro del alcance radar.

3.5.2. Frameworks DIS

El componente DIS es fundamental para el desarrollo de RaViGEn. El núcleo de la aplicación basa su capacidad de integración en el uso de este estándar.

Por este motivo es necesario hacer un análisis de alternativas exhaustivo. El objetivo de este estudio será elegir el mejor framework de desarrollo para el cumplimiento del estándar DIS.

En el caso de que ninguna alternativa obtenga la nota suficiente será responsabilidad de este proyecto implementar el estándar.

El desarrollo del análisis de alternativas queda recogido en el apéndice D

3.5.3. Sistemas de presentación de información geoposicionada y video radar

RaViGEn Test Console es un componente auxiliar del sistema. Desde esta aplicación veremos el resultado de los cálculos y de la simulación realizada por RaViGEn.

La elección de la herramienta de apoyo a estas labores es WorldWind de la NASA. Al ser un componente secundario y cumplir con todos los requisitos definidos se plantea que no es necesario un análisis de alternativas exhaustivo. El retorno de la inversión no sería valioso.

3.5.4. Qt

Qt es un framework multiplataforma orientado a objetos ampliamente usado para desarrollar programas (software) que utilicen interfaz gráfica de usuario, así como también diferentes tipos de herramientas para la línea de comandos y consolas para servidores que no necesitan una interfaz gráfica de usuario.

Aporta técnicas, como el manejo de señales y slots, que facilitan el desacople de componentes.

3.5.5. C++

C++ es un estándar de facto en la industria de los sistemas de mando control. En el caso de RaViGEn se necesita un lenguaje que ofrezca un alto rendimiento y cierto grado de retrocom-

patibilidad. Los sistemas del centro de referencia seleccionado para este proyecto, Antonio de Escaño, hace uso del sistema operativo Red Hat Linux 5.0.

Para que RaViGEn sea un producto que pueda ser instalado en estos centros necesita restringirse a tecnología compatible con la infraestructura de este tipo de centros. Por este motivo no hacemos uso de tecnología que limite su implantación. Debido a este asunto la versión de C++ usada en este proyecto es la ISO/IEC de 1998 disponible en el compilador GNU/GCC 4.1.2.

El uso de Qt sobre C++ ofrece las extensiones necesarias para facilitar las pruebas y el desarrollo en este lenguaje. Debido a la limitación del uso de compilador y lenguaje utilizaremos Qt4 en vez de Qt5.

3.5.6. Java

Es un lenguaje de propósito general orientado a objetos. La sencillez del manejo de la concurrencia en este lenguaje nos permite desarrollar el componente secundario RaViGEn Test Console con cierta agilidad.

La gran cantidad de componentes existentes para este lenguaje, sumada al uso de WorldWind, permite una gran versatilidad en el desarrollo.

3.6. Solución Propuesta

Se opta por el uso de Qt como software base, en su licencia libre, que nos permite limitar el coste del proyecto y aportar a la comunidad de Qt aquellas incidencias encontradas durante el desarrollo del producto.

Usamos WorldWind como base de software base, en su licencia NASA Open Source, que nos permite su uso sin limitaciones para la naturaleza de este proyecto.

Usamos KDIS como framework de tratamiento de datos DIS que, bajo su licencia libre, nos permite no tener que desarrollar el estándar IEEE que define este protocolo puedo aumentar el retorno de la inversión del tiempo en este proyecto.

Capítulo 4

Análisis del Sistema

4.1. Modelo Conceptual

En este apartado se incluyen los puntos más destacados del diseño. Se dispone de una documentación detalla en el apéndice F de este proyecto.

Se usa el diagrama de alto nivel de la figura 4.1 para introducir los conceptos básicos que, conceptualmente, componen RaViGEn. El modelo sigue las recomendaciones del IEEE en el 1278-3 [NATO Standardization Agency, 1996].

Se trata de una aplicación principal que hace uso de clases que encapsulan la implementación de *worker threads* cada una con un propósito bien definido. Cada uno de estos *trabajadores* tienen su propio bucle principal para que sean capaces de realizar las tareas que tienen encomendadas:

- DISReceiver es el encargado de recibir el tráfico DIS. Inicializa el lector UDP multicast, decodifica los mensajes recibidos y los almacena en la base de datos del sistema.
- ReturnsGenerator tiene como función principal generar los radios de la simulación radar afectados por un generador de ruidos. La generación está compuesta de dos pasos principales. El primer paso consta de la construcción de la escena de simulación y la segunda de la extracción de dicha escena de los elementos necesarios para generar los radios de emisiones a ser enviados por ReturnsSender. Además los radios son generados con ruido incluidos por el generador de ruido seleccionado como configuración en el sistema.
- ReturnsSender procesa la información generada por ReturnsGenerator y la serializa en el formato de vídeo radar elegido dentro del catálogo disponible según se haya configurado el sistema. Obtiene los datos de los radios generados en el paso anterior, radios que responden a un modelo genérico de formato radar, para adaptarlo al formato real elegido para desarrollar la simulación. Existe un componente, ThroughputControl, que controle la cantidad de flujo de datos transmitida por red.

La extensibilidad es un requisito del proyecto. Tras varias iteraciones acotadas por el uso de las metodologías ágiles de desarrollo se ha conseguido un producto extensible. En este afán cabe destacar las siguientes clases:

- Noise Generator. Es el interfaz que permite generar cualquier tipo de ruido dentro de la simulación. Para que una simulación radar sea fidedigna es fundamental la generación de ruido. La elección radar se realiza haciendo uso del patrón factory que responde a la configuración del sistema.
- Double buffer. Mediante esta clase se resuelve el problema clásico del productor-consumidor en el desarrollo de simuladores.
- DataReaders y DataWriters. Bajo estos nombres se implementan interfaces genéricas que permiten guardar o leer cualquier parte del proceso en diferentes formatos. Por ejemplo DataUDPWriter y DataDBWriter cumplen el mismo interfaz. Con el mismo tipo de objeto genérico es posible mandar la información por mensaje o almacenarla en la base de datos.

La aplicación principal inicializa todos los parámetros de configuración del radar a simular. Elige los formatos de producción de la simulación y de envío a la red. Además permite elegir la calidad de la simulación y el ruido generado.

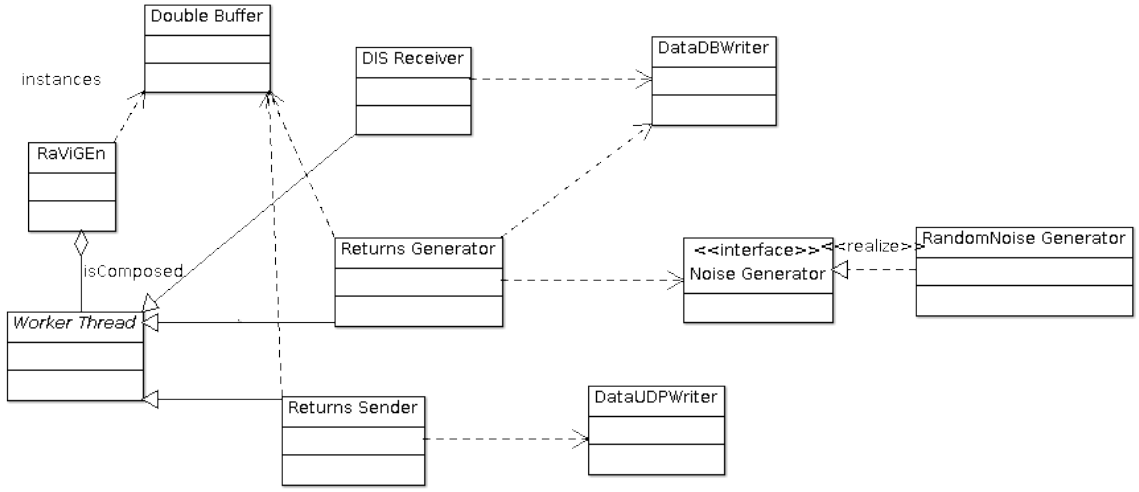


Figura 4.1: Diagrama de clases RaViGEn

4.2. Modelo de Casos de Uso

Dentro de las actividades definidas para este TFM no se considera como elementos de valor la realización de este modelo. Decidimos su realización, dentro del proceso de ingeniería para facilitar la compresión, el mantenimiento y la extensión del producto RaViGEn.

4.2.1. Actores

Existen tres actores principales: la simulación, el tiempo y el generador de vídeo radar.

Actor	Descripción
Simulación	Este rol es externo al sistema. El rol es soportado por la aplicación *Simulation Application DIS* que genera el escenario coordinado.
Tiempo	Este rol es llevado a cabo por el sistema. Consiste en el tiempo de revolución o ciclo radar que condiciona la simulación del mismo.
Generador	Este rol es responsabilidad del propio sistema. Efectúa el trabajo propio de generar la señal radar necesaria para la simulación producida por RaViGEn.

Tabla 4.1: Actores

4.2.2. Casos de uso

A continuación de describen los principales casos de uso del producto RaViGEn Core.

Identificador	UC_01
Nombre	Inicio simulación radar
Descripción	Inicio de la simulación de vídeo radar
Actor	Simulación
Precondiciones	Una *simulation application* debe estar generando un escenario DIS
Postcondiciones	RaViGEn produce simulación de vídeo radar
Flujo principal	1. El tráfico de la simulación es recibido. 2. Se almacena en la base de datos. 3. La simulación comienza.
Flujo alternativo	-
Inclusiones	-
Exclusiones	-

Identificador	UC_02
Nombre	Simular
Descripción	RaViGen ejecuta la lógica necesaria para generar vídeo radar
Actor	Tiempo, Simulador
Precondiciones	La simulación está en marcha
Postcondiciones	La simulación es recibida en un cliente multicast

Flujo principal	1. Se revisa si hay plataformas recibidas. 2. El simulador procesa las plataformas dentro del alcance radar. 3. Se crea una escena colocando cada plataforma en su lugar. 4. Se secciona la escena. 5. Se almacena la escena en un formato radial. 6. Se transforman los radios a un formato de vídeo radar específico. 7. Se mandan los radios producidos por red. 8. Se comprueba la cantidad de tráfico enviada. 9. Se repite la operación cada t segundos.
Flujo alternativo	1. b. No hay plataformas recibidas. Se espera un ciclo. 2. b. No hay plataformas en rango. Se espera un ciclo.
Inclusiones	-
Exclusiones	-

Identificador	UC_03
Nombre	Control de flujo
Descripción	RaViGen ejecuta la lógica necesaria para controlar el flujo de datos insertado en la red
Actor	Simulador
Precondiciones	La simulación está en marcha
Postcondiciones	La simulación es recibida en un cliente multicast

Flujo principal	1. Se inicia el temporizador de control de flujo. 2. Se comprueba si se ha excedido el flujo establecido. 3. Se incrementa si es necesario el número de ocurrencias de exceso de flujo. 4. Si el número de ocurrencias supera el número por umbral de tiempo se disminuye la cantidad de tráfico enviado por red.
Flujo alternativo	▪ 2a. No se ha excedido. ▪ 2b. Si se ha excedido se inicia un temporizador para . medir los excesos en un umbral de tiempo.
Inclusiones	-
Exclusiones	-

4.3. Modelo de Interfaz de Usuario

El componente que forma el núcleo de simulación RaViGEn, o RaViGEn Core, es el componente esencial del sistema. El proyecto se apoya en una herramienta auxiliar, RaViGEn Test Console, para validar la simulación de vídeo radar realizada por el elemento principal. En este apartado se presenta el interfaz hombre-máquina para dicho componente.

En la figura 4.2 se muestra la disposición general de la herramienta. Consta de dos zonas bien delimitadas. Una primera, situada a la izquierda, donde se situan las herramientas de ayuda a la presentación. La segunda, situada a la derecha, que muestra un mapa que sirve como elemento georeferenciador para la presentación de vídeo radar.

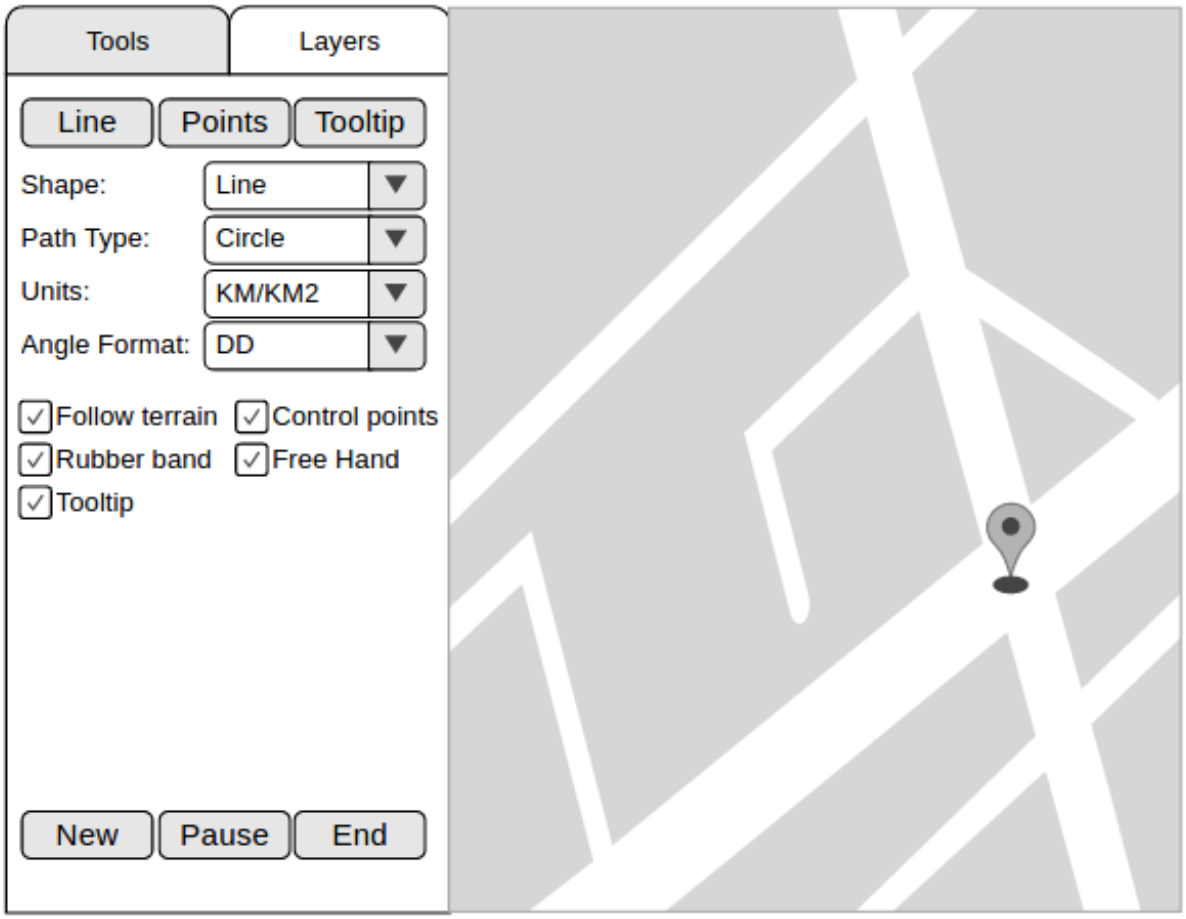


Figura 4.2: Disposición de RaViGEn Test Console

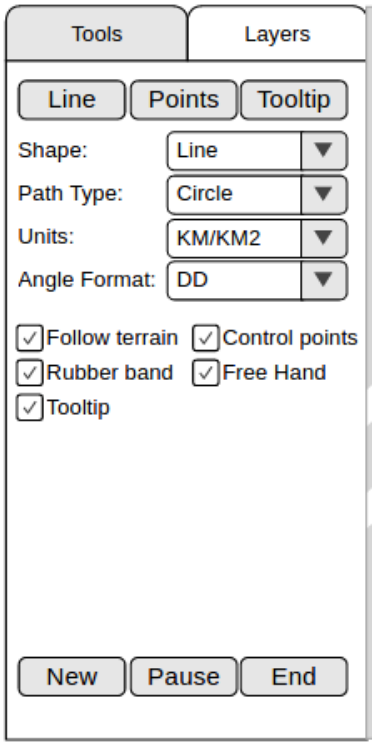


Figura 4.3: Control de utilidades de medición

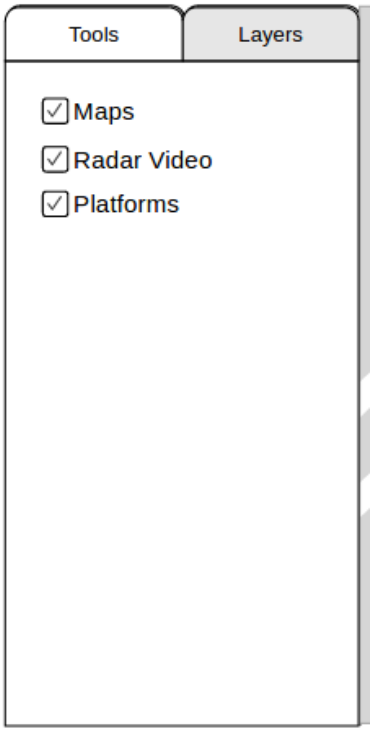


Figura 4.4: Control de presentación

Capítulo 5

Diseño del Sistema

5.1. Arquitectura del Sistema

RaViGEn Core, el componente principal del proyecto, responde al diseño para simuladores propuesto por la especificación DIS [NATO Standardization Agency, 1996] donde, la idea general es, frente a la estimulación que produce un evento -como por ejemplo, que llegue un temporizador a un valor concreto-, produce una salida que imita a los valores producidos por un sistema real [Hodson and Gehl, 2008].

En este caso el principal estímulo es la recepción de tráfico DIS, concretamente de la representación que hace este protocolo de las plataformas, que desembocan en la producción de la simulación del vídeo radar en crudo. Esta recepción es realizada en un ciclo de programa mientras que otros dos hilos de ejecución van realizando en paralelo trabajo asociado a la simulación de vídeo radar. El segundo de los hilos se dedica a generar vídeo radar. El sistema genera una vuelta de radar adelantada para que el tercer hilo de ejecución sea capaz de enviar una vuelta radar completa sin interrupciones ni parones. El objetivo principal de la generación adelantada es conseguir una simulación fluida.

RaViGEn Test Console es un componente del sistema destinado a la verificación y pruebas. Se encarga de la presentación de vídeo radar, geolocalizada, para asegurar que los resultados obtenidos son los esperados. Este componente genera la imagen radar a una frecuencia definida por el usuario.

Plaform Simulator es una aplicación desarrollada para poder inyectar escenarios simulados en el sistema RaViGEn para pruebas. A partir de parámetros definidos en ficheros de configuración crea y realiza el movimiento de las plataformas en función del tiempo.

5.1.1. Arquitectura Física

El sistema se despliega en un cliente sin necesidad de un hardware concreto. Gracias al uso de entornos de trabajo multiplataforma está preparado para ser ejecutado en cualquier entorno. Las pruebas han sido realizadas en entornos GNU/Linux por lo que se recomienda un equipo con Ubuntu 16.04 Desktop. El equipo recomendado es el siguiente:

- PC, con procesador Intel Core i5 o equivalente, al menos 4GiB de RAM y tarjeta gráfica Nvidia.

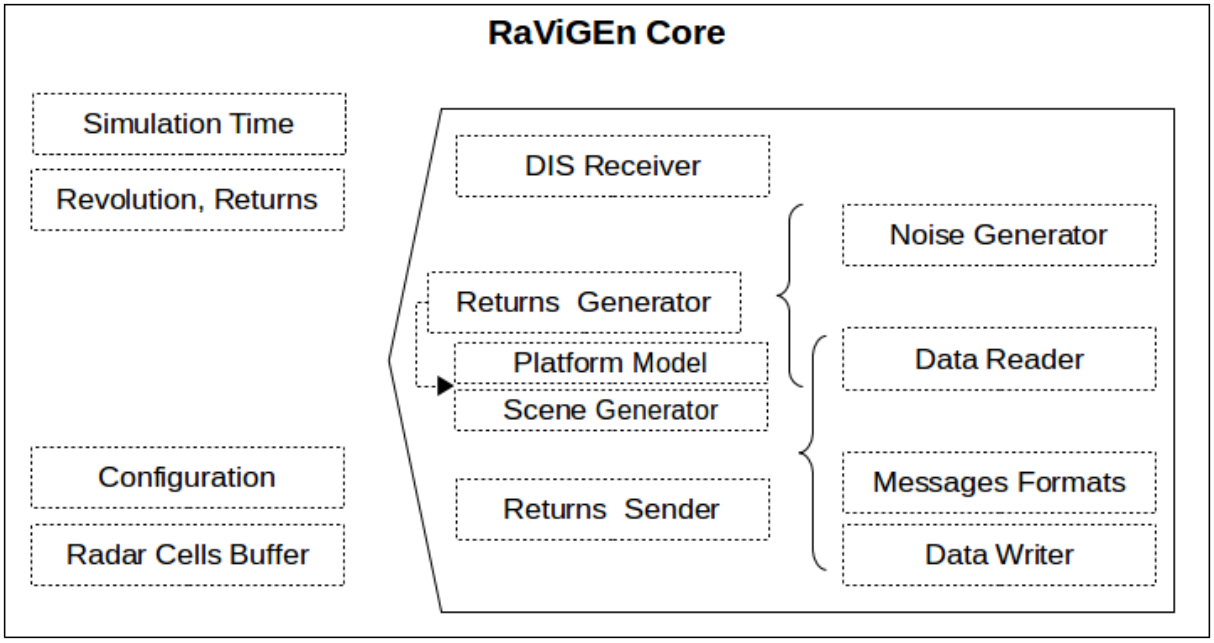


Figura 5.1: Arquitectura de RaViGEn Core

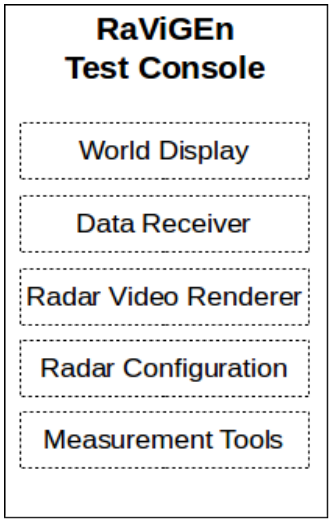


Figura 5.2: Arquitectura de RaViGEn Test Console

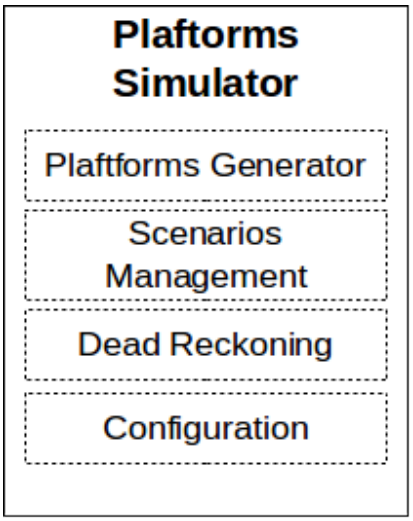


Figura 5.3: Arquitectura de Plaftorms Simulator

- Sistema operativo Ubuntu 16.04 Desktop 64 bits
- OpenJDK con soporte para Java en su versión 7

Para el despliegue en un entorno GNU/Linux es suficiente con ejecutar el siguiente comando dentro del directorio de instalación de RaViGEn:

```
$ ./bin/RaViGEn
```

Si se desea ejecutar el visualizador de video radar es suficiente con ejecutar el siguiente comando dentro del directorio de instalación de RaViGEn:

```
$ java -jar ./bin/RaViGEnTestConsole
```

En un equipo con Ubuntu 16.04 Desktop con una instalación estándar no es necesario ningún servicio adicional.

5.1.2. Arquitectura Lógica

La principal diferencia entre el diseño de una aplicación convencional y el de un simulador radica en la gestión de sucesos o eventos que se realiza de manera continuada. Mientras que, en las aplicaciones de sistemas de la información, el sistema está esperando la interacción del usuario en el caso de los simuladores la aplicación está siempre procesando datos en un ciclo principal realizando cálculos y ajustes que proporcionen al usuario la experiencia más cercana a la realidad posible.

El diseño de RaViGEn recoge una estructura básica bien definida:

1. **Inicialización**. La simulación se sitúa en su estado inicial procesando los parámetros de configuración e inicializando los recursos necesarios.

2. **Ciclo de simulación**. RaViGEn posee un ciclo principal basado en tres hilos de trabajo con, a su vez, un ciclo de proceso cada una. Estos ciclos están siempre en ejecución desarrollando la simulación:

   - El primero de ellos recibe las plataformas que generarán la señal radar.
   - El segundo de los hilos de trabajos es el encargado de generar la señal radar -figura 5.5- a partir de la construcción de una escena como la mostrada en la figura 5.4.
   - El tercero recoge la información, la serializa, y la manda en el tiempo establecido por la configuración radar.

3. **Finalización**. Una vez terminada la simulación la aplicación libera los recursos y sale ordenadamente.

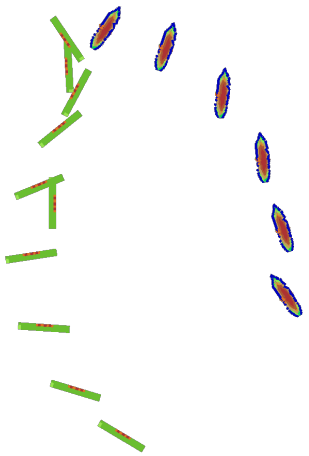


Figura 5.4: Escena. Diez buques y seis detecciones radar.

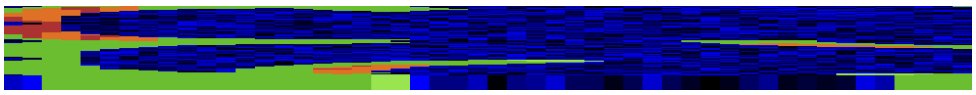


Figura 5.5: Señal radar sintetizada. Diez buques y seis detecciones radar.

Estamos ante un modelo arquitectónico de dos capas. La capa de persistencia no es necesaria en este producto. La información utilizada es eminentemente volátil siendo generada en tiempo real. No se contempla el almacenamiento de datos para simulaciones futuras. La base de datos utilizada en la aplicación se almacena en memoria temporal y sólo conserva información durante la ejecución del sistema.

Capa de presentación (frontend) La capa de presentación de RaViGEn es una herramienta auxiliar que nos permite visualizar el resultado producido por el simulador. Recoge los eventos producidos por el usuario y reacciona ante ellos. Recoge un diseño análogo al núcleo de simulación aunque de menor complejidad.

Capa de negocio Podemos considerar la capa de negocio al componente RaViGEn Core. Es el demonio encargado de la generación del producto, en este caso vídeo radar simulador, y aglutina todos los procesos de negocio del producto.

5.2. Diseño detallado de Componentes

En este apartado se enumeran los aspectos más importantes del diseño del sistema. El diseño detallado de componentes se encuentra en el anexo F de este proyecto. En la documentación detallada se usa el idioma inglés para que, una vez liberado el proyecto a la comunidad, pueda recibir aportes de un mayor número de colaboradores.

El diseño de RaViGEn tiene como elementos generales los siguientes aspectos [Eckel, 2000]:

- El uso de composición sobre herencia.
- La generalización de todos los elementos seleccionables.
- La disponibilidad de clases *Factory* para la creación de diferentes tipos de objetos.

Todo ello para garantizar una cohesión y acoplamiento adecuados que permitan la reutilización, extensión y mantenibilidad del producto.

5.2.1. RaViGEn Core

Uno de los principales objetivos de este proyecto es la capacidad de ampliación del catálogo de formatos de vídeo radar soportados. Con este fin se implementa la clase RadarVideoFormat que generaliza los formatos de codificación de vídeo radar. En la figura 5.7 podemos ver un caso concreto de especificación. Para implementar el formato Asterix 240 se hereda de la clase base y se implementan los mensajes necesarios. Además, esta generalización, facilita la inclusión del factory que discrimina el formato a utilizar por el producto. En la figura 5.6 se refleja el grado de complejidad del formato de vídeo radar.

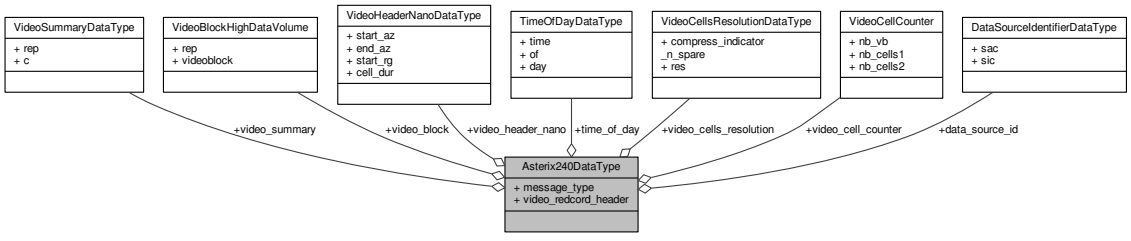


Figura 5.6: Tipo de datos Asterix 240

Un radar realiza barridos en ciclos de 360 grados conocidos como revoluciones. Se modela la revolución radar mediante la clase Revolution mostrada en la figura 5.8.

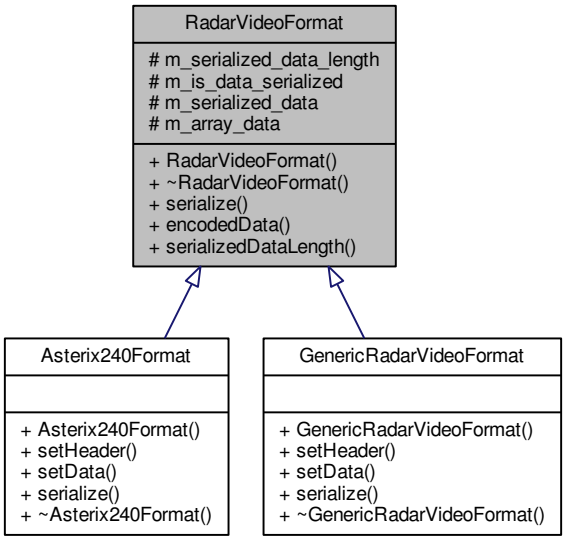


Figura 5.7: Clase RadarVideoFormat

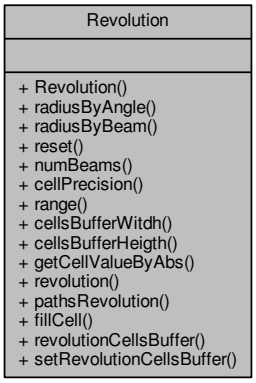


Figura 5.8: Clase Revolution

La precisión mínima angular de un radar es conocida como radio o beam. Cada uno de esos radios -figura 5.10 [United States Naval Academy, 1958]- presenta a distintas distancias un nivel de intensidad conocido como retorno o intensidad. La distancia o alcance máximo del radar se conoce como rango.

Se modela el radio en la clase Radius de la figura 5.9. El radio está definido por un ángulo inicial y un ángulo final. Además se puede definir su distancia de inicio con respecto al origen mediante el uso del método *setRange*.

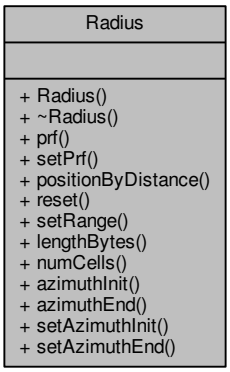


Figura 5.9: Clase Radius

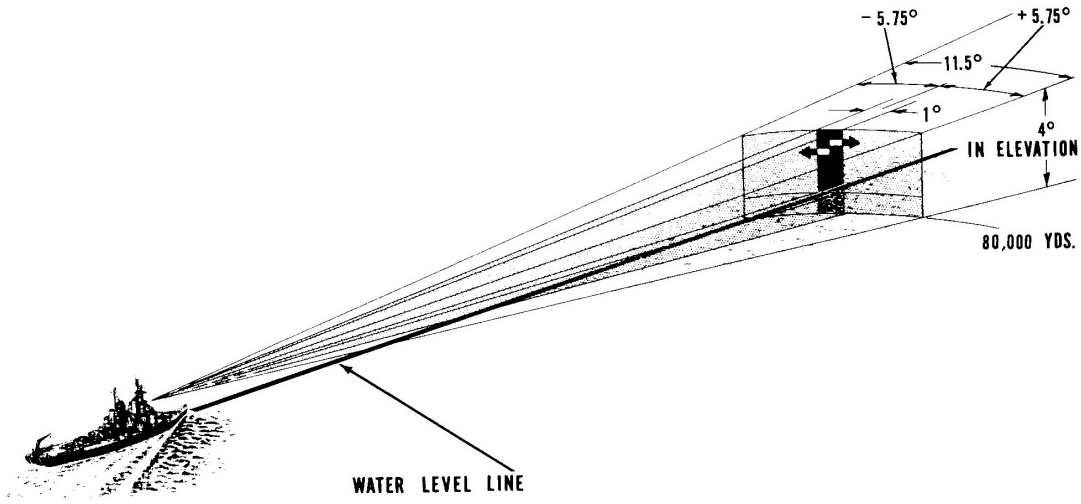


Figura 5.10: Parámetros radar. Representación de un radio.

Es común que los formatos que definen los mensajes que distribuyen la señal detectada necesiten más de una instanciación para mandar el rango total de detecciones de un sólo radio. Se define la clase RadarVideoBuffer 5.11 para generalizar el contenedor de los diferentes mensajes que

componen un sólo radio.

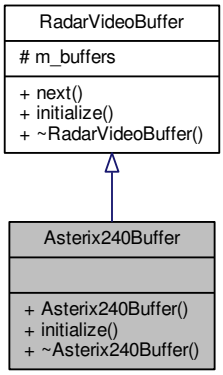


Figura 5.11: Clase RadarVideoBuffer.

La celda es la unidad de representación de la detección radar. En una misma celda, con un valor de intensidad, se recoge la información que, acotada por dos ángulos y dos distancias, se encuentra en la realidad. Por ejemplo si la celda radar tiene un arco de 5 metros será capaz de representar, usando la mejor de sus precisiones, objetos de tamaño de 5 x 5 tal como puede verse en la figura 5.12. En 5 x 5 metros en la realidad puede haber mucha información pero el radar sólo devolverá un valor. La técnica más común es recibir el mayor de los valores de intensidad de dicha realidad [J.-F. Nouvel, 2004].

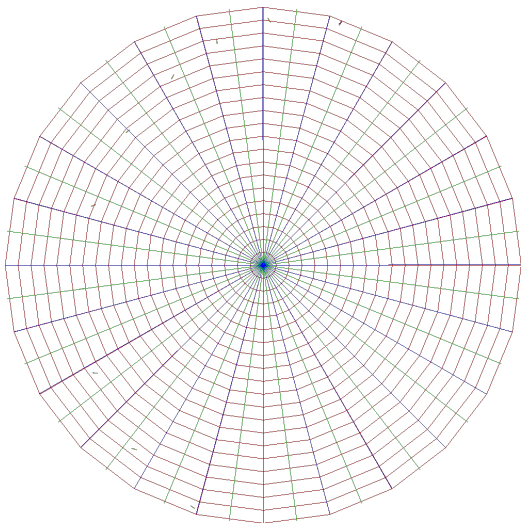


Figura 5.12: División en radios y celdas de la revolución radar

Para facilitar la mantenibilidad, permitiendo además la selección de tipo rellenador de celda, podemos ver en la figura 5.13 como hemos generalizado esta actividad. En dicha figura se observa

la herencia que se aplica para implementar el rellenador más común. Este rellenador, calcula el valor máximo entre las intensidades que en el escenario que representa el mundo real, están dentro de una celda. Este valor máximo sería el usado el mensaje de vídeo radar a difundir por la red.

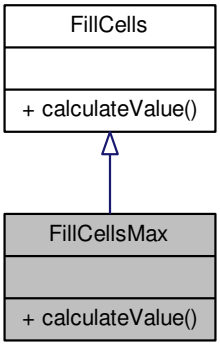


Figura 5.13: Class FillCell

Para facilitar la reutilización de componentes, la ampliación y la depuración del sistema se generalizan las clases que permiten la producción y generación de información. En la figura 5.14 podemos ver el diagrama de herencia de la clase abstracta DataWriter. La clase base establece los métodos fundamentales de escritura que son implementados por las clases hijas [Erich Gamma, 1995]. Como proceso adicional la clase base dispone de la capacidad opcional de ofrecer la cantidad de información procesada. Este contador es usado por las clases de control del flujo de datos, especialmente en la implementación DataUDPWriter. Uno de los objetivos fundamentales de este proyecto es conseguir no saturar la infraestructura de red. La implementación de este contador forma parte de esta implementación.

La clase DBConnect facilita el desacople de funcionalidades del sistema. Como se puede observar en la figura 5.15 el método OnPDUReceived del receptor de tráfico DIS almacena en la base de datos la información referente a las plataformas recibidas. Sin necesidad de conocer quién es el productor, el generador de vídeo radar ReturnsGenerator, lee la información de la base de datos para realizar su trabajo.

Para poder general la señal de vídeo necesitamos procesar el escenario sintético recibido por red. Una de las partes esenciales del procesamiento de dicho escenario es situar y orientar los modelos de plataforma convenientemente como puede verse en la figura 5.4. Sólo así podremos producir un vídeo radar de calidad, fidedigno al escenario recibido. La clase ModelStamper, reflejada en la figura 5.16, es un singleton encargado de realizar este proceso. Realiza la estampación de las plataformas, haciendo uso de un rellenador de celdas y un generador de ruido.

La clase ModelStamper necesita conocer la forma de los objetos detectados. Modelamos esta forma dentro del concepto firma radar. En la figura 5.17 vemos la definición de la clase Signature. ModelStamper, sabiendo la escala a la que está representada la plataforma y su centro de giro

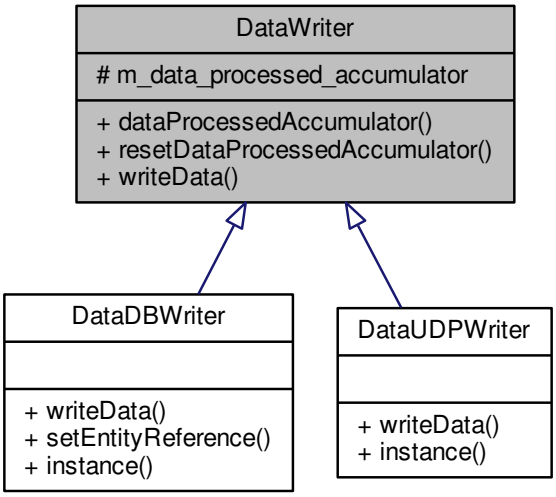


Figura 5.14: Diagrama de herencia clase DataWriter

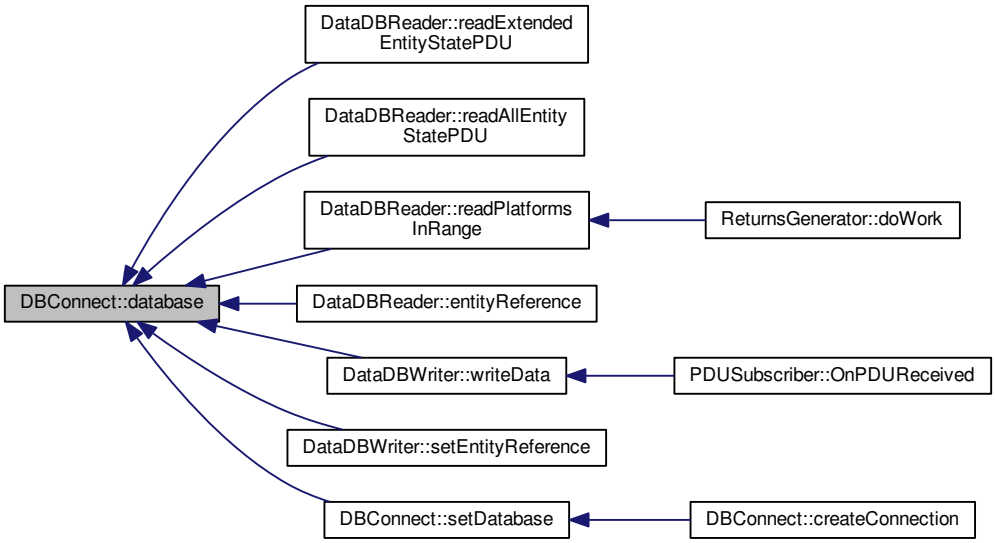


Figura 5.15: Clase DBConnect

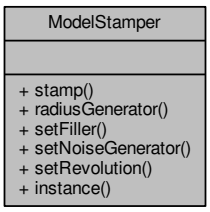


Figura 5.16: Clase ModelStamer

puede situar y orientarla de manera adecuada.

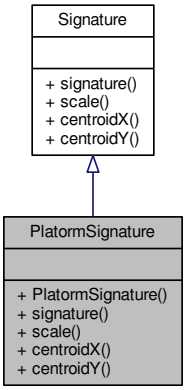


Figura 5.17: Clase Signature

El generador de ruidos ayuda a cumplir con uno de los objetivos del proyecto. La simulación de vídeo radar tiene que acercarse a la realidad. Con este fin generalizamos la generación de ruido, como podemos ver en la clase abstracta de la figura 5.18 con el fin de poder crear e implementar varios generadores de ruido para poder seleccionar el que mejor se adapte a las condiciones del escenario simulado. En la figura 5.18 podemos ver la implementación de un generador de ruido, RandomNoiseGenerator, que se aproxima a la simulación del clutter o ruido de mar.

Una de las necesidades a cubrir por este proyecto es la no saturación de la red. Para ello se establecen los mecanismos necesarios para limitar el consumo de este recurso. En la figura 5.19 se presenta la clase que modela este comportamiento. Consulta la cantidad de información procesada por los DataUPDWriters por segundo y, en el caso de que se exceda un número de violaciones en un tiempo determinado, se reduce el envío de paquetes por segundo en un porcentaje definido mediante fichero de configuración.

La base del funcionamiento de RaViGEn core son los hilos de procesado. En la figuras 5.20 y

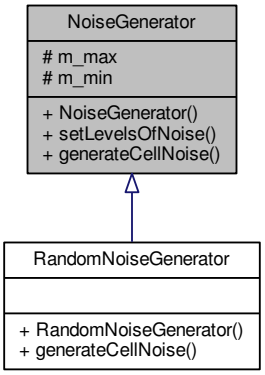


Figura 5.18: Clase Noise Generator

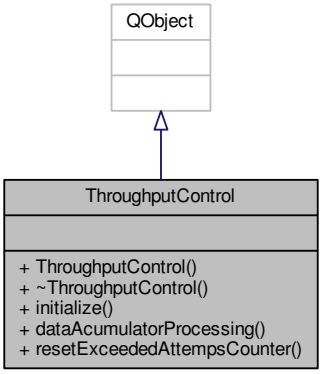


Figura 5.19: Clase Throughput Control

5.21 podemos ver como, haciendo uso de las propiedades que nos ofrece Qt, implementamos hilos de proceso que tendrán su propio bucle principal dentro del método *doWork()* que será llamado en la inicialización de la aplicación siendo ejecutado en un hilo secundario.

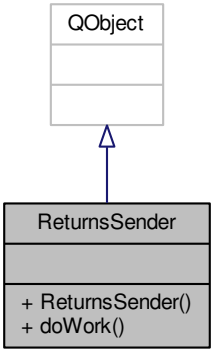


Figura 5.20: Clase Returns Sender

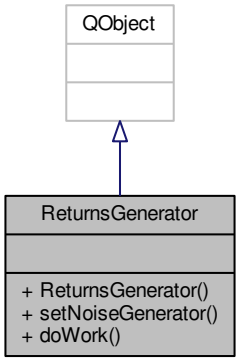


Figura 5.21: Clase Returns Generator

5.2.2. RaViGEn Test Console

El diseño del componente RaViGEn Test Console sigue los mismos principios que RaViGEn Core. La capacidad de configuración, ampliación y mantenimiento son fundamentales en este componente. En la figura 5.22 podemos ver un diagrama de herencia que resume el diseño de este componente. Está compuesto principalmente por tres hilos de trabajo. UDPReceiver recibe el tráfico multicast, principalmente señal de vídeo radar simulada. La clase RadarVideoIma-

geGenerator compone la información recibida y genera la imagen radar a presentar. La clase PlatformsDBProcessing se encarga de presentar las plataformas recibidas haciendo uso de la especificación DIS con el objetivo de poder compararla con el vídeo radar generado certificando o no el buen funcionamiento del producto.

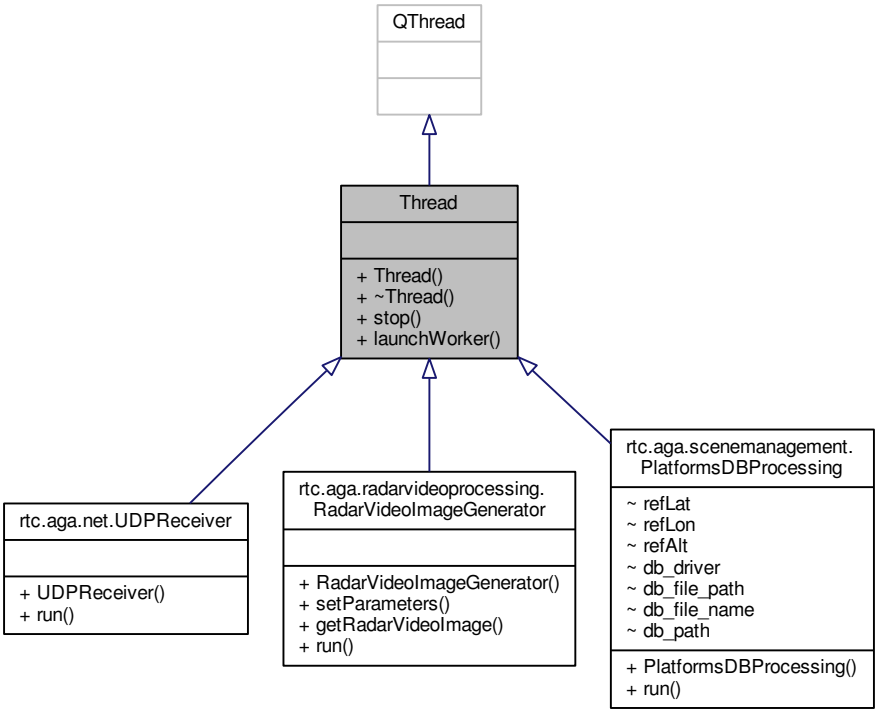


Figura 5.22: Diagrama de herencia RaViGEn Test Console

Como ocurría con RaViGEn Core, es fundamental poder procesar varios tipos de formato de mensaje de vídeo radar. Con este fin, y complementado con el uso de un patrón Factory, se generaliza la implementación de los mensajes radar en la clase RadarVideoFormat de la figura 5.23.

Para completar este objetivo necesitamos también generalizar el procesamiento de cada radar. Utilizamos la clase RadarRadiusFormat de la figura 5.24 para conseguirlo.

Es posible representar el vídeo radar de múltiples maneras. Una son más eficientes que otras. Para poder seleccionar el tipo de renderizado, tal como se muestra en la figura 5.25, se generalizan las clases que tienen la responsabilidad de generar la imagen de radar. Además RaViGEn Test Console ofrece la capacidad de presentar el vídeo radar en tres dimensiones como se muestra en la figura 5.26

RaViGEn Test Console, además de presentar el vídeo radar, es capaz de procesar el tráfico DIS de manera independiente para, una vez establecida una simbología concreta, mostrar las plata-

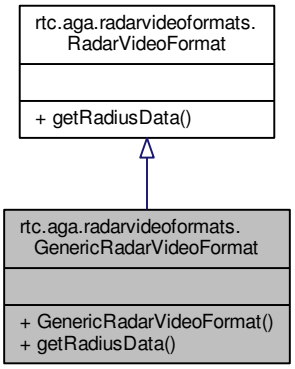


Figura 5.23: Clase RadarVideoFormat

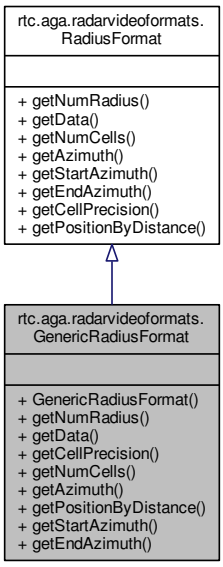


Figura 5.24: Clase RadarRadiusFormat

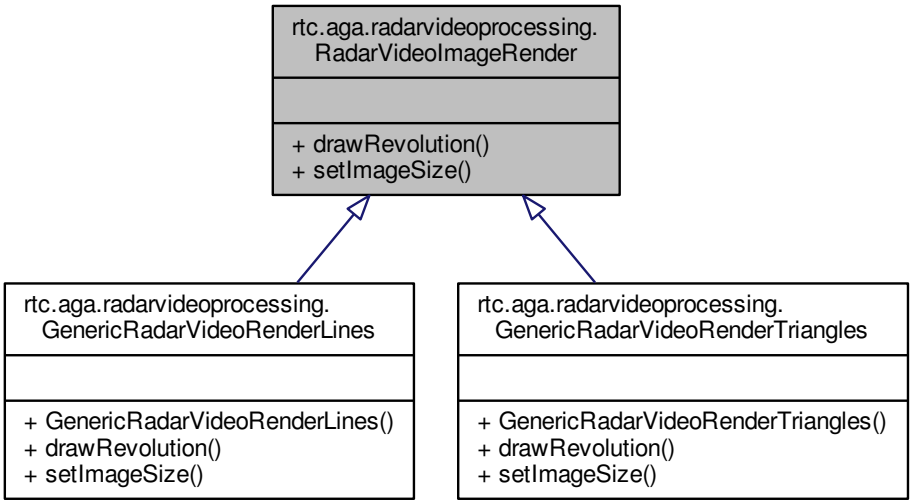


Figura 5.25: Clase RadarVideoImageRender

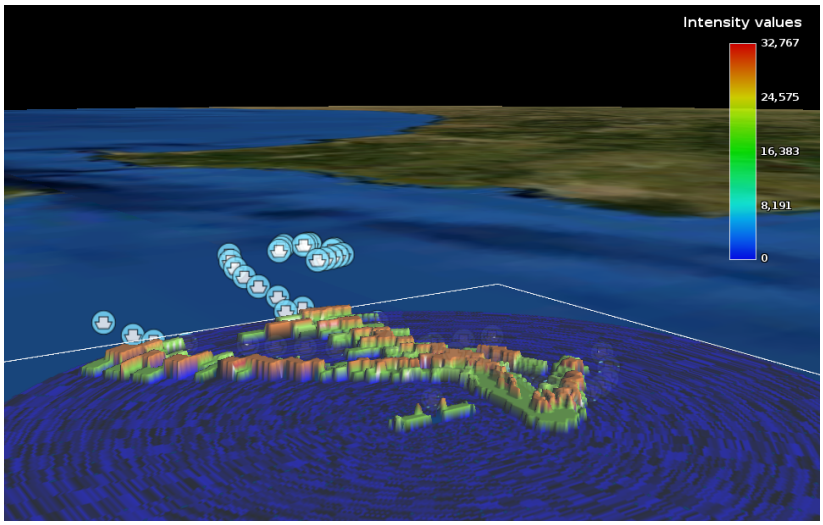


Figura 5.26: Presentación en 3D

formas recibidas. Esta capacidad es fundamental para poder comprobar que el vídeo generado está situado donde se le espera. En la figura 5.27 vemos el diagrama de llamadas que nos permite ver como funciona el gestor de plataformas.

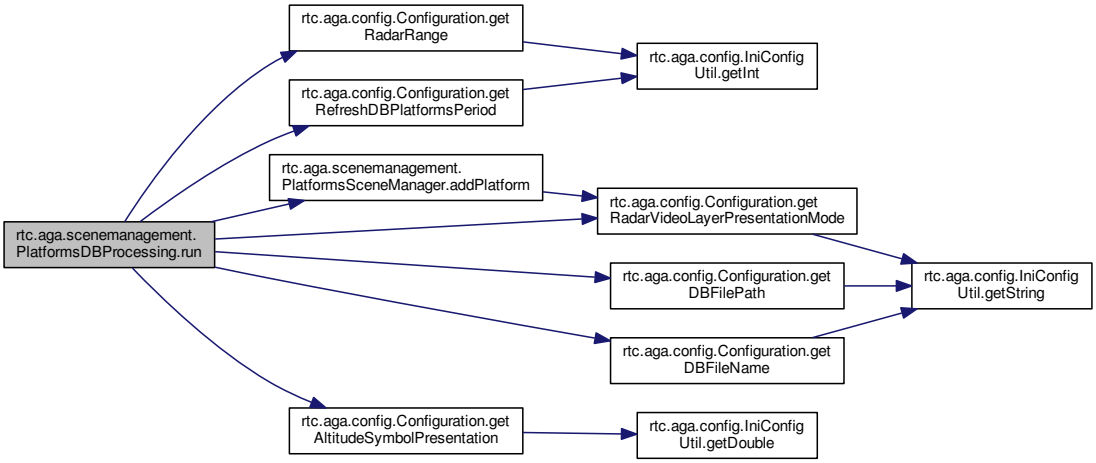


Figura 5.27: Diagrama de llamadas Scene Manager

5.2.3. Platforms Simulator

Platform Simulator es una herramientas de pruebas que nos permite inyectar tráfico DIS como fuente de información para las pruebas de RaViGEn. Es diseñado con el fin de que sea fácilmente extensible para facilitar la inclusión de nuevos escenarios. En la figura 5.28 se presenta los diferentes escenarios creados para los distintos casos de prueba de RaViGEn.

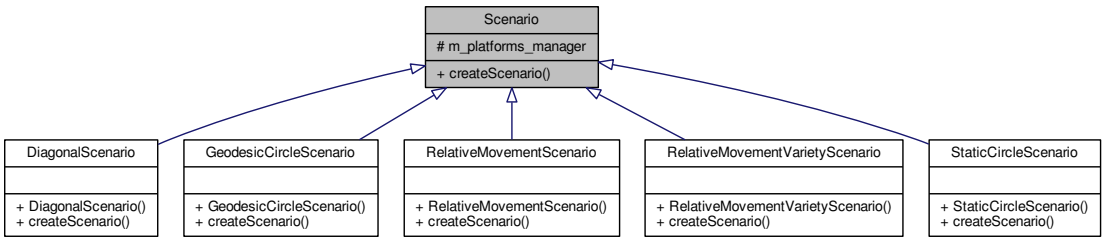


Figura 5.28: Diagrama de herencia de la clase Scenario

5.3. Diseño detallado de la Interfaz de Usuario

RaViGEn Test Console presenta dos zonas claramente diferenciadas mostradas en la figura 5.29. La zona lateral izquierda, donde se sitúan las herramientas de ayuda, y la zona de la derecha, donde podemos ver el mapa que sirve de referencia para el pintado del vídeo radar.

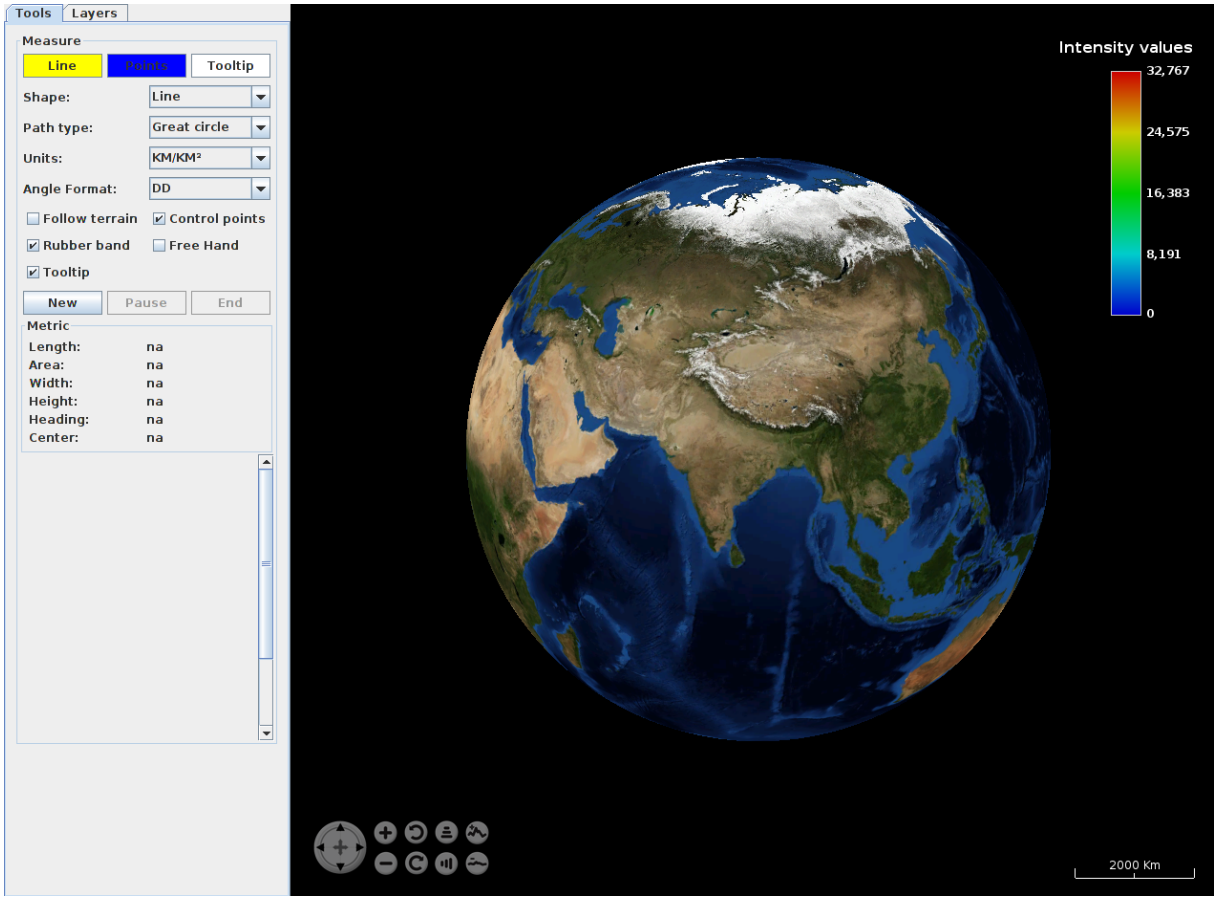


Figura 5.29: Diseño detallado Interfaz de Usuario

5.3.1. Herramientas

La herramienta Measure sirve para generar figuras que permitan medir distancias y superficies sobre el globo terráqueo. Los controles responden de la siguiente manera:

- Shape. Permite elegir la figura a representar.
- Path Type. Define qué tipo de adaptación a la superficie terrestre queremos usar.
- Units. Permite seleccionar las unidades de trabajo.
- Angle Format. En este control seleccionamos el formato del ángulo a presentar.

Al pulsar el botón New el cursor se modifica y se habilita el modo edición sobre el globo terráqueo. Una vez completada la figura se debe pulsar el botón End quedando la figura estática

sobre el globo.

El control Pause sirve para dejar la edición congelada. Es útil para poder navegar por el globo sin tener que terminarla edición de una figura.

En la pestaña Layers, mostrada en la figura 5.30 podemos seleccionar y deseleccionar las capas mostradas sobre el globo terráqueo. Principalmente se agrupan en tres categorías: Vídeo Radar, Plataformas DIS y Cartografía.

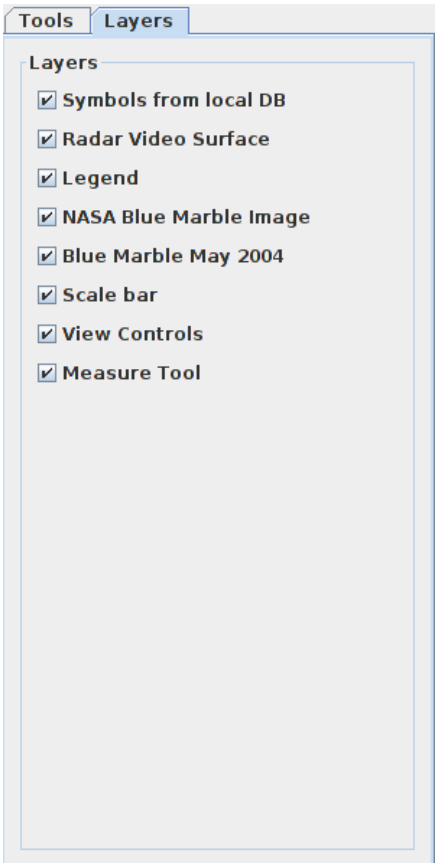


Figura 5.30: Interfaz de Usuario. Selección de capas

5.3.2. Globo Terrestre

Los controles para el manejo y la navegación sobre el globo terrestre del componente WorldWind se especifica en el anexo E.

Capítulo 6

Construcción del Sistema

6.1. Entorno de Construcción

RaViGEn Core usa como lenguaje C++ y como framework de desarrollo Qt. Se ha utilizado el entorno de desarrollo integrado (IDE) QtCreator que integra las herramientas de construcción del proyecto.

RaViGEn Test Console ha sido desarrollado en Java, basado en el framework WorldWind, usando como IDE Eclipse. Ambos componentes se entregan con los proyectos creados y configurados para cada IDE facilitando la construcción del sistema.

Es necesario tener instalada la versión de Qt 4.8 y del SDK de Java 7 tal como se adelantaba en el apartado 2.3.2.

6.2. Código Fuente

El proyecto se distribuye en las siguientes carpetas:

- *bin* En esta carpeta se generan los ejecutables del proyecto
- *cfg* Contiene los ficheros de configuración del sistema.
- *db* Almacena los scripts de base de datos.
- *dep* Incluye las dependencias externas del sistema.
- *doc* Almacena la documentación del proyecto.
- *img* Contiene las imágenes usadas en el proyecto.
- *lib* Almacena las bibliotecas del proyecto.
- *log* En esta carpeta se genera el log del proyecto.
- *src* Contiene el código fuente del proyecto.
- *tests* En esta carpeta están disponibles los programas de prueba.
- *tools* En esta carpeta se almancenan las herramientas de otros proveedores.

6.2.1. RaViGEn Core

La funcionalidad del componente RaViGEn Core está distribuido en bibliotecas. Cada una de estas bibliotecas está contenida en un directorio propio que facilita su mantenimiento y su reutilización.

- *libAsterix* Implementación del formato Asterix 240
- *libConfig* Biblioteca de acceso a los ficheros de configuración.
- *libDebugTools* Biblioteca de funciones de ayuda a la depuración.
- *libDISReceiver* Implementación de la recepción de tráfico DIS.
- *libIOProcessing* Implementación de los Data Writers y Data Readers.
- *libModelStamper* Biblioteca que genera la escena de simulación a partir de la firma radar de las plataformas.
- *libNoiseGenerator* Implementación de los generadores de ruido radar.
- *libRadarModel* Biblioteca encarga de la configuración de modelo radar.
- *libRadarSignature* Implementación de la carga de firmas radar de los elementos de simulación.
- *libRVDB* Base de datos en memoria para RaViGEn. Creación y conexiones.
- *libThroughputControl* Implementación del control de flujo de datos.
- *PlatformsSimulator* Simulador DIS de plataformas.
- *RaViGEn* Programa principal.

En el listado 6.1 podemos ver un ejemplo de la definición de los hilos de trabajo de RaViGEn. La parte fundamental de este diseño es tener implementado el bucle de proceso dentro del método doWork. Hay que tener en cuenta que este método se ejecutará en un hilo secundario y hay que tener especial cuidado con la afinidad de objetos.

```
1  #ifndef RETURNSSENDER_H
   #define RETURNSSENDER_H
3
   // Local
5
   #include "doublebuffer.h"
7  #include "revolution.h"
   #include "radarconfig.h"
9
   // Qt
11
   #include <QObject>
13
   /**
15  * @brief The ReturnsSender class is the worker in charge of send by multicast
```

```
 * the radius builts by ReturnsGeneratos in previus steps.
 */
class ReturnsSender : public QObject
{
    Q_OBJECT
public:

    /**
     * @brief ReturnsSender Constructor
     * Use the configuration \a passed in rct and the \a double_buffer to read
     * the data that will be sent by work method
     */
    ReturnsSender(RadarConfigType *rct,  DoubleBuffer *double_buffer, QObject *
    parent = 0);

public slots:

    /**
     * @brief doWork contains de main loop of the \a Returns Sender worker
     */
    void doWork();

private:

    DoubleBuffer *m_double_buffer; /// Reference to double buffer
    RadarConfigType *m_rct;        /// Reference to radar configuration

    void sendFullTurn(Revolution *rev);
};

#endif // RETURNSSENDER_H
```

Listing 6.1: returnsender.h

En el listado 6.2 se muestra el proceso de generación de la escena que en pasos posteriores, será enviado en el formato de vídeo radar especificado. Esencialmente realiza cambios de escala y posición para generar un buffer usado como representación de la escena. La función radiusGenerator se encarga de calcular la información que debe ir contenida en los distintos radios radar y aplica el ruido. También se observa en este listado la protección de ciertas secciones críticas usando la tecnología OpenMP.

```
// Local
#include "modelstamper.h"
#include "radius.h"
#include "revolution.h"
#include "extended_entity_state_pdu.h"
#include "radarsignature.h"
#include "fillcellsfactory.h"
#include "debugtools.h"
#include "noisegenerator.h"
#include "qimageutils.h"
#include "config.h"

// Qt
#include <QPainter>
#include <QPainterPath>
#include <QCoreApplication>
```

```
17
   // Common
19 #include <Logger.h>
   #include <cmath>
21

23 // KDIS
   #include <KDIS/Extras/KConversions.h>
25
   const double kRotateToNorth = 270;
27
   ModelStamper* ModelStamper::m_instance = NULL;
29
   ModelStamper::ModelStamper()
31 {
       m_radar_signature = RadarSignature::instance();
33
       // By default
35     // Processing cell stamp with max value of the intersection
       m_filler = FillCellsFactory::cellFiller(rvc::Config::cellFiller());
37
       // By default
39     // Empty noise generator
       m_noiser = new NoiseGenerator();
41     m_rev = NULL;
   }
43
   void ModelStamper::setRevolution(Revolution *rev)
45 {
       m_rev = rev;
47 }

49 void ModelStamper::setNoiseGenerator(NoiseGenerator *noiser)
   {
51     m_noiser = noiser;
   }
53
   ModelStamper *ModelStamper::instance()
55 {
       if(m_instance == NULL)
57     {
           LOG_DEBUG("New instance");
59         m_instance = new ModelStamper();
       }
61     return m_instance;
   }
63

65 QPoint ModelStamper::platformPosition(double precision, QPoint origin, double
       distance, double angle)
   {
67     QPoint reference(origin);

69     int distance_pix = (int) distance * precision ;

71     int posy = distance_pix * sin(KDIS::UTILS::DegToRad(angle + kRotateToNorth));
       int posx = distance_pix * cos(KDIS::UTILS::DegToRad(angle + kRotateToNorth));
73
```

```
    reference.rx() += posx;
    reference.ry() += posy;

    return reference;
}

void ModelStamper::stamp(Extended_Entity_State_PDU *ees_pdu)
{

    QImage *buffer = m_rev->revolutionCellsBuffer();

    KDIS::DATA_TYPE::EntityType entity_type = ees_pdu->espdu()->GetEntityType();

    Signature *platform_signature = m_radar_signature->radarSignature(entity_type);
    Q_ASSERT(platform_signature != NULL);

    double heading, pitch, roll;

    KDIS::DATA_TYPE::EulerAngles platform_orientation = ees_pdu->espdu()->
    GetEntityOrientation();
    KDIS::UTILS::EulerToHeadingPitchRoll(KDIS::UTILS::DegToRad(ees_pdu->lat()),
                                         KDIS::UTILS::DegToRad(ees_pdu->lon()),
                                         (double) platform_orientation.
    GetPsiInRadians(),
                                         (double) platform_orientation.
    GetThetaInRadians(),
                                         (double) platform_orientation.
    GetPhiInRadians(),
                                         heading, pitch, roll);


    LOG_DEBUG("Platform %s psi %f, theta %f, phi %f, HEADING %f, PITCH %f, ROLL %f",
              ees_pdu->espdu()->GetEntityMarking().GetEntityMarkingString().c_str(),
              (double) platform_orientation.GetPsiInRadians(),
              (double) platform_orientation.GetThetaInRadians(),
              (double) platform_orientation.GetPhiInRadians(),
              heading, pitch, roll);

    // Rotate and scale the platform signature to take position
    double scale_factor =  platform_signature->scale() / m_rev->cellPrecision();
    QTransform rotation_n_scale;
    rotation_n_scale.rotateRadians(heading);
    rotation_n_scale.scale(scale_factor, scale_factor);

    QImage *buffer_signature = platform_signature->signature();
    QImage signature_rotated_scaled = buffer_signature->transformed(rotation_n_scale
    );

    // Calculate the platform position
    QPoint place = platformPosition( (double) (buffer->width() / 2.0) / m_rev->range
    () ,
                                     QPoint(buffer->width() / 2.0, buffer->height() /
     2.0),
                                     ees_pdu->distance(),
                                     ees_pdu->from_origin_angle());

```

```
125     // Rectify place with bounding box of buffer of signature
        place.rx() -= signature_rotated_scaled.width() / 2.0;
127     place.ry() -= signature_rotated_scaled.height() / 2.0;

129     // Calculate de QRect that cover the signature rotated to save the path
        // The signature path will be used to calculate the intersection with de radius
131     // of the radar
        QPainterPath *path = new QPainterPath();
133     QRect rect_signature = QImageUtils::getBoundsWithoutColor(
        signature_rotated_scaled);
        path->addRect(QImageUtils::addQPointQRect(place, rect_signature));
135     m_path_stamps.append(path);

137     QPainter composer(buffer);
        composer.setCompositionMode(QPainter::CompositionMode_SourceOver);
139     composer.drawImage(place, signature_rotated_scaled);
        composer.end();
141
#ifdef QT_DEBUG
143     DebugTools::saveQImageCapture(*buffer, "rev_buffer.png");
#endif
145 }


147
void ModelStamper::radiusGenerator()
149 {

151 #ifdef QT_DEBUG
        QImage *buffer = m_rev->revolutionCellsBuffer();
153     QImage image = buffer->copy();
        DebugTools::saveQImagePathsRevolution(image, m_rev);
155 #endif

157     // We are going to calculate the insertection shapes
        // to load the fille value in each cell of revolution
159     std::vector<QPainterPath *> *rev_paths = m_rev->pathsRevolution();

161     int size_stamp_paths = m_path_stamps.size();
        int size_rev_paths = rev_paths->size();
163
        LOG_DEBUG("Number of stamp paths to process %d", size_stamp_paths);
165
        // Path intersections finding
167     #pragma omp parallel for
        for(int i = 0; i < size_stamp_paths; i++)
169     {
            unsigned absolute_cell_counter = 0;
171         QPainterPath* path_stamp = m_path_stamps.at(i);

173 #ifdef QT_DEBUG
            #pragma omp critical
175         {
                QImage *buffer = m_rev->revolutionCellsBuffer();
177             QImage paths_debug_image(buffer->size(), QImage::
    Format_ARGB32_Premultiplied);
                paths_debug_image.fill(Qt::transparent);
179
                // For deep debug only
```

```
181             // DebugTools::saveQImagePainterPath(paths_debug_image, *path_stamp, "
    paths_stamp_" + QString::number(i) + ".png", Qt::red);
182         }
183 #endif
184
185         for(int j = 0; j < size_rev_paths; j++)
186         {
187             // Calculating the intersection between radar path and platform
188             // in a QRect F structure
189             QPainterPath *rev_path = rev_paths->at(j);
190             QPainterPath path_intersect = path_stamp->intersected(*rev_path);
191             QRectF rect_intersect = path_intersect.boundingRect();
192
193             // Less than a pixel
194             if(rect_intersect != QRectF())
195             {
196
197                 // Calculating the measurement of value that it has to fill
198                 // the cell to be transmited by RaViGEn
199                 QRect copy_rect = rect_intersect.toRect();
200
201                 // We lost precision using a QRect instead of QRectF but pixel
202                 // operations needs an int
203                 if(copy_rect.height() != 0 && copy_rect.width() != 0)
204                 {
205                     QImage intersected_image = m_rev->revolutionCellsBuffer()->copy(
    copy_rect);
206                     unsigned value = m_filler->calculateValue(intersected_image);
207
208                     m_rev->fillCell(absolute_cell_counter, value);
209
210 #ifdef QT_DEBUG
211                     // For deep debug only
212                     // DebugTools::saveQImageCapture(intersected_image, "copy" +
    QString::number(absolute_cell_counter) + ".png");
213 #endif
214                 }
215             }
216
217             // Making and applying noise
218             unsigned noise_value = m_noiser->generateCellNoise(absolute_cell_counter
    , m_rev->getCellValueByAbs(absolute_cell_counter));
219             m_rev->fillCell(absolute_cell_counter, noise_value);
220
221             absolute_cell_counter++;
222
223         }
224     }
225
226 #ifdef QT_DEBUG
227     DebugTools::saveQImageLinearRadiusRevolution(m_rev);
228     DebugTools::saveQImageCenterRadiusRevolution(m_rev);
229 #endif
230
231     qDeleteAll(m_path_stamps);
232     m_path_stamps.clear();
233 }
```

```
235 void ModelStamper::setFiller(FillCells *filler)
{
237     m_filler = filler;
}
```

Listing 6.2: modelstamper.cpp

El la imagen 6.1 se presenta la primera parte de este proceso. Se ha creado la imagen que representa la escena y se ha seccionado en radios. Seguidamente Se rellena cada sección, bien con detecciones, bien con ruido, como podemos ver en la figura 6.2, y se transfiere al proceso que va a transformar esta información al formato de vídeo radar seleccionado.

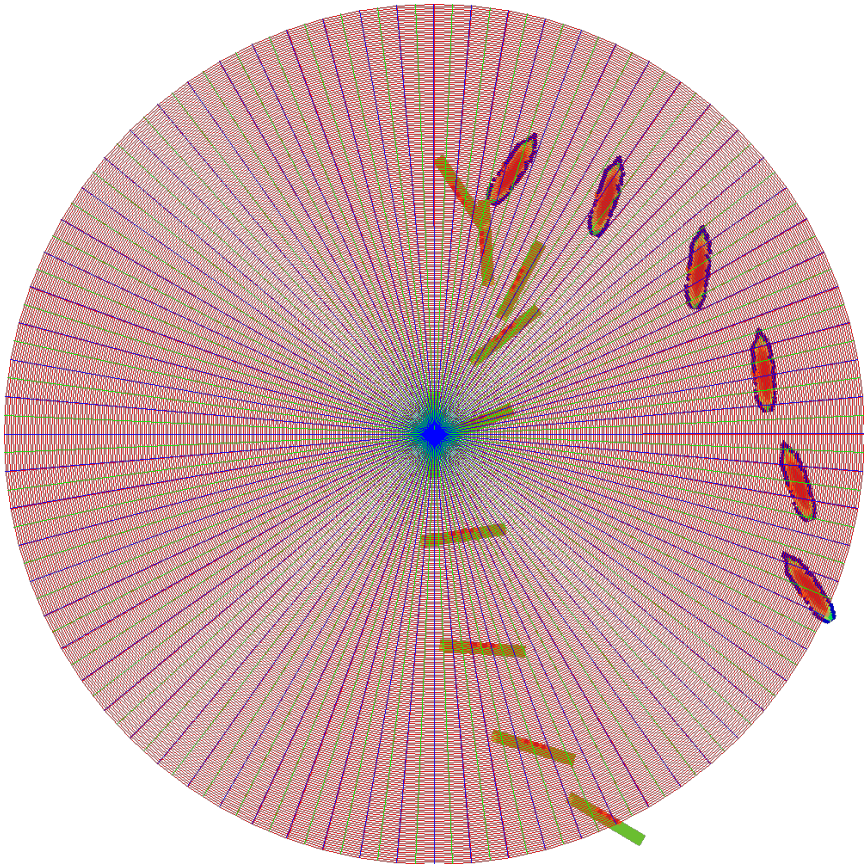


Figura 6.1: Escena seccionada

En el listado 6.3 se muestra un ejemplo de implementación de generador de ruido. Cumple el interfaz de los generadores de ruido, ya que su construcción se hace en un factory siendo seleccionado de entre disponibles. En este caso se pretende generar un ruido aleatorio dentro de unos márgenes establecidos que responda a ruido de mar.

```
// Locals
2 #include "randomnoisegenerator.h"
#include "revolution.h"
4
// Common
```

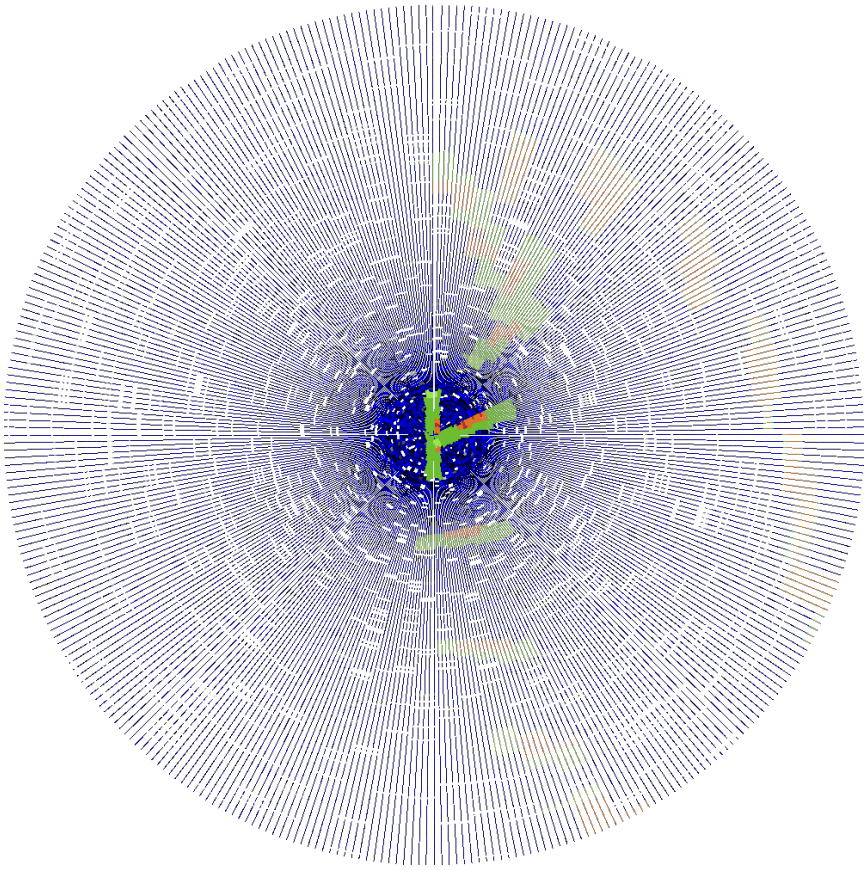


Figura 6.2: Escena divida en radios

```
6  #include <Logger.h>
   #include <climits>
8  #include <cmath>
   #include <ctime>
10
   RandomNoiseGenerator::RandomNoiseGenerator()
12 {
       srand (time(NULL));
14 }

16
   unsigned RandomNoiseGenerator::generateCellNoise(unsigned pos, unsigned
       original_value)
18 {

20     if(0x0 ==  original_value)
       {
22         unsigned distance = (m_max - m_min);

24         // Reference value: median
           double median = (distance / 2.0) + m_min;
26         double angle = ((median + pos) * 360) / m_max;

28         double signal = sin(angle * M_PI / 180);
           double sigma = 0.07; //  % Noise standard desviation
30
           double noisy = (signal*distance + sigma*rand()) / (pos + 1);
32
           unsigned in_range = ((int) noisy)  % distance + m_min;
34         return (0xFF000000 | in_range);

36     }
       else
38     {
           return original_value;
40     }
   }
```

Listing 6.3: randomnoisegenerator.cpp

6.2.2. RaViGEn Test Console

Con los mismos objetivos que RaViGEn Core, la funcionalidad del componente se divide los siguientes paquetes:

- *rtc.aga.config* Biblioteca de acceso a los ficheros de configuración.
- *rtc.aga.net* Implementación de la recepción de vídeo radar.
- *rtc.aga.scenemanagement* Gestiona la presentación de las plataformas.
- *rtc.aga.radarmodel* Paquete encargado de la configuración de modelo radar.
- *rtc.aga.radarvideoformats* Implementación de los formatos radar.
- *rtc.aga.radarvideoprocessing* Procesa la señal radar y renderiza la imagen resultante.

- *rtc.aga.tools* Implementación de utilidades auxiliares.
- *rtc.aga.main* Programa principal.

Capítulo 7

Pruebas del Sistema

7.1. Estrategia

Para garantizar la estabilidad y rendimiento se realizan, principalmente, cuatro niveles de pruebas: pruebas unitarias, pruebas de integración, pruebas de aceptación y pruebas de regresión.

7.2. Entorno de Pruebas

El equipo donde se realizan las pruebas es el mismo utilizado para el desarrollo del sistema.

- Intel Quad Core q6600
- 4 GiB RAM
- Tarjeta gráfica Nvidia 9500 GT
- Ubuntu 16.04 LTS 64 bits

7.3. Análisis estático

Como paso previo a las pruebas de software se realiza un análisis estático del código generado haciendo uso de la herramienta *cpp_ check*. La política del proyecto frente a los errores y avisos de las herramientas de análisis estático es realizar entregas del producto con cero advertencias o errores.

Se realiza la entrega del proyecto con cero positivos en la prueba de análisis estático. La herramienta ofrece un falso positivo en la gestión de un puntero, perteneciente a la clase *Radius* que no debe ser liberado durante la ejecución del sistema.

```
Checking libRadarModel/radius.cpp...
[libRadarModel/radius.cpp:22]: (error) Mismatching allocation and
deallocation: Radius::prf_buffer
```

7.4. Niveles de Pruebas

7.4.1. Pruebas Unitarias y Pruebas de Integración

Para la creación de los test unitarios utilizamos el framework Qt Testing. Todas las clases han sido depuradas haciendo uso de los casos de prueba definidos utilizando las capacidades ofrecidas por el depurador GNU/GDB.

Se realizan pruebas unitarias de todos aquellos componentes de nuevo desarrollo. En aquellos casos que la dependencia de las librerías aconseje realizar una prueba de integración que nos facilite la prueba unitaria este será el enfoque escogido.

En el listado 7.1 se ofrece un ejemplo particular de prueba unitaria. La orientación de esta prueba no es probar la clase (*QSettings*) que subyace en la implementación de libConfig sino asegurar que los métodos implementados hacen uso de los campos de configuración tal como los espera la aplicación.

En el listado 7.2 se muestra un extracto de prueba unitaria del uso de un formato radar. Se instancian las clases con unos valores definidos y se comprueba que la inicialización y codificación del mensaje ha sido correcta.

Las pruebas realizadas son de caja blanca, depurando las librerías construidas hasta conseguir su madurez y garantizando un grado de cobertura mínimo para considerar a aplicación estable. Para todo el código generado, se siguen las líneas principales de ejecución comprobando los valores tomados por las variables [Kelly J. Hayhurst, 2001].

Las aplicaciones de pruebas se distinguen en el repositorio por estar alojadas en la carpeta *test* y compartir el sufijo Test.

```
1  #include <QString>
   #include <QtTest>
3
   #include "config.h"
5
   class ConfigTest : public QObject
7  {
       Q_OBJECT
9
   public:
11     ConfigTest();

13 private Q_SLOTS:
       void initTestCase();
15     void cleanupTestCase();
       void IPSetting();
17     void PortSetting();

19 private:
       QString m_ip;
21     unsigned m_port;
   };
23
```

```
ConfigTest::ConfigTest()
{
}

void ConfigTest::initTestCase()
{
    // Backup deployment values

    m_ip = rvc::Config::DISIP();
    m_port = rvc::Config::DISPort();

}

void ConfigTest::cleanupTestCase()
{
    // Restore deployment values
    // and check all is right

    rvc::Config::setDISIP(m_ip);
    rvc::Config::setDISPort(m_port);

    QVERIFY2(m_ip == rvc::Config::DISIP() &&
             m_port == rvc::Config::DISPort(),
             "Cleanup failed");
}

void ConfigTest::IPSetting()
{
    rvc::Config::setDISIP(QString("235.10.20.255"));
    QString ip = rvc::Config::DISIP();

    QVERIFY2(ip == "235.10.20.255", "DIS IP configuration failed");
}

void ConfigTest::PortSetting()
{
    rvc::Config::setDISPort(20255);
    unsigned port = rvc::Config::DISPort();

    QVERIFY2(port == 20255, "DIS Port configuration failed");
}


QTEST_MAIN(ConfigTest)

#include "tst_configtest.moc"
```

Listing 7.1: tst_configtest.cpp

```
// Local

#include "genericradarvideoformat.h"
#include "genericradarvideobuffer.h"
#include "radarvideomarshallfactory.h"


// Qt
#include <QString>
```

```
#include <QtTest>

class RadarVideoFormatTest : public QObject
{
    Q_OBJECT

public:
    RadarVideoFormatTest();

private Q_SLOTS:
    void RadarVideoFormatTestCase();
    void RadarVideoBufferTestCase();
};

RadarVideoFormatTest::RadarVideoFormatTest()
{
}

void RadarVideoFormatTest::RadarVideoFormatTestCase()
{
    RadarVideoFormat *rvf = new GenericRadarVideoFormat();
    GenericRadarVideoFormat *grvf = static_cast<GenericRadarVideoFormat *>(rvf);

    unsigned data[1000];
    memset(data, 250, 100 * sizeof(unsigned));

    grvf->setHeader(0, 30, 20);
    grvf->setData(data, 1000);

    grvf->serialize();
    char *encoded_data = grvf->encodedData();

    Q_UNUSED(encoded_data);

    QVERIFY2(grvf->serializedDataLength() == 16 + 4000 , "Failure");
}

void RadarVideoFormatTest::RadarVideoBufferTestCase()
{
    RadarVideoBuffer* rvb = RadarVideoMarshallFactory::buffer();

    unsigned data[1000];
    memset(data, 250, 100 * sizeof(unsigned));

    GenericRadarVideoBufferSettings grvbs;
    grvbs.start_az = 0;
    grvbs.end_az = 30;
    grvbs.cell_precision = 5;
    grvbs.data = data;
    grvbs.data_size = 1000;

    rvb->initialize((void *) &grvbs);


    RadarVideoFormat *rvf = rvb->next();

    int cont = 0;

```

```
    while(NULL != rvf)
    {
        char *data = rvf->encodedData();
        Q_UNUSED(data);
        rvf = rvb->next();

        cont++;
    }

    QVERIFY2(rvf == NULL && cont == 1 , "Failure");
}

QTEST_APPLESS_MAIN(RadarVideoFormatTest)

#include "tst_radarvideoformattest.moc"
```

Listing 7.2: tst_radarvideoformattest.cpp

7.4.2. Pruebas Funcionales del Sistema

Se implementa una aplicación, llamada PlatfomsSimulator, que implementa escenarios deterministas, de los que se conoce la salida esperada de RaViGEn, y que provocan el uso del sistema completo. Estos escenarios son seleccionables en los ficheros de configuración y son:

- Escenario diagonal. Presenta dos diagonales en cruz entorno al buque de referencia como puede verse en la figura 7.1.
- Escenario circular estático. Presenta un número de plataformas configurable a una distancia equidistante en el plano de la plataforma de referencia como se muestra en la figura 7.2.
- Escenario circular estático geodésico. Presenta un número de plataformas configurable a una distancia, equidistante sobre la superficie de la tierra, establecida mediante fichero de configuración de la plataforma de referencia.
- Escenario circular con movimiento relativo. Presenta un número de plataformas configurable a una distancia equidistante establecida mediante fichero de configuración de la plataforma de referencia. Las plataformas tienen un movimiento hacia la plataforma de referencia mientras que esta plataforma tiene una velocidad y rumbo constante hacia el norte.

Se presenta el Escenario Circular en la figura 7.3. En las pruebas de sistema se comprueban los siguientes aspectos principales:

- El generador de vídeo radar responde correctamente a cambios de velocidad, orientación y dirección.
- El resultado del vídeo radar es el esperado:
  - El vídeo radar se genera en el lugar definido.
  - Tiene los valores de intensidad esperados.
  - El tiempo de ciclo es adecuado.
  - El vídeo radar se auto-ajusta frente a un consumo excesivo de red.

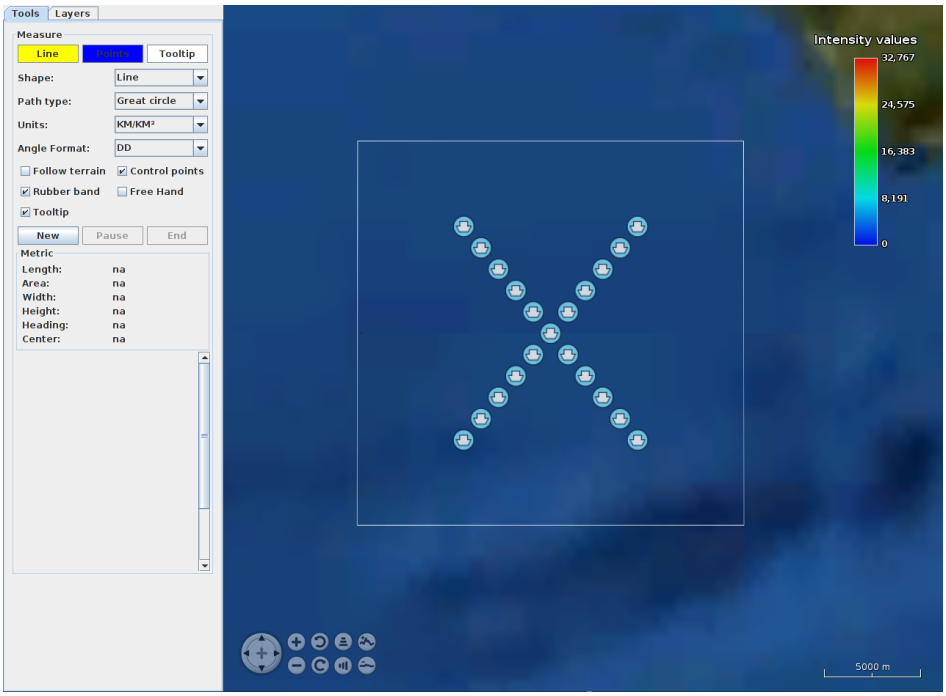


Figura 7.1: Escenario diagonal

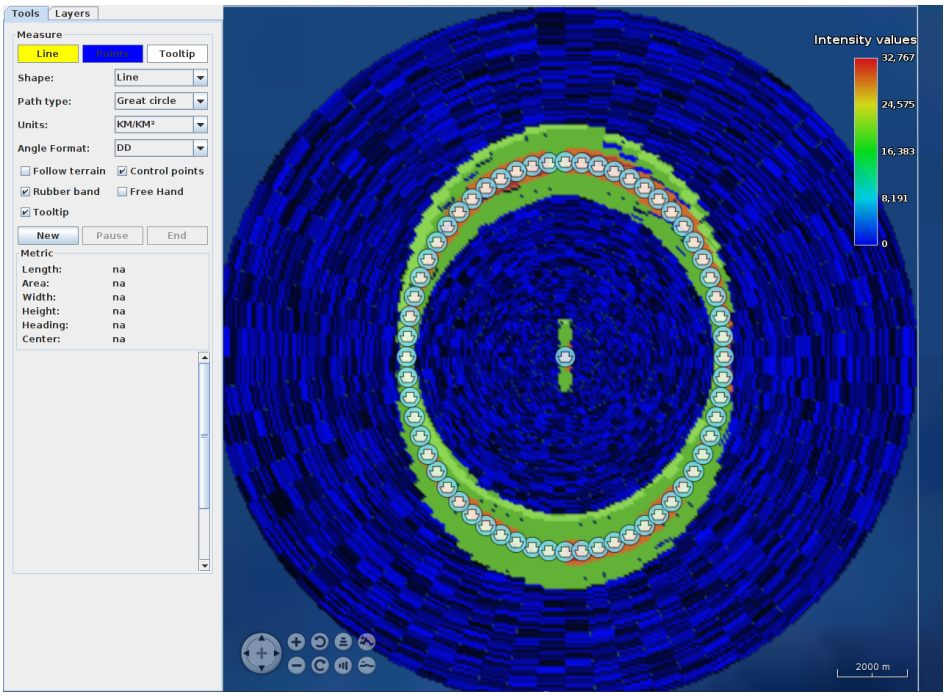


Figura 7.2: Escenario circular estático

El consumo de red es acotado por fichero de configuración.

En RaViGEn distinguimos dos tipos de flujos alternativos:

- Flujo alternativo por elección de ciertos parámetros de configuración.
- Flujo alternativo por tratamiento de errores.

El primer caso de flujo alternativo se ha tratado en cada prueba que realizada. La configuración del sistema para las pruebas hace que que hayan sido usadas las distintas configuraciones. Las excepciones producidas por errores, como la de fallo en escritura de fichero, no han podido ser probadas en tu totalidad debido a la complejidad de producir los errores tratados en el sistema. RaViGEn está desarrollado de modo que, ante un error fatal, se produzca una salida abrupta de la aplicación para evitar dejar al sistema en un estado inconsistente. Al tratarse de un sistema de simulación, la evaluación de riesgos de seguridad de los problemas que pueden darse en RaViGEn por un mal funcionamiento de la aplicación son de baja prioridad [David L. Parnas and Kwan, 1990].

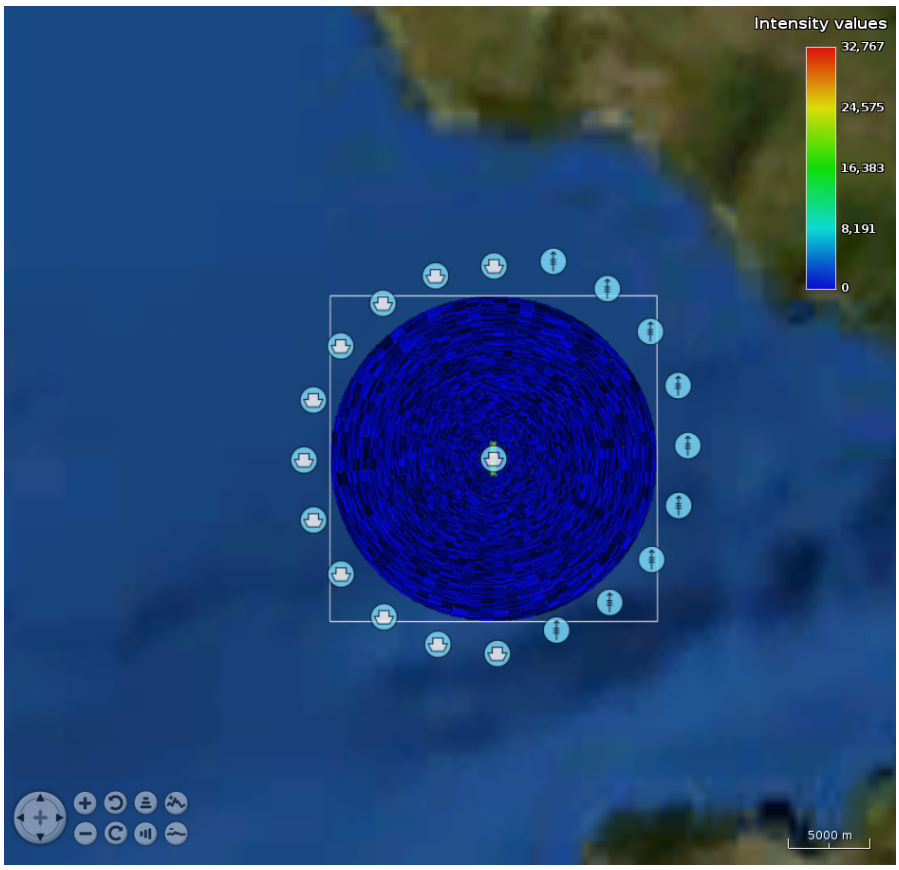


Figura 7.3: Escena seccionada

Pruebas No Funcionales

Se han realizado pruebas para garantizar el cumplimiento de todos los requisitos no funcionales del sistema. Además se han realizado pruebas de rendimiento del sistema para garantizar que sea

capaz de procesar una carga razonable de entidades DIS. Se ha usado la herramienta QElapsedTimer, de Qt, para medir el rendimiento de determinados procesos del sistema obteniendo como resultado el poder procesar cuarenta plataformas DIS en movimiento, de manera simultánea, en el entorno de configuración definido par ala prueba.

Durante el desarrollo se han usado los registros de tiempo para mejorar el desempeño del sistema. Los procesos escriben en el registro del sistema el tiempo qué tarda cada parte del proceso en ejecutarse e intentamos mejorarlos para cumplir con los umbrales de tiempo establecidos en los ficheros de configuración. RaViGEn es un sistema de tiempo real blando que necesita que la ejecución cumpla con dichos tiempos para que la simulación sea realista.

Una vez revisados estos tiempos, y una vez detectadas las secciones críticas, se ha refactorizado el código para aplicar OpenMP es varios de sus procesos. La naturaleza del problema resuelto, junto con la aproximación a la solución elegida que hace uso de imágenes representados en matrices de bits, permite aplicar las técnicas de paralelización que ofrece OpenMP [Tim Mattson, 2009].

Se han realizado pruebas de duración de cinco horas donde se ha anotado el consumo de memoria de la aplicación en varios puntos de la prueba para garantizar la no existencia de fugas de memoria. La aplicación Valgrind ofrece algún positivo considerado falso después de haber ejecutado esta prueba.

Para verificar que el sistema ajusta el consumo de red a los valores definidos en la configuración del sistema hemos usado la herramienta Wireshark y Gnome System Monitor. Gnome System Monitor nos ha permitido tener una referencia visual de los momentos de ajuste del consumo de red. Con Wireshark hemos realizado un análisis pormenorizado del consumo de la aplicación. Los resultados han sido positivos. El sistema es capaz de ajustar el consumo de red de manera satisfactoria.

La extensibilidad y la aceptación de nuevos formatos se ha probado de manera implícita en las pruebas funcionales del sistema. Se han usado varios tipos de plataformas y de formatos de vídeo para generar los diferentes escenarios.

7.4.3. Pruebas de Aceptación

Las pruebas de aceptación consisten en la ejecución del *Escenario Circular con movimiento relativo* durante diez minutos con la versión de producción del sistema comprobando los siguientes aspectos principales:

- El generador de vídeo radar responde correctamente a cambios de velocidad, orientación y dirección.
- El resultado del vídeo radar es el esperado:
  - El vídeo radar se genera en el lugar definido.
  - Tiene los valores de intensidad esperados.
  - El tiempo de ciclo es adecuado.
  - El vídeo radar se auto-ajusta frente a un consumo excesivo de red.

El consumo de red es acotado por fichero de configuración.

Parte III

Epílogo

Capítulo 8

Manual de usuario

8.1. Introducción

RaViGEn es un sistema dedicado a la simulación de vídeo radar en escenarios coordinados haciendo uso de la tecnología DIS. Este producto, principalmente, nos proporciona:

- Uso de diferentes formatos de vídeo radar.
- Autogestión del consumo de red.

Además incorpora una aplicación, RaViGEn Test Console, para comprobar que el envío de datos de video radar es correcto.

8.2. Instalación

Se recomienda la siguiente configuración:

- PC, con procesador Intel Core i5 o equivalente, al menos 4GiB de RAM y tarjeta gráfica Nvidia.
- Sistema operativo Ubuntu 16.04 Desktop 64 bits
- OpenJDK con soporte para Java en su versión 7

La instalación consta de los siguientes pasos:

1. Descomprimir la carpeta *ravigen* en el directorio principal del usuario.
2. Extraer el tarball de instalación los binarios y librerías en la ruta especificada. No son necesarios privilegios especiales.
3. Establecer los parámetros deseados en el fichero de configuración según el apartado 8.2.1.
4. En el caso de modificar los parámetros del fichero de configuración, crear la carpeta *db* en el directorio casa del usuario.

8.2.1. Fichero de configuración

El fichero de configuración del sistema sigue el formato "clave = valor". A continuación se describe, por secciones, cada uno de los parámetros del fichero de configuración. El proyecto es entregado con un fichero de configuración totalmente operativo.

FILE

db_dir Directorio de almacenamiento de la base de datos.

db_filename Nombre del fichero de la base de datos.

log_dir Directorio de almacenamiento del registro del sistema.

THROUGHPUTCONTROL

interval Establece el intervalo de comprobación de uso del flujo de red (ms).

threshold Establece el límite de uso de red deseado (bytes).

attemps Sirve para configurar el número de faltas permitidas. Una falta es un valor por encima del límite o umbral.

attemps_interval Una vez que el algoritmo correctivo del uso de recursos de red es aplicado establece un tiempo de no comprobación para que el sistema pueda adaptarse a las nueva situación. Ese periodo es definido en este parámetro (ms).

slow_down_factor Es el factor de corrección que se aplica sobre el tiempo de revolución de radar para reducir el consumo de red en el caso que sea necesario (porcentaje).

DIS

ip Establece la IP UDP multicast del ejercicio DIS al que suscribirse.

port Establece el puerto del ejercicio DIS al que suscribirse.

exercise Establece el número de ejercicio en el que queremos participar.

RADAR

range Establece el alcance del radar (metros).

azimut_precision Establece la precisión angular del radar en grados.

cell_precision Establece la precisión de cada celda radar en metros.

revolution_time Define el tiempo de revolución o ciclo de radar (ms).

radar_video_format Permite seleccionar el formato de radar.

port Define el puerto por el que la simulación de vídeo radar será enviada.

ip_multicast Define la IP multicast por el que la simulación de vídeo radar será enviada.

CELL_BUFFER

width Define el tamaño del sensor del radar en bits.

height Define el tamaño del sensor del radar en bits.

cells_filler Define el protocolo seguido para rellenar cada celda radar.

ENVIRONMENT

noise_generator Permite seleccionar el generador de ruido.

max_noise_level Establece el valor máximo de intensidad de ruido generado (entero).

min_noise_level Establece el valor mínimo de intensidad de ruido generado (entero).

REFERENCE

entity_id Parte de la identificación DIS del buque de referencia. identificación numérica.

app_id Parte de la identificación DIS del buque de referencia. Aplicación que la publica.

site_id Parte de la identificación DIS del buque de referencia. Lugar desde el que es publicado.

OMP

dynamic Establece la adaptación genérica de los hilos OpenMP.

num_threads Cuando la configuración no es dinámica permite especificar el número de hilos a usar.

DEBUG

log_level Define el nivel de registro del sistema.

SIMULATOR

site_id Parte de la identificación DIS del simulador.

application_id Parte de la identificación DIS del simulador.

period_time Tiempo de actualización de las plataformas simuladas (ms).

SIMULATION

ip Define la IP multicast por el que la simulación de vídeo radar será enviada.

simulation_rate Define la velocidad de la simulación (ms).

dead_reckoning_period Establece el periodo del cálculo de la navegación por estima (ms).

scenario Permite elegir el escenario a ejecutar.

lat Establece el punto central del escenario en latitud (grados).

lon Establece el punto central del escenario en longitud (grados).

platforms_velocity Define la velocidad de las plataformas (m/s).

oriented_velocity Establece si se desea que las plataformas se muevan hacia donde están orientadas.

range Establece la distancia a la que se situarán las plataformas en los escenarios dinámicos (metros).

num_platforms Define el número de plataformas a incluir en el escenario.

reference_velocity Permite establecer la velocidad de la unidad de referencia (m/s).

CONSOLE

cell_buffer_size Establece el tamaño del pixmap usado en el interfaz gráfica para representar el vídeo radar (píxeles).

legend_pos_x Define la posición del control de leyenda en pantalla (píxeles).

legend_pos_y Define la posición del control de leyenda en pantalla (píxeles).

refresh_image_period Define el periodo de actualización de la escena en pantalla (ms).

radar_layer_opacity Establece la opacidad de la capa de vídeo radar (porcentaje).

radar_layer_presentation Nos permite definir el modo de presentación del radar (2D/3D).

altitude_3D_presentation Establece el nivel de altura de la capa de presentación radar en el modo 3D (metros).

max_altitude_3D_presentation Define una cota de presentación 3D del radar (metros).

altitude_symbols_3D En el caso de establecer el modo de presentación 3D para el vídeo radar esta opción permite elevar la presentación de las plataformas para que puedan verse en todo momento (metros).

regenerate_image_period Establece el tiempo de regeneración de la imagen radar. Debe ser superior al tiempo de refresco (ms).

refresh_db_platforms Establece el periodo de actualización de presentación de las platormas (ms).

radar_video_image_render Establece el modo de renderizado de vídeo radar (líneas o triángulos).

symbol_sidc Establece el identificador del tipo de símbolo a mostrar para las plataformas.

symbol_opacity Establece el grado de opacidad de los símbolos que representan a las plataformas (porcentaje).

symbol_scale Establece la escala para los símbolos que representan a las plataformas (porcentaje).

8.3. Uso del sistema

Para el despliegue en un entorno GNU/Linux es suficiente con ejecutar el siguiente comando dentro del directorio de instalación de RaViGEn:

```
$ ./bin/RaViGEn
```

Si se desea ejecutar el visualizador de video radar es suficiente con ejecutar el siguiente comando dentro del directorio de instalación de RaViGEn para lanzar el componente RaViGEn Test Console:

```
$ java -jar ./bin/RaViGEnTestConsole
```

RaViGEn Test Console presenta dos zonas claramente diferenciadas mostradas en la figura 8.1. La zona lateral izquierda, donde se sitúan las herramientas de ayuda, y la zona de la derecha, donde podemos ver el mapa que sirve de referencia para el pintado del vídeo radar.

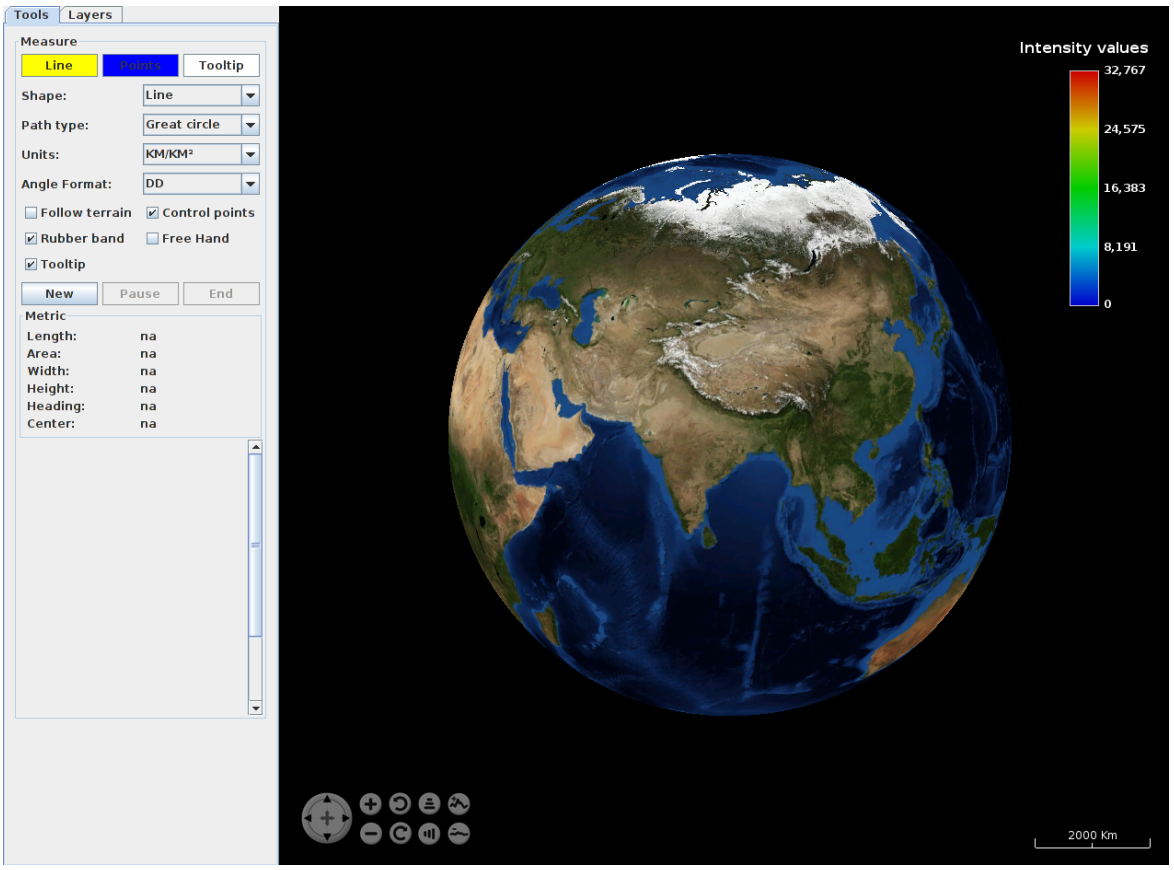


Figura 8.1: Interfaz de Usuario RaViGEn Test Console

8.3.1. Herramientas

La herramienta Measure sirve para generar figuras que permitan medir distancias y superficies sobre el globo terráqueo como se observa en la figura 8.2. Los controles responden de la siguiente manera:

- Shape. Permite elegir la figura a representar.
- Path Type. Define qué tipo de adaptación a la superficie terrestre queremos usar.
- Units. Permite seleccionar las unidades de trabajo.
- Angle Format. En este control seleccionamos el formato del ángulo a presentar.

Al pulsar el botón New el cursor se modifica y se habilita el modo edición sobre el globo terráqueo. Una vez completada la figura se debe pulsar el botón End quedando la figura estática sobre el globo.

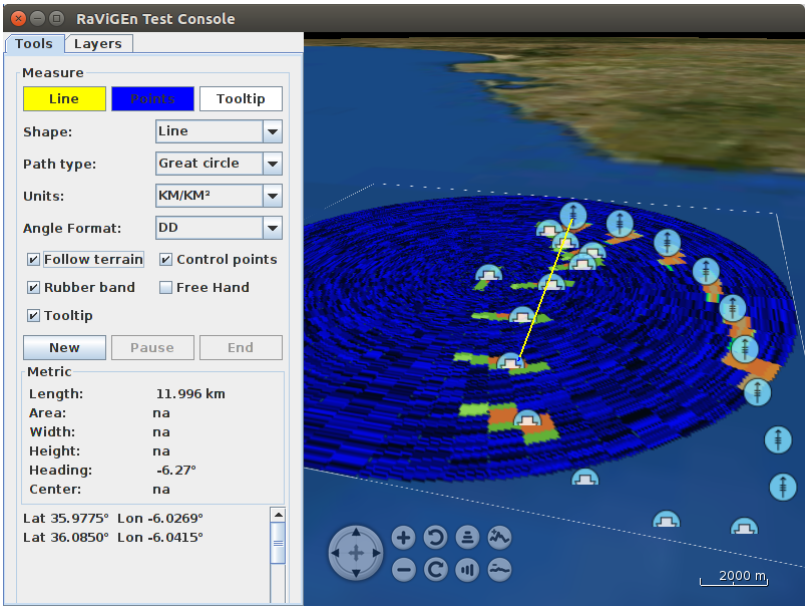


Figura 8.2: Herramienta de toma de medidas en funcionamiento

El control Pause sirve para dejar la edición congelada. Es útil para poder navegar por el globo sin tener que terminarla edición de una figura.

En la pestaña Layers, mostrada en la figura 8.3 podemos seleccionar y deseleccionar las capas mostradas sobre el globo terráqueo. Principalmente se agrupan en tres categorías: Vídeo Radar, Plataformas DIS y Cartografía.

8.3.2. Globo Terrestre

Los controles para el manejo y la navegación sobre el globo terrestre del componente WorldWind se especifica en el anexo E.

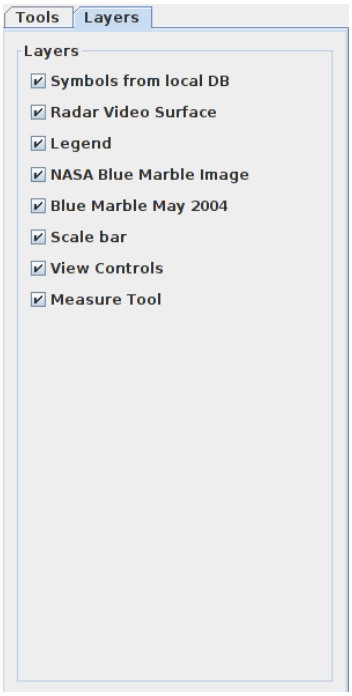


Figura 8.3: Interfaz de Usuario. Selección de capas

Capítulo 9

Conclusiones

9.1. Objetivos alcanzados

Se ha conseguido, haciendo uso de metodologías ágiles y de técnicas aprendidas en el itinerario del máster, desarrollar un producto que permite conseguir los siguientes objetivos:

- Participar en un escenario de simulación coordinado aportando la generación de vídeo radar que colabore con el adiestramiento en sistemas distribuidos.
- Disponer de una solución software abierta a la extensión, capaz de incorporar nuevos modelos y formatos de radar.
- Integrar el producto en un sistema con una red de ancho de banda limitado, acotando el consumo de red del simulador de vídeo radar.
- Utilización e integración de diferentes tecnologías. Se han usado varios lenguajes de programación, se aplican estándares como DIS o Asteris 240, manejo de conexiones de red, OpenMP... para resolver el problema propuesto.
- Aplicar técnicas de verificación, pruebas y depuración.

El acometer un TFM con codificación software implicaba cierto riesgo en cuanto a desviaciones con respecto a la planificación. El hacer uso de metodologías ágiles nos ha permitido ir ajustando estas desviaciones en cada sprint. El elegir un framework de desarollo maduro, como son las librerías Qt, y confiar en software de la NASA como entorno de trabajo para presentación en pantalla también han sido puntos claves a la hora de conseguir el éxito del proyecto.

9.2. Lecciones aprendidas

Se destacan las siguientes lecciones aprendidas:

1. Las metodologías ágiles ayudan a centrar el foco en el desarrollo de un producto de calidad para el usuario final. Se reduce la generación de desperdicio consiguiendo un rendimiento óptimo para la inversión del cliente. Orientar el desarrollo a generar pequeños incrementos de funcionalidad, donde tiene una alta importancia la refactorización, ayuda a tener un producto de presupuesto ajustado y alta calidad.

2. Trabajar con historias de usuario, priorizadas desde el primer momento y siendo capaces, además, de aceptar el cambio durante el desarrollo, permite reducir los riesgos clásicos del desarrollo software [Jones, 2006].

3. El centrar totalmente el desarrollo en las necesidades del cliente puede tener como consecuencia un menor desempeño en tareas de análisis y diseño del sistema, delegando estas tareas en las iteraciones del producto introduciendo una posible deuda técnica en el proyecto. Como comenta Lefinegwell en Agile Software Requirements [Leffinegwell, 2010], esta es un aspecto a tener en cuenta a la hora de acometer el desarrollo ágil. El desarrollo ágil es mal entendido cuando altera la calidad del producto final, considerando como producto todo el desarrollo del mismo.

4. Las nuevas tecnologías, como OpenMP, ayudan a realizar tareas muy complejas con un mínimo esfuerzo de tiempo. Sólo es necesario conocer bien la técnica para poder aplicarla a los diferentes problemas a resolver.

5. Disponer de pruebas de regresión nos ha permitido asegurar el buen desempeño del sistema en cada incremento funcional y en cada refactorización.

6. Usar herramientas de seguimiento y control de proyectos, como *Taiga* y emphBitbucket han facilitado el desarrollo del proyecto. Han ofrecido en todo momento una visión global del proyecto que nos ha permitido resolver en todo momento aquellos aspectos que eran más prioritarios.

9.3. Trabajo futuro

El trabajo futuro de RaViGEn se centra en la ampliación de sus capacidades funcionales. Cabe destacar:

- Generar vídeo radar asociado al entorno físico de la simulación. Por ejemplo, es de gran ayuda para el adiestramiento, poder generar vídeo radar a partir de información cartográfica y modelos digitales de terreno.
- Procesar entidades DIS de entorno meteorológico. La lluvia es un aspecto influyente en la señal de vídeo radar. Aunque el estado actual del proyecto permite generar ruido radar equivalente al que produce la lluvia, su procesamiento automático aportaría valor al usuario.
- Mejorar la extensibilidad de formatos. Actualmente se incorpora nuevo código C++ para proce nuevos formatos radar. El poder incluir, en tiempo de ejecución, nuevos formatos leidos desde una especificación en texto aportaría valor a la capacidad de extensión del producto.

Apéndices

Apéndices A

Iniciación del proyecto

Tal como presenta J. Ramusson en su libro *The Agile Samurai* [Ramusson, 2010], en un proyecto ejecutado haciendo uso de metodologías ágiles es de vital importancia que el equipo que desarrolla el producto esté correctamente orientado a la consecución de los objetivos desde la concepción del mismo. Al comienzo de los proyectos las personas que forman parte del proyecto tienen sus propias aproximaciones a la idea que se quiere desarrollar haciendo necesario establecer una visión común del mismo.

Para conseguir este objetivo se utilizan distintas técnicas que facilitan la comunicación de los objetivos, visión y contexto del proyecto para que todos los implicados puedan tomar las decisiones correctas a lo largo de la vida del producto conocidas como *Inception Desk* [Ramusson, 2010]. En este caso, al encontrarnos en un proyecto unipersonal, desarrollaremos estas técnicas para conseguir que las decisiones sean lo más certeras posibles buscando el mayor retorno posible de la inversión de tiempo realizada en la elaboración de este Trabajo Fin de Máster.

A.1. Escritorio de Iniciación (Inception Desk)

A.1.1. ¿Por qué estamos aquí? (Why we are here?)

Para crear un inyector de video radar a partir de entidades DIS que permita el adiestramiento en entornos marítimos.

A.1.2. Elevator Pitch

Para dotaciones de buques
Que necesitan ser adiestradas
El producto RaViGEn
Es un *SimulationApplication* dentro del entorno DIS
Qué produce señal digital de video radar en un entorno de simulación, con un consumo de red controlado y de bajo coste
Diferente a la competencia porque necesita desplegar una menor infraestructura
Nuestro producto no tiene competencia

A.1.3. NOT list

- RaViGEn no es un simulador de video radar de costa. No está en el ámbito de este proyecto generar las montañas y playas detectadas.
- No genera pulsos eléctricos. Es un producto software que cumple con formatos de generación de video radar digital definidos en documento de interfaz para tal propósito.
- Las utilidades de visualización entregadas con el producto son de depuración. No forman parte del producto final.
- No es parte del proyecto modelar el conjunto de plataformas definidas en el estándar. Sólo ofrece el mecanismo para incluir nuevos modelos incluyendo ejemplos.

A.1.4. Show the solution

Figura A.1.

A.1.5. What keeps us up at night

- El ámbito del proyecto puede ser mayor del deseado como Trabajo Fin de Máster.
- La necesidad de realizar cálculos de estabilización y corrección puede suponer un riesgo para el proyecto.
- Los frameworks que implementan el estándar DIS no son completos en muchos casos.
- La suite presentación está pendiente de resolver pudiendo ser problemática.
- Se necesita una imagen de un radar real que sirva para validar nuestro producto. El resultado debe ser fidedigno.

A.1.6. Size it up & Show what it's going to take

El coste de los trabajadores es obtenido de la tabla salarial de la Universida de Cádiz. [CCOO, 2010]

Concepto	**Cantidad**	**Comentarios**
Puntos por Sprint	37,50 puntos	Velocidad del equipo
Sprints	8,00	
Capacidad	300,00 puntos	12 créditos
Meses	2,00	Al 100 % de dedicación
Puntos por mes	150,00	Según creditaje
Estimación total	331,00 puntos	
Puntos por mes	165,50	A tiempo completo
Margen	-15,50 puntos	Sobretrabajo
Coste salario / hora	20,00 €	
Coste salario / año	34000,00 €	
Equipo	1 persona	
Coste equipo total año	34000,00 €	
Coste equipo año	37513,33 €	
Dedicación	1,103	10 % de horas extras

Concepto	Cantidad	Comentarios
Coste equipo sprint	781,53 €	
Coste equipo mes	3126,11 €	
Coste equipo punto	20,84 €	
Coste equipo proyecto	6252,22 €	
Horas mes	141,67 h	
Horas año	1700,00 h	
Costes indirectos	375,13 €	
Coste asesoría	937,83 €	
Coste total	7565,19 €	
Salario aprobado	6000 €	Según creditaje (12)
Margen inversión	-252,22 €	Sólo costes directos

Tabla A.1: Mediciones del proyecto

A.1.7. Be clear on what's going to give

El proyecto se estima en unas 300 horas de trabajo (12 créditos). El alcance será variable dentro de los parámetros que permite la normativa que regula el desarrollo de los Trabajos Fin de Máster en la Universidad de Cádiz.

- El proyecto se centra en la generación de video radar en varios formatos a partir de la información DIS.
- El flujo de información de salida será controlable y limitable.
- El proyecto no tiene como objetivo desarrollar un interfaz hombre máquina que presente video radar aunque desarrollará los controles de configuración necesarios.
- La inserción de modelos será realizada a modo de demostración sin ser ámbito de este proyecto modelar el conjunto de plataformas propuestas por el estándar DIS.
- La generación de simuluaciones DIS no es parte de este proyecto aunque tengamos que realizar simuladores que soporten las pruebas del sistema.

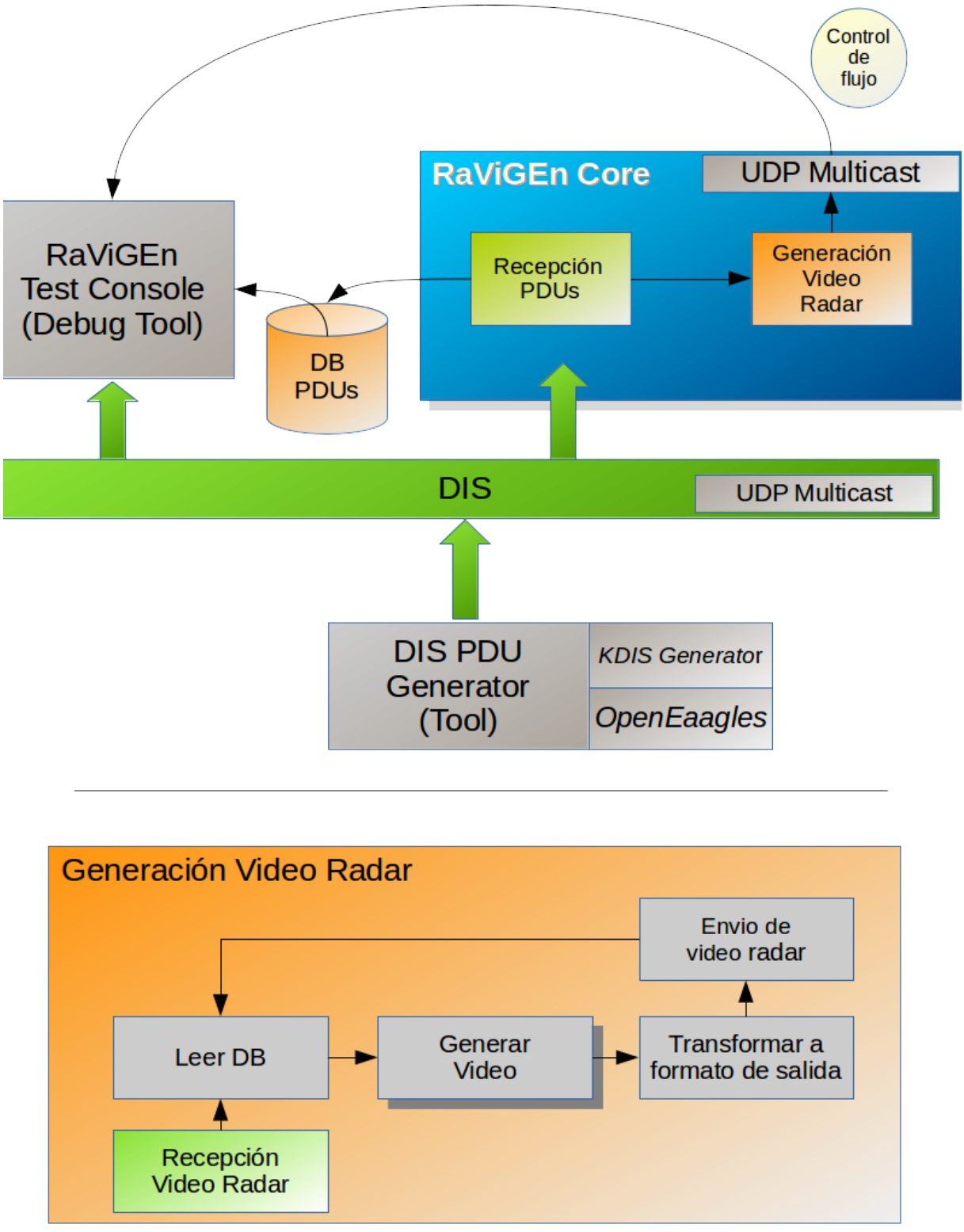


Figura A.1: Solución conceptual

Apéndices B

Análisis del estándar DIS

B.1. Introducción

La versión actual de Distributed Interactive Simulation (DIS) define 72 tipos diferentes de Protocol Data Units (PDUs) agrupadas en 13 familias. Entre ellas encontramos La familia de información de entidades, de guerra, de logística, de gestión de la simulación, de comunicaciones por radio, etc.

El estándar DIS, en la IEEE 1278.3 [NATO Standardization Agency, 1996], considera que una Simulation Application, como RaViGEn, no necesita procesar todas las PDUs para ser considerada una aplicación que cumpla con el estándar.

Aunque todas son susceptibles de influenciar la firma radar de un contacto en la realidad no existen aplicaciones DIS que contemplen el uso de todas las familias de PDUs. Un ejemplo es la propuesta de implementación mínima DIS Francesa [Direction generale de l'armament, 2010] desarrollada por el Ministerio de la Defensa francés. Asumen esta realidad y proponen a sus contratistas cumplir con la recepción y envío de un subconjunto común de PDUs para conseguir que los productos de simulación franceses sean útiles e interoperables.

B.2. Objeto

El objeto de este análisis es decidir cuál es el número mínimo de PDUs que RaViGEn debe procesar para que el sistema produzca una señal de vídeo radar básica para el adiestramiento. En el caso de existir varias PDUs a implementar se priorizarán por importancia.

El proyecto es desarrollado siguiendo metodologías ágiles. En el caso de que la planificación y las tareas lo permitan, incluir la mayoría de las PDUs posible dentro del procesamiento del sistema será un asunto a tratar en las reuniones de preparación del sprint.

B.3. Desarrollo

En este apartado se realiza el análisis de familias de PDUs DIS y su aplicabilidad al proyecto.

B.3.1. Entity Information Interaction

La familia Entity Information Interaction es una de las más importantes del estándar. Esta familia recoge la PDU Entity State que es la base de la representación de las plataformas en el sistema de simulación.

PDU	Prioridad	Análisis
Entity State (ESPDU)	1	Es la PDU fundamental para la generación de vídeo radar. En esta PDU se transmiten los datos de posición, orientación y tipo de plataforma.
Entity State Update	5	La ESPDU es una PDU con toda la información de plataforma. Para ciertos sistema sólo es necesaria la información de posición y orientación. Aunque es una PDU interesante para aquellas entidades que son controladas interactivamente, debido a las mejoras en las infraestructuras de red, existen muy pocos sistemas que la implementen.
Collision	9	La influencia de las colisiones contra PDUs no tienen un reflejo significativo en las imágenes radar más allá de la desaparición de la misma. Por este motivo marcamos esta PDU como no prioritaria.
Collision-Elastic	9	La influencia de las colisiones contra PDUs no tienen un reflejo significativo en las imágenes radar más allá de la desaparición de la misma. Por este motivo marcamos esta PDU como no prioritaria.

Tabla B.1: Entity information PDUs

B.3.2. Distributed Emission Regeneration

La familia Distributed Emission Regeneration se centra en transmitir las propiedades de las firma electromagnéticas y de radiofrencuencia de las plataformas. La imagen vídeo radar viene definida por dos aspectos fundamentales: la forma de la plataforma, definida por el retorno radar, y su firma electromagnética, que condiciona el retorno radar hasta el punto de poder crear interferencias.

Las PDUs de esta familia son de alta importancia para la generación de vídeo radar. El que una plataforma esté haciendo jamming a otra perturba su señal radar. Esta situación no es común en el adiestramiento pero consideramos fundamental el procesamiento de estas PDUs.

PDU	Prioridad	Análisis
IFF/ATC/NAV AIDS	3	Esta PDU simula las emisiones producidas por los sistemas de la plataforma. Modifican levemente la firma de la imagen radar
Electromagnetic Emission	2	En esta PDU se transmiten los datos propios de las emisiones electromagnéticas de la plataforma. Tiene un alto impacto en la generación de la escena radar
Designator	9	Esta PDU es considerada auxiliar. No se considera en el alcance de este proyecto.
Unserwater Acoustic	9	La señal acústica, propia de un sónar no tiene impacto en la generación de vídeo radar. El vídeo radar sólo detecta contactos aéreos y de superficie
Suplemental Emisison	8	Esta PDU es considerada auxiliar. Aunque puede transportar información acerca de la firma radar de las plataformas no se considera cien por cien estandarizada, siendo dependiente en muchos casos del fabricante, por lo que no se considera en el alcance de este proyecto.

Tabla B.2: Distributed Emission Regeneration PDUs

B.3.3. Entity Management

Las PDUs de gestión de entidades recogen las capacidades de agrupación del estándar. Esta agrupación, realizada de manera lógica, es irrelevante para este proyecto.

PDU	Prioridad	Análisis
Aggregate State	10	Para el caso de estudio RaViGEn no tiene interés conocer las agregaciones de plataformas.
IsGroupOf	10	Para el caso de estudio RaViGEn no tiene interés conocer las agregaciones de plataformas.
Transfer Control Request	10	RaViGEn como *simulation application* no distribuirá su carga por el número de plataformas que controla si no mediante el uso de configuración por sectores de vigilancia.
IsPartOf	10	Para el caso de estudio RaViGEn no tiene interés conocer las agregaciones de plataformas.

Tabla B.3: Entitiy Management PDUs

B.3.4. Live Entity

Las PDUs dan soporte a la apariencia de las entidades vivas en la simulación (personas, animales, plantas...) no son de interés para la generación de vídeo radar.

B.3.5. Logistics

Las PDUs definidas para realizar un juego de guerra, donde se pueden realizar peticiones de reparación o de suministro, no son de interés para la generación de una escena de vídeo radar.

B.3.6. Minifield

La familia *Minifield* ha sido creada para dar soporte a los juegos de guerra centrados en los campos de minas. Los campos de minas no son de interés para la generación de una escena de vídeo radar.

B.3.7. Radio Communications

Tal como son definidas las comunicaciones por radio, en las PDUs que soportan esta actividad en DIS, no tienen impacto en la generación de vídeo radar. Serán sus PDUs "hermanas" las que definirán la señal electromagnética que producen los equipos de radio.

B.3.8. Simulation Management

La familia de PDUs Simulation Management recoge todas aquellas PDUs que son destinadas al control de la simulación: inicio, marcha, parada... Dentro de esta familia también se incluyen aquellas PDUs definidas para la consulta de valores de los campos de las entidades.

PDU	Prioridad	Análisis
Start/Resume	3	Esta PDU marca el comienzo de un nuevo ejercicio. En el inicio de una nueva simulación es interesante borrar los datos residuales de anteriores simulaciones.
Stop/Freeze	4	Esta PDU marca el final de un nuevo ejercicio. Puede ser suficiente con eliminar los datos residuales al comienzo de un ejercicio. Por ese motivo marcamos como 4 la prioridad de implementación de esta PDU.
Acknowledge	5	RaViGEn será definido como *simulation application* dentro de la red DIS por lo que no tendrá que reportar el ack de las acciones que realice mediente esta PDU.
Set Data	9	RaViGEn no actúa como motor de la simulación y no establecerá datos de las plataformas.
Data	9	RaViGEn no actúa como motor de la simulación y no establecerá datos de las plataformas.

PDU	Prioridad	Análisis
Data Query	9	RaViGEn no actúa como motor de la simulación y no responderá a la petición de datos de las plataformas.
Action Request	9	RaViGEn no actúa como motor de la simulación y, en principio, no usaremos la PDU Action Request para realizar configuraciones.
Action Response	9	RaViGEn no actúa como motor de la simulación y, en principio, no usaremos la PDU Action Request para realizar configuraciones.
Create Entity	7	Aunque el motor de la simulación puede avisar de la creación de una entidad RaViGEn dará de alta una entidad al recibir un identificador no conocido en la Entity State PDU. No tiene interés el ser avisados previamente de la generación de un nuevo contacto.
Remove Entity	3	Es interesante saber cuando ha desaparecido una entidad para eliminar la información asociada de la base de datos de RaViGEn.
Event Report	9	El informe de eventos del motor de la simulación no aporta información elevante para la generación de vídeo radar.
Comment	10	No es de interés para el generador de video radar RaViGEn conocer los comentarios asociados a las plataformas.

Tabla B.4: Simulation Management PDU

B.3.9. Warfare

Las PDUs de la familiar de *Warfare* se definen para poder simular las acciones de guerra asociadas al disparo y la detonación. Estas actividades tienen un impacto residual en la simulación de vídeo radar por lo que no serán implementadas en este proyecto.

B.3.10. Synthetic Environment

Las PDUs de la familia *Synthetic Environment* se centran en ofrecer a las aplicaciones que gestionan la simulación el entorno atmosférico. De este modo se puede conseguir una simulación más real donde el viento o la lluvia afecte a la operación.

El entorno atmosférico, como la definición de la corriente y la lluvia, son aspectos que, aunque en un primer alcance no son fundamentales, sí tienen relevancia en la simulación de vídeo radar.

B.4. Conclusión

Las PDUs fundamentales para la producción de vídeo radar son las referentes a la publicación de la posición, orientación y tipo de plataforma de las entidades DIS. Además se deben considerar aquellas PDUs, como la *Electromagnetic Emission* que influyen a la hora de presentar la firma radar de una plataforma. Si la planificación lo permite tener en cuenta las PDUs de entorno sintético, que hará que la presentación de vídeo radar varíe según unas condiciones climáticas concretas, nos permitirán tener un sistema de simulación más próximo a la realidad.

Sólo es necesario procesar las PDUs de control de la simulación para limpiar el escenario producido. Tampoco es necesario saber cuando se ha creado una entidad. Con la primera recepción es suficiente para que sea dada de alta en la base de datos a procesar por lo que se descarta realizar la implementación del procesamiento de los mensajes dedicados al control de la simulación.

Apéndices C

Formatos de vídeo radar

C.1. Introducción

Los formatos digitales de vídeo radar permiten la distribución de esta información a través de capas de transporte como redes LAN. El mundo del vídeo radar, o vídeo en crudo, procede de una historia eminentemente analógica. Cuando la industria decide modernizar este componente fundamental de su sistema cada fabricante decide definir su propio formato con una estructura muy próxima al funcionamiento del radar que producen. Incluso existen fabricantes que deciden no ofrecer estas capacidad y es un postprocesamiento de la señal radar la que permite la inclusión de esta funcionalidad en la integración de sistemas.

Organizaciones internacionales, como Eurocontrol, hacen un esfuerzo importante para definir formatos de vídeo radar estándares y abiertos. Aún habiendo realizado este esfuerzo, actualmente no existe un estándar de facto en la industria. La solución de vídeo radar depende fuertemente de la decisión de la integración de un producto concreto.

Debido a esta situación RaViGEn, para que sea una herramienta con aceptación, debe soportar el envío de diferentes formatos de vídeo radar.

C.2. Objeto

El objetivo de este análisis es conocer los principales formatos de vídeo radar del mercado para poder tomar decisiones del modelado conceptual de los datos que manejará RaViGEn y su relación con la fuente de información DIS.

C.3. Desarrollo

C.3.1. La detección radar

La señal radar es procesada en distintas etapas. Cada una de las etapas del procesamiento de la señal radar permite ofrecer varios productos diferentes con un nivel de grano acorde con dicha etapa. El tipo de radar que distribuye la información condiciona el tipo de vídeo a enviar. Por ejemplo, existen radares de superficie, que no ofrecen valores de altura precisos, que descartan este valor en la información que envían.

La señal radar es recepcionada y almacenada en un elemento llamado blob. El blob es una figura geométrica definida por dos ángulos y dos medidas de rango desde un punto de referencia siendo relativo a dicho punto. Es, habitualmente, la representación de un echo radar y no suele presentar medida de altura.

El plot es el resultado de realizar procesamiento sobre los blobs. Se procesa un conjunto de blobs y, según ciertos patrones, el radar crea un elemento sintético representado un un sólo punto definido bajo sus tres coordenadas, usando como punto de referencia el origen, en el caso de que el radar sea capaz de ofrecer altura.

Los formatos radiales se apoyan en los dos formatos presentados para ser definidos. La técnica consiste en agrupar todos los blobs o plots existentes entre dos ángulos disminuyendo la cantidad de información que se incluye en el mensaje. No sería necesario enviar, por cada detección, los ángulos -o el ángulo- en el que se cuenta, sólo hay que ir definiendo la distancia a la que se encuentra el contacto. El video radial con blobs suele dar mejor resultado que con plots debido a la precisión de la definición de los ángulos que se manejan. Los radios con plots tienen zonas muertas en puntos alejados del punto de referencia.

Como dato común a todos los tipos de información de radar cruda se acompaña un nivel de intensidad normalizado entre cero y doscientos cincuenta y cinco existiendo implementaciones que cubren el espectro de los 32 bits.

C.3.2. Formatos de vídeo radar

Plot. Asterix. CAT 48. Transmission of Monoradar Target Reports

Es un estándar abierto definido por Eurocontrol [EUROCONTROL, 2012]. Su principal característica es ofrecer un contacto previo a la creación de una posible traza en un determinado punto. Ofrece una gran cantidad de información adicional (datos IFF, datos de identificación aérea, identificador de traza, ...) que no es necesaria para la generación de vídeo radar.

Este formato soporta el reporte de altura en unidades de 25 pies en un campo de 14 bits. La posición es informada mediante coordenadas polares inclinadas desde el punto de referencia definida por un valor RHO -con precisión 1/256 NM- y un ángulo THETA -con una precisión de $1/2^{16}$ grados-. Mediante esta información polar y la altura podemos proyectar de manera estereográfica la posición del blanco.

Actualmente se encuentra en su versión de definición 1.21, publicada en julio de 2012, de gran madurez, estable en su definición con poca necesidad de cambios.

Radial blob. Asterix. CAT 240. Radar Video Transmission

Es un estándar abierto definido por Eurocontrol [EUROCONTROL, 2009]. Cuando la industria necesita definir un formato de vídeo radar utiliza esta especificación que es considerada más pura al distribuir una información del radar menos procesada que con los plots. Define dos ángulos bajo el término de azimut con una precisión de 16 bits y un valor de rango que representa el origen de los datos radar para el que reserva 32 bits.

Se encuentra en su versión de definición 1.1 de mayo de 2009. Gracias a recoger las características comunes de los vídeo radares en sólo diecisiete páginas ha sufrido pocos cambios desde su creación en el mismo 2009.

Radial plot. SPX. Cambridge Píxel

Cambridge Píxel es una empresa experta en el manejo de datos de radar. Definieron un formato de radar previo al CAT 240 de Asterix pero con muchas similitudes. La principal diferencia es que envían radios unitarios sin comunicar dos cotas angulares. Cada radio se muestrea en un ángulo concreto.

La principal desventaja de este formato es que, al no ofrecer un "ancho" de radio, queda como responsabilidad por parte de la presentación de rellenar los huecos que la definición de la información deja sin cubrir. En otras palabras, a largas distancias, los arcos entre un radio y el siguiente son lo suficientemente largos para observar zonas huecas en la señal radar.

C.4. Conclusión

Los distintos formatos de radar tienen suficiente información en común para poder realizar un modelo de procesamiento de la información interno para RaViGEn. Este formato interno de RaViGEn tendrá la suficiente información para ser convertido en cualquiera de las categorías definidas por la industria.

El principal caso de estudio debe ser el Asterix CAT 240. Es el caso más usado para distribuir video radar y, además, es el que más se aproxima a la definición canónica de vídeo Radar.

Se utilizará, dentro de RaViGEn, una formato estándar que, de manera genérica, sirva para validar y depurar la implementación del producto.

Apéndices D

Análisis de alternativas de frameworks DIS

D.1. Objetivo

El objetivo de este documento es comparar las distintas librerías existentes que facilitan la utilización del protocolo DIS (Distributed Interactive Simulation), con el fin de simplificar la elección de una librería para el uso de DIS en el proyecto RaViGEn.

D.2. Introducción

Se ha establecido como requisito para RaViGEn el uso del protocolo DIS. Este protocolo está definido en el estándar IEEE 1278.1 (1998) y posteriormente se realizó una actualización, IEEE 1278.1a (2012), que extiende el protocolo.

El protocolo DIS define una serie de tipos de mensaje, llamados PDUs, que permiten estandarizar la comunicación entre simulaciones pertenecientes a una simulación distribuida en el ámbito de los juegos de guerra.

Ante la necesidad de utilizar DIS se ha buscado si existen librerías que faciliten el uso del protocolo y se han encontrado 3 librerías candidatas a ser usadas: Open-DIS, KDIS y OpenEaagles.

El método de evaluación consistirá en definir una serie de criterios de evaluación, asignarle un peso a cada uno y hacer una media ponderada. Los criterios se valorarán en el rango [0, 10].

La funcionalidad mínima requerida consiste en: - Leer y modificar con facilidad los campos de las PDUs - Empaquetar/desempaquetar PDUs - Soporte de IEEE 1278.1a (2012)

D.3. Análisis de alternativas

Los criterios de evaluación que se van a utilizar y sus correspondientes pesos propuestos son:

Criterio de evaluación	Peso propuesto
Alcance	3

Criterio de evaluación	Peso propuesto
Facilidad de trabajo con PDU	4
Documentación	3
Plataformas	2
Lenguajes	1
Desarrollo y mantenimiento	2
Estándares DIS	2
Modularidad	2
Licencia	1

Tabla D.1: Criterios y ponderación.

D.3.1. Alcance

Funcionalidad que ofrece la librería. Lo mínimo exigido es que se puedan leer/modificar los campos de las PDUs y que se puedan empaquetar/desempaquetar.

Se puntúa con un 5 si la librería cumple con lo mínimo y con un 10 a la librería que ofrezca más funcionalidad que podamos utilizar además del mínimo.

- Open-DIS cumple con lo mínimo, pero no ofrece nada más.
- KDIS cumple con lo mínimo y además ofrece comunicaciones multicast, logueo de PDUs, conversiones y Dead Reckoning.
- OpenEaagles es un framework para simulaciones de juegos de guerra, ofrece mucha funcionalidad pero no cumple con lo mínimo exigido, no empaqueta/desempaqueta mensajes de tipo PDU.

Librería DIS	Puntuación
Open-DIS	5
KDIS	10
OpenEaagles	4

Tabla D.2: Valoración del alcance.

D.3.2. Facilidad de trabajo con PDUs

Facilidades para trabajar con los campos y valores de las PDUs, estructuras que mapeen los campos de las PDUs y enumerados para los valores.

Se dan 5 puntos si la librería tiene clases que mapeen los campos de las PDUs con métodos para trabajar con ellos, y 5 puntos si tiene enumerados que mapeen los definidos en el estándar.

- Open-DIS tiene clases para las PDUs y para los campos de las PDUs pero no tiene enumerados para los valores de campos que son enumerados en el estándar (en la versión C++).

- KDIS tiene clases para PDUs y campos, y además enumerados.
- OpenEaagles tiene algunas clases y unos pocos enumerados.

Librería DIS	Puntuación
Open-DIS	5
KDIS	10
OpenEaagles	3

Tabla D.3: Valoración de la facilidad de trabajo.

D.3.3. Documentación

Documentación disponible para el usuario de la librería.

Se dan 5 puntos si existe documentación de las clases (tipo Doxygen) y se dan puntos extra si existen tutoriales, ejemplos u otro tipo de documentación.

- Open-DIS tiene documentación javadoc de las clases correspondiente a la versión Java de la librería, y varios ejemplos de uso.
- KDIS tiene documentación de las clases generada con Doxygen y varios tutoriales.
- OpenEaagles tiene documentación generada con Doxygen y varios tutoriales, además existe un libro sobre el framework.

Librería DIS	Puntuación
Open-DIS	6
KDIS	8
OpenEaagles	10

Tabla D.4: Valoración de la documentación.

D.3.4. Plataformas

Plataformas que soporta la librería.

Se puntúa con 5 puntos si la librería soporta GNU/Linux y otros 5 si además soporta Windows. No se valoran el resto de plataformas.

Todas las librerías candidatas soportan tanto Windows como GNU/Linux.

Librería DIS	Puntuación
Open-DIS	10
KDIS	10
OpenEaagles	10

Librería DIS	Puntuación

Tabla D.5: Valoración de las plataformas soportadas.

D.3.5. Lenguajes

Lenguajes de programación soportados, ya sea nativamente o mediante bindings.

Se establece como requisito mínimo que se pueda utilizar desde C++, otorgándose 5 puntos en caso positivo. Se dan puntos extra si soporta otros lenguajes de programación.

- Open-DIS soporta C++, Java y C#.
- KDIS sólo soporta C++.
- OpenEaagles sólo soporta C++.

Librería DIS	Puntuación
Open-DIS	10
KDIS	5
OpenEaagles	5

Tabla D.6: Valoración de lenguajes soportados.

D.3.6. Desarrollo y mantenimiento

Frecuencia de actualización de los desarrolladores de la librería (commits) y resolución de bugs.

Se estima la puntuación según el nivel de actividad del repositorio de la librería y la existencia de herramientas para el seguimiento de cambios y bugs.

- Open-DIS tiene su repositorio en Sourceforge, con registro de commits y bugs. La versión C++ no está tan actualizada como la de Java (que va por la versión 4.0) y la última versión oficial en este momento (3.0) necesita unos pequeños cambios para que compile (añadir algunos ";" y un include de string.h). Hay un informe de bug abierto en el repositorio desde Abril de 2009 con este problema concreto.
- KDIS tiene su repositorio en Sourceforge, con registro de commits y bugs, y un buen ritmo de desarrollo a pesar de tener un único desarrollador.
- OpenEaagles tiene su repositorio en GitHub, con registro de commits y bugs, y mucha actividad, con varios desarrolladores.

Librería DIS	Puntuación
Open-DIS	2
KDIS	7

Librería DIS	Puntuación
OpenEaagles	10

Tabla D.7: Valoración madurez de desarrollo.

D.3.7. Estándares DIS

Las versiones del estándar de DIS que soporta la librería. Las versiones existentes son: Versión 5: IEEE 1278.1 (1995) Versión 6: IEEE 1278.1a (1998) (Extensión de la versión 5) Versión 7: IEEE P1278.1 (2012)

Se puntúa con un 5 si soporta la versión 6 (lo mínimo requerido) y con un 10 si además soporta la versión 7 o se está trabajando en su soporte.

- Open-DIS soporta las versiones 5 y 6.
- KDIS soporta las versiones 5, 6 y 7.
- OpenEaagles soporta varias PDUs de las versiones 5 y 6, pero ninguna versión al completo.

Librería DIS	Puntuación
Open-DIS	5
KDIS	10
OpenEaagles	2

Tabla D.8: Valoración Soporte estándares DIS.

D.3.8. Modularidad

Modularidad de la funcionalidad de la librería, la separación entre las distintas funciones ofrecidas y dependencias entre módulos.

Se valora la independencia del módulo correspondiente al trabajo con PDUs.

- Open-DIS tiene poco más que lo mínimo requerido y por tanto no tiene problemas de modularidad.
- KDIS añade varios extras a la funcionalidad requerida pero no son dependencias de la funcionalidad base.
- OpenEaagles está organizado como un conjunto de librerías. La librería de DIS depende de Basic y Simulation, existiendo un gran acople entre los módulos del framework.

Librería DIS	Puntuación
Open-DIS	10
KDIS	10

Librería DIS	Puntuación
OpenEaagles	2

Tabla D.9: Valoración modularidad.

D.3.9. Licencia

Conveniencia de la licencia con la que se ofrece la librería.

Se puntúa con un 5 si la licencia permite el uso de cualquier licencia en el software que enlaza con la librería, y con 10 puntos si además permite utilizar cualquier licencia si se modifica la propia librería de DIS.

- Open-DIS tiene licencia estilo BSD, que permite el uso de cualquier licencia si se modifica la librería o se enlaza con ella.
- KDIS tiene licencia LGPL, que permite el uso de cualquier licencia si se enlaza con la librería, pero obliga a entregar al cliente el código fuente modificado si se modifica la propia librería.
- OpenEaagles también tiene licencia LGPL.

Librería DIS	Puntuación
Open-DIS	10
KDIS	5
OpenEaagles	5

Tabla D.10: Valoración licencia.

D.4. Conclusiones

Al realizar la media ponderada de los criterios de evaluación el resultado es el siguiente:

Criterios	Open-DIS	KDIS	OpenEaagles
Alcance	5	10	4
Facilidad de trabajo con PDU	5	10	3
Documentación	6	8	10
Plataformas	10	10	10
Lenguajes	10	5	5
Desarrollo y mantenimiento	2	7	10
Estándares DIS	5	10	2
Modularidad	10	10	2
Licencia	10	5	5
Resultado	6,4	8,9	5,6

Tabla D.11: Resumen valoraciones.

Se decide el uso de la librería KDIS, que cumple todos los requisitos mínimos y añade funcionalidad extra que puede ser útil. Además es la única que proporciona enumerados para facilitar la interpretación de los valores de las PDUs.

También podría usarse Open-DIS, que cumple los requisitos mínimos, pero habría que hacer enumerados para no tener números "mágicos" en el código. Éstos podrían incluirse en la propia librería, ya que la licencia lo permite.

OpenEaagles queda descartado porque no cumple los requisitos mínimos y su librería de DIS está acoplada con el resto del framework.

Apéndices E

Controles RaViGEn Test Console

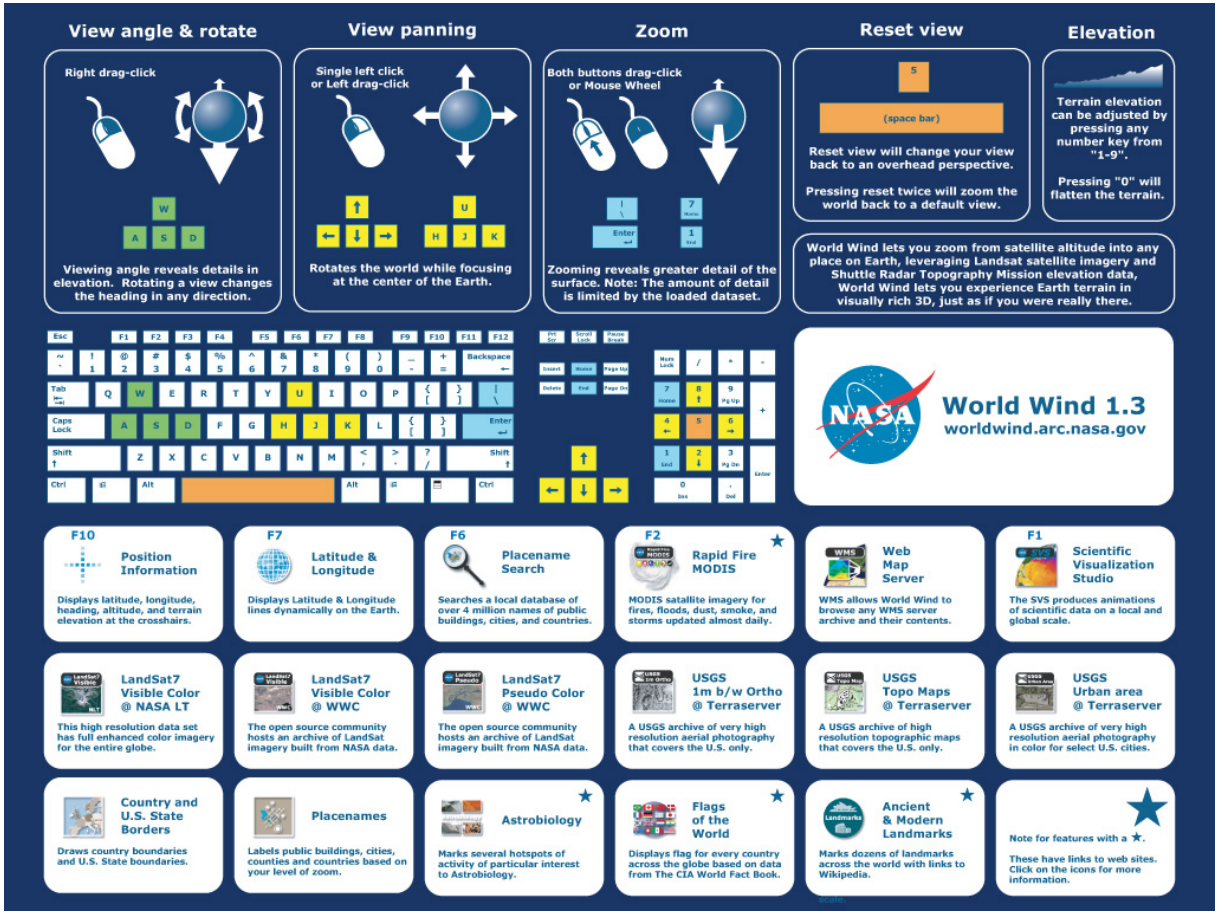


Figura E.1: Interfaz de Usuario. Controles WorldWind

Apéndices F

Documentación RaViGEn

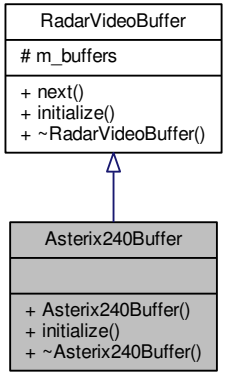
Contents

Chapter 1

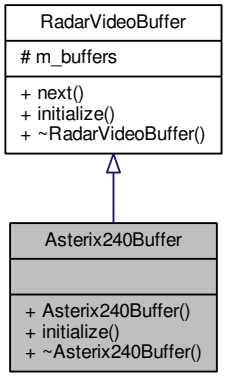
Class Documentation

1.1 Asterix240Buffer Class Reference

Inheritance diagram for Asterix240Buffer:



Collaboration diagram for Asterix240Buffer:



Public Member Functions

- **Asterix240Buffer** ()

  Asterix240Buffer *(p. 1) Constructor.*
- void **initialize** (void ∗setup)

  initialize This method is used for the buffer properties initialization using ***Asterix240BufferSettings*** *(p. 3) structure*
- ∼**Asterix240Buffer** ()

  ∼*Asterix240Buffer Destructor*

Additional Inherited Members

1.1.1 Detailed Description

Definition at line **22** of file **asterix240buffer.h**.

The documentation for this class was generated from the following files:

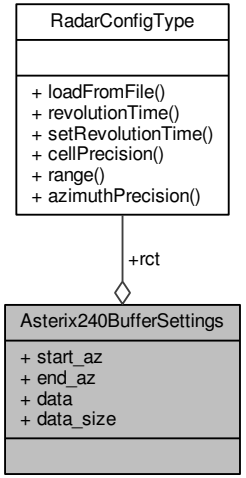
- libRadarVideoFormat/asterix240buffer.h
- libRadarVideoFormat/asterix240buffer.cpp

1.2 Asterix240BufferSettings Struct Reference

The **Asterix240BufferSettings** (p. 3) struct Properties definitifion of generic format video radar buffer.

```
#include <asterix240buffer.h>
```

Collaboration diagram for Asterix240BufferSettings:



Public Attributes

- double **start_az**
- double **end_az**

  Initial angle in degrees.
- char * **data**

  Final angle in degrees.
- unsigned **data_size**

  Buffer data message.
- const **RadarConfigType** & **rct**

  Data size of buffer data.

1.2.1 Detailed Description

The **Asterix240BufferSettings** (p. 3) struct Properties definitifion of generic format video radar buffer.

Definition at line **13** of file **asterix240buffer.h**.

The documentation for this struct was generated from the following file:

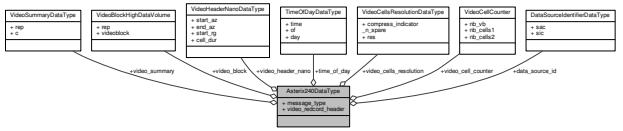
- libRadarVideoFormat/asterix240buffer.h

1.3 Asterix240DataType Struct Reference

The **Asterix240DataType** (p. 4) struct stores Asterix specification for CAT 240.

```
#include <asterix240format.h>
```

Collaboration diagram for Asterix240DataType:



Public Attributes

- **DataSourceIdentifierDataType data_source_id**
- quint8 **message_type**

  I240/010 : M : frn 1 :
- quint32 **video_redcord_header**

  I240/000 : M : frn 2 : id->002.
- **VideoSummaryDataType video_summary**

  I240/020 : M : frn 3 : id_secuencial.
- **VideoHeaderNanoDataType video_header_nano**

  I240/030 :N/A : frn 4 : rep->000.
- **VideoCellsResolutionDataType video_cells_resolution**

  I240/040 : O : frn 5.*
- **VideoCellCounter video_cell_counter**

  I240/048 : M : frn 7.
- **VideoBlockHighDataVolume video_block**

  *I240/052 : M : frn 11 : D = CELL_DUR * (START_RG + POSITION - 1) * c / 2 where c = 299 792 458 m/s.*
- **TimeOfDayDataType time_of_day**

1.3.1 Detailed Description

The **Asterix240DataType** (p. 4) struct stores Asterix specification for CAT 240.

Definition at line **80** of file **asterix240format.h**.

1.3.2 Member Data Documentation

1.3.2.1 VideoCellCounter Asterix240DataType::video_cell_counter

I240/048 : M : frn 7.

I240/049 : M : frn 8 : NB_VB -> NB_CELLS depending on (RES)

Definition at line **91** of file **asterix240format.h**.

The documentation for this struct was generated from the following file:

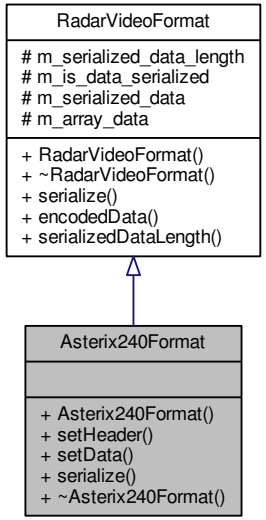
- libRadarVideoFormat/asterix240format.h

1.4 Asterix240Format Class Reference

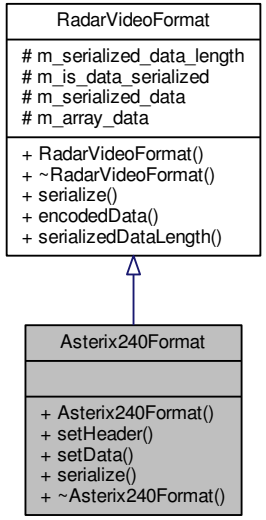
The **Asterix240Format** (p. 5) class implemets version 1.2 of Asterix Video Radar format CAT 240 published by Eurocontrol.

```
#include <asterix240format.h>
```


Inheritance diagram for Asterix240Format:



Collaboration diagram for Asterix240Format:



Public Member Functions

- **Asterix240Format** ()

  Asterix240Format *(p. 5) config the radar video format message for CAT 240 message.*
- void **setHeader** (double start_az, double end_az, unsigned start_rg, double cell_dur=0.↩ 000002373185)

  setHeader Describes the information included in the radar video message
- void **setData** (char ∗data, unsigned data_size)

  setData Establish the data in the message
- void **serialize** ()

  serialize Marshall the data in a CAT 240 format for multicast
- ∼**Asterix240Format** ()

  ∼Asterix240Format Destructor

Additional Inherited Members

1.4.1 Detailed Description

The **Asterix240Format** (p. 5) class implemets version 1.2 of Asterix Video Radar format CAT 240 published by Eurocontrol.

Definition at line **102** of file **asterix240format.h**.

1.4.2 Member Function Documentation

1.4.2.1 void Asterix240Format::setData (char ∗ *data*, unsigned *data_size*)

setData Establish the data in the message

Parameters

data	Buffer message
data_size	Size of buffer message

Definition at line **44** of file **asterix240format.cpp**.

Here is the caller graph for this function:

1.4.2.2 void Asterix240Format::setHeader (double *start_az*, double *end_az*, unsigned *start_rg*, double *cell_dur* = 0.000002373185)

setHeader Describes the information included in the radar video message

Parameters

start_az	Initial angle in degrees
end_az	Final angle in degrees
start_rg	Distance from the origin in meters
cell_dur	Configures radar range. (9000 metres by default)

Definition at line **33** of file **asterix240format.cpp**.

Here is the caller graph for this function:

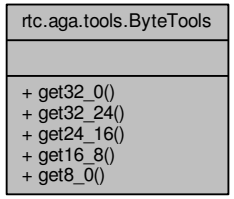
The documentation for this class was generated from the following files:

- libRadarVideoFormat/asterix240format.h
- libRadarVideoFormat/asterix240format.cpp

1.5 rtc.aga.tools.ByteTools Class Reference

ByteTools (p. 9) is a set of utilities to work with 32 bits words.

Collaboration diagram for rtc.aga.tools.ByteTools:



Static Public Member Functions

- static int **get32_0** (byte data0, byte data1, byte data2, byte data3)
  Returns a 32 bits integer built from raw data.
- static int **get32_24** (byte data)
  Returns a 32 bits integer built from raw data between 32 and 24 bits positions.
- static int **get24_16** (byte data)
  Returns a 32 bits integer built from raw data between 24 and 16 bits positions.
- static int **get16_8** (byte data)
  Returns a 32 bits integer built from raw data between 16 and 8 bits positions.
- static int **get8_0** (byte data)
  Returns a 32 bits integer built from raw data between 8 and 0 bits positions.

1.5.1 Detailed Description

ByteTools (p. 9) is a set of utilities to work with 32 bits words.

Definition at line **6** of file **ByteTools.java**.

The documentation for this class was generated from the following file:

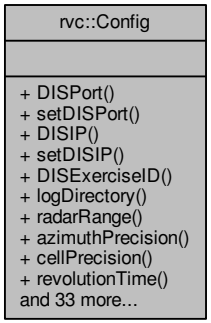
- RaViGEnTestConsole/RaViGEnTestConsole/src/rtc/aga/tools/ByteTools.java

1.6 rvc::Config Class Reference

This class provides the interface to manage config file. The **Config** (p. 10) class allow obtain configuration values to RaViGEn Core.

`#include <config.h>`

Collaboration diagram for rvc::Config:



Static Public Member Functions

- static unsigned **DISPort** ()
  Return DIS Port to listen UDP traffic stored in the configuration file.
- static void **setDISPort** (unsigned port)
  Set DIS Port to listen UDP traffic and stores the value in the configuration file.
- static QString **DISIP** ()

Return IP of multicast DIS group stored in the configuration file.

- static void **setDISIP** (QString ipv4)

  Set DIS IP to listen UDP traffic and stores the value in the configuration file.
- static unsigned **DISExerciseID** ()

  DISExerciseID return the exercise ID of reference.
- static QString **logDirectory** ()

  Reads log storage path.
- static unsigned **radarRange** ()

  Reads the radar range in metres.
- static double **azimuthPrecision** ()

  Reads the azimuth precision in degrees.
- static double **cellPrecision** ()

  Reads cell precision in meters.
- static double **revolutionTime** ()

  Reads the cycle radar time in milliseconds.
- static QString **loggingLevel** ()

  Reads the user defined logging level.
- static unsigned **entitiyIDReference** ()

  Reads the entity id of reference PDU.
- static unsigned **applDReference** ()

  Reads the application id of reference PDU.
- static unsigned **siteIDReference** ()

  Reads the site id of reference PDU.
- static int **widthCellsBuffer** ()

  Reads the width of bitmap cells buffer.
- static int **heightCellsBuffer** ()

  Reads the height of bitmap cells buffer.
- static int **dynamicOMP** ()

  Reads the OpenMP dynamic configuration.
- static int **numThreadsOMP** ()

  Reads the number of thReads for OpenMP configuration.
- static unsigned **maxLevelOfNoise** ()

  Reads the max value of level of noise for noise generator.
- static unsigned **minLevelOfNoise** ()

  Reads the min value of level of noise for noise generator.
- static QString **noiseGenerator** ()

  Reads the noise generator selected.
- static QString **radarVideoFormat** ()

  Reads the radar video message format choosen.
- static unsigned **simulatorApplicationID** ()

  Reads the application id for platform simulator application.
- static unsigned **simulatorSiteID** ()

  Reads the site id for platform simulator application.
- static double **simulationRate** ()

  Reads the rate of simulation for the simulator application.

- static int **simulatorPeriodTime** ()

  Reads the simulation period for the simulator application.
- static int **deadReckoningPeriod** ()

  Reads the dead reckoning period for the simulator application.
- static QString **DISSimulatorIP** ()

  Reads the IP configuration for DIS sender of simulatior application.
- static unsigned **radarVideoPort** ()

  Reads the port configuration for video radar generation.
- static bool **simulatorPlatformsOrientedVelocity** ()

  Reads the velocity (in m/s) for the oriented simulated platforms.
- static double **simulatorPlatformsVelocity** ()

  Reads the velocity (in m/s) for the simulated platforms.
- static QString **simulatorScenario** ()

  Reads the simulator scenario for the simulation.
- static double **scenarioRange** ()

  Reads the scenario range for the simulation.
- static unsigned **numPlatforms** ()

  Reads the number of platforms to scenario simulation.
- static double **simulatorOSReferenceVelocity** ()

  Reads the velocity of platform reference.
- static double **simulationOriginLat** ()

  Reads the center of the simulation.
- static double **simulationOriginLon** ()

  Reads the center of the simulation.
- static int **intervalThroughputControl** ()

  Reads the interval to take measurements of data processed to throughput control.
- static unsigned **throughputThreshold** ()

  Reads the threshold of data processed for throughput control.
- static unsigned **maxExceededAttemps** ()

  Reads the number of faults allowed in the throughtput control.
- static unsigned **intervalExceededAttemps** ()

  Reads the interval of the faults in the throughtput control.
- static double **slowDownFactor** ()

  Reads the factor of modifcation of revolution time in the throughtput control.
- static QString **cellFiller** ()

  Reads the cell filler for the video radar generator.

1.6.1 Detailed Description

This class provides the interface to manage config file. The **Config** (p. 10) class allow obtain configuration values to RaViGEn Core.

RESPONSIBILITIES: The member functions provide the functionality to perform the following:

- Returns the configuration values.

Definition at line **20** of file **config.h**.

1.6.2 Member Function Documentation

1.6.2.1 QString rvc::Config::DISIP ( ) [static]

Return IP of multicast DIS group stored in the configuration file.

Returns

String with IP of multicast group in v4 format.

Definition at line **41** of file **config.cpp**.

1.6.2.2 unsigned rvc::Config::DISPort ( ) [static]

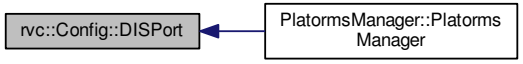
Return DIS Port to listen UDP traffic stored in the configuration file.

Returns

Integer with DIS port value.

Definition at line **22** of file **config.cpp**.

Here is the caller graph for this function:



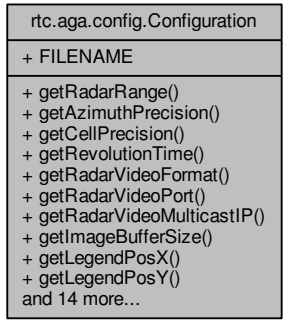
The documentation for this class was generated from the following files:

- libConfig/config.h
- libConfig/config.cpp

1.7 rtc.aga.config.Configuration Class Reference

Configuration (p. 14) class allow to the RaViGEn test console load the values stored the configuration file of the project.

Collaboration diagram for rtc.aga.config.Configuration:



Static Public Member Functions

- static int **getRadarRange** ()
  Returns the range of radar in metres.
- static double **getAzimuthPrecision** ()
  Reads the azimuth precision in degrees.
- static double **getCellPrecision** ()
  Reads cell precision in meters.
- static double **getRevolutionTime** ()
  Reads the cycle radar time in milliseconds.
- static String **getRadarVideoFormat** ()
  Reads the radar video message format choosen.
- static int **getRadarVideoPort** ()
  Read the port configuration for video radar generation.
- static String **getRadarVideoMulticastIP** ()
  Reads the IP of video radar generation.
- static int **getImageBufferSize** ()
  Reads the size of bitmap cells buffer.
- static int **getLegendPosX** ()
  Reads the x window position of the legend control.

- static int **getLegendPosY** ()

  Reads the y window position of the legend control.
- static int **getRefreshImagePeriod** ()

  Reads the y window position of the legend control.
- static double **getRadarLayerOpacity** ()

  Reads the layer opacity percentage for the video radar presentation.
- static int **getRegenerateImagePeriod** ()

  Reads the period time to regenerate.
- static String **getRadarVideoImageRender** ()

  Reads the kind of render to paint the radar video.
- static String **getDBFilePath** ()
- static String **getDBFileName** ()

  Reads the file name of database.
- static int **getRefreshDBPlatformsPeriod** ()

  Reads the period to read and paint the platforms from DB.
- static double **getSymbolScale** ()

  Reads the symbol scale for Platform symbology.
- static double **getSymbolOpacity** ()

  Reads the symbol opacity in percentage.
- static String **getSidcSymbol** ()

  Reads sidc identification according to vessel specification.
- static String **getRadarVideoLayerPresentationMode** ()

  Reads the presentation mode of video radar: 3D or 2D.
- static double **getAltitudeRadarVideoLayerPresentation** ()

  Reads the altitude of Radar 3D layer presentation.
- static double **getMaxAltitudeRadarVideoLayerPresentation** ()

  Reads the max altitude of radar video presentation in 3D mode.
- static double **getAltitudeSymbolPresentation** ()

  Reads the altitude of platform symbology presentation in 3D mode.

Static Public Attributes

- static final String **FILENAME** = "/home/user/ravigen/cfg/ravigen.ini"

1.7.1 Detailed Description

Configuration (p. 14) class allow to the RaViGEn test console load the values stored the configuration file of the project.

Definition at line **10** of file **Configuration.java**.

1.7.2 Member Function Documentation

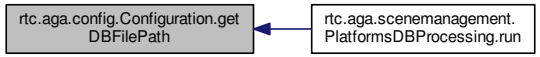
1.7.2.1 static String rtc.aga.config.Configuration.getDBFilePath ( ) [inline], [static]

Reads the file path of database

Definition at line **192** of file **Configuration.java**.

Here is the call graph for this function:

Here is the caller graph for this function:



The documentation for this class was generated from the following file:

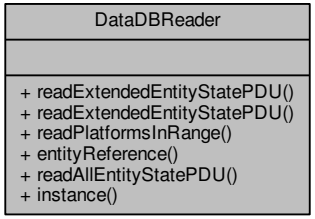
- RaViGEnTestConsole/RaViGEnTestConsole/src/rtc/aga/config/Configuration.java

1.8 DataDBReader Class Reference

The **DataDBReader** (p. 16) class is reponsible of retrieve data from database.

```
#include <datadbreader.h>
```

Collaboration diagram for DataDBReader:



Public Member Functions

- **Extended_Entity_State_PDU** * **readExtendedEntityStatePDU** (int site_id, int application_id, int entity_id)

  readExtendedEntityStatePDU read an entity state PDU defined by parameters site_id, application_id and entity_id
- Entity_State_PDU * **readExtendedEntityStatePDU** (int site_id, int application_id, int entity_id, bool &is_reference_pdu)

  readExtendedEntityStatePDU read an entity state PDU defined by parameters site_id, application_id and entity_id indicating if the reference pdu in the is_reference_pdu parameter
- std::vector< **Extended_Entity_State_PDU** * > **readPlatformsInRange** (const **RadarConfigType** &rct)

  readPlatformsInRange returns the platforms in range defined in rct parameter
- EntityIdentifier **entityReference** ()

  entityReference returns the entity marked as reference
- QList< **Extended_Entity_State_PDU** * > **readAllEntityStatePDU** ()

  readAllEntityStatePDU returns all entities on database

Static Public Member Functions

- static **DataDBReader** * **instance** ()

  instance returns the singleton pointer to the class

1.8.1 Detailed Description

The **DataDBReader** (p. 16) class is reponsible of retrieve data from database.

Definition at line **27** of file **datadbreader.h**.

The documentation for this class was generated from the following files:

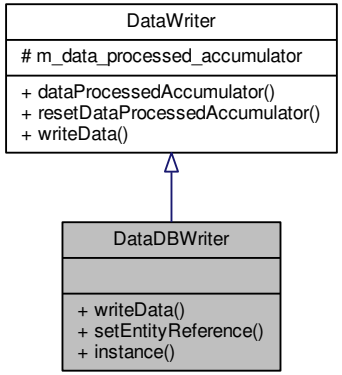
- libIOProcessing/datadbreader.h
- libIOProcessing/datadbreader.cpp

1.9 DataDBWriter Class Reference

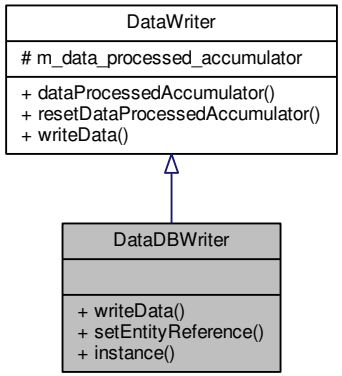
The **DataDBWriter** (p. 18) class is reponsible of store on data base.

```
#include <datadbwriter.h>
```

Inheritance diagram for DataDBWriter:



Collaboration diagram for DataDBWriter:



Public Member Functions

- bool **writeData** (char *data, unsigned size)
  writeData decode and write data in data base. The size of data, in bytes, have to be specified
- void **setEntityReference** (EntityIdentifier entity_id)
  setEntityReference stores the entity_id as reference platform of video radar generator

Static Public Member Functions

- static **DataDBWriter** * **instance** ()
  instance returns the singleton pointer to the class

Additional Inherited Members

1.9.1 Detailed Description

The **DataDBWriter** (p. 18) class is reponsible of store on data base.

Definition at line **25** of file **datadbwriter.h**.

The documentation for this class was generated from the following files:

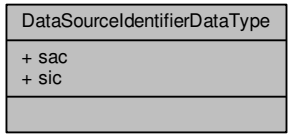
- libIOProcessing/datadbwriter.h
- libIOProcessing/datadbwriter.cpp

1.10 DataSourceIdentifierDataType Struct Reference

Light velocity.

```
#include <asterix240format.h>
```

Collaboration diagram for DataSourceIdentifierDataType:



Public Attributes

- quint8 **sac**
- quint8 **sic**

1.10.1 Detailed Description

Light velocity.

Definition at line **16** of file **asterix240format.h**.

The documentation for this struct was generated from the following file:

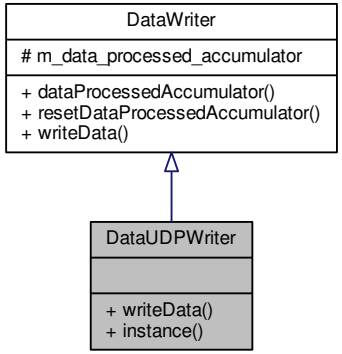
- libRadarVideoFormat/asterix240format.h

1.11 DataUDPWriter Class Reference

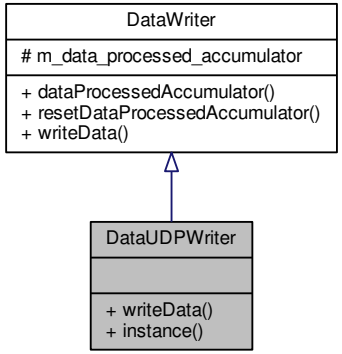
The **DataUDPWriter** (p. 21) class is reponsible of send data by multicast.

`#include <dataudpwriter.h>`

Inheritance diagram for DataUDPWriter:



Collaboration diagram for DataUDPWriter:



Public Member Functions

- bool **writeData** (char *data, unsigned size)

  writeData decode and write data in an udp socket. The size of data, in bytes, have to be specified

Static Public Member Functions

- static **DataUDPWriter** * **instance** ()

  instance returns the singleton pointer to the class

Additional Inherited Members

1.11.1 Detailed Description

The **DataUDPWriter** (p. 21) class is reponsible of send data by multicast.

Definition at line **16** of file **dataudpwriter.h**.

The documentation for this class was generated from the following files:

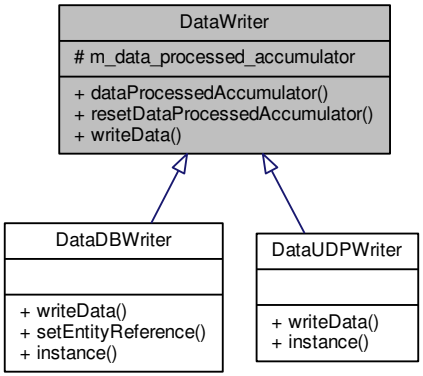
- libIOProcessing/dataudpwriter.h
- libIOProcessing/dataudpwriter.cpp

1.12 DataWriter Class Reference

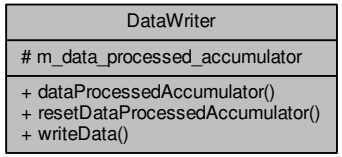
The **DataWriter** (p. 22) abstract class is the interface for data writers.

`#include <datawriter.h>`

Inheritance diagram for DataWriter:



Collaboration diagram for DataWriter:



Public Member Functions

- unsigned **dataProcessedAccumulator** () const

  dataProcessedAccumulator return the amount of data processed by data writer since the las reset data accumulator
- void **resetDataProcessedAccumulator** ()

  resetdataProcessedAccumulator establish the data procesed accumulator to zero
- virtual bool **writeData** (char ∗data, unsigned size)=0

  writeData decode and write data. The size of data, in bytes, have to be specified

Protected Attributes

- unsigned **m_data_processed_accumulator**

1.12.1 Detailed Description

The **DataWriter** (p. 22) abstract class is the interface for data writers.

Definition at line **7** of file **datawriter.h**.

The documentation for this class was generated from the following files:

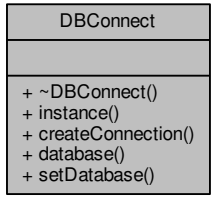
- libIOProcessing/datawriter.h
- libIOProcessing/datawriter.cpp

1.13 DBConnect Class Reference

This class provides the interface to create and manage databases.

`#include <dbconnect.h>`

Collaboration diagram for DBConnect:



Public Member Functions

- ∼**DBConnect** ()

Static Public Member Functions

- static **DBConnect** ∗ **instance** ()
- static void **createConnection** ()
- static QSqlDatabase **database** ()
- static void **setDatabase** (QSqlDatabase **database**)

1.13.1 Detailed Description

This class provides the interface to create and manage databases.

The **DBConnect** (p. 24) class is a database containing info of all elements in the system. The database is populated with data from the XML file. At running mode the data will be updated with data from DIS.

RESPONSIBILITIES: The member functions provide the functionality to perform the following:

- Creation of the connection to the database.
- Creation of the tables.

Definition at line **21** of file **dbconnect.h**.

1.13.2 Constructor & Destructor Documentation

1.13.2.1 DBConnect::~DBConnect ( )

Class destructor.

Destroy an instance of the class.

1.13.3 Member Function Documentation

1.13.3.1 void DBConnect::createConnection ( ) [static]

Create the connection to the database.

Definition at line **57** of file **dbconnect.cpp**.

Here is the call graph for this function:

1.13.3.2 QSqlDatabase DBConnect::database ( ) [static]

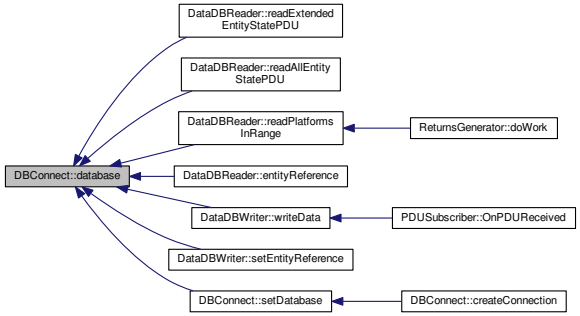
If you call this method from any thread but the application's main thread, a new connection to the database will be created. The connection will automatically be torn down once the thread finishes.

Returns

QSqlDatabase connection to the database.

Definition at line **165** of file **dbconnect.cpp**.

Here is the caller graph for this function:



1.13.3.3 DBConnect * DBConnect::instance ( ) [static]

Return a instance of the class.

Return a instance of the class.

Returns

DBConnect (p. 24) The instance of the class.

Definition at line **27** of file **dbconnect.cpp**.

1.13.3.4 void DBConnect::setDatabase (QSqlDatabase *database*) [static]

Sets the database used by the current process.

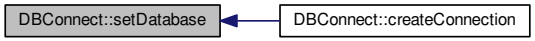
Parameters

database	Specifies the database.

Definition at line **190** of file **dbconnect.cpp**.

Here is the call graph for this function:

Here is the caller graph for this function:



The documentation for this class was generated from the following files:

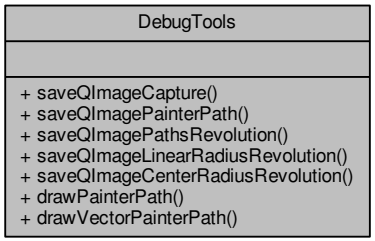
- libRVDB/dbconnect.h
- libRVDB/dbconnect.cpp

1.14 DebugTools Class Reference

The **DebugTools** (p. 27) class store bitmaps of each process state for RaViGEn debugging.

`#include <debugtools.h>`

Collaboration diagram for DebugTools:



Static Public Member Functions

- static void **saveQImageCapture** (const QImage &image, QString file_name)

  saveQImageCapture stores image in the file_name file
- static void **saveQImagePainterPath** (QImage &image, const QPainterPath &path, QString file_↩ name, QColor color)

  saveQImagePainterPath stores an image in file_name file given a composed path in
- static void **saveQImagePathsRevolution** (QImage &image, **Revolution** ∗rev)

  saveQImagePathsRevolution stores image in a file defined by rev
- static void **saveQImageLinearRadiusRevolution** (**Revolution** ∗rev)

  saveQImageLinearRadiusRevolution stores image in a file defined by rev in a linear representation
- static void **saveQImageCenterRadiusRevolution** (**Revolution** ∗rev)

  saveQImageCenterRadiusRevolution stores image in a file defined by rev in a radius center reference representation
- static void **drawPainterPath** (const QPainterPath &path, QPainter &painter, QColor color)

  drawPainterPath Helper function to draw a path in a color with a painter
- static void **drawVectorPainterPath** (std::vector< QPainterPath ∗ > ∗paths, QPainter &painter, Q↩ Color color)

  drawVectorPainterPath Helper function to draw paths in color with a painter

1.14.1 Detailed Description

The **DebugTools** (p. 27) class store bitmaps of each process state for RaViGEn debugging.

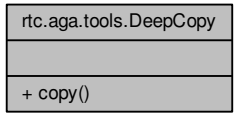
Definition at line **17** of file **debugtools.h**.

The documentation for this class was generated from the following files:

- libDebugTools/debugtools.h
- libDebugTools/debugtools.cpp

1.15 rtc.aga.tools.DeepCopy Class Reference

Collaboration diagram for rtc.aga.tools.DeepCopy:



Static Public Member Functions

- static Object **copy** (Object orig)

1.15.1 Detailed Description

[1] `http://javatechniques.com/blog/faster-deep-copies-of-java-objects/`
Utility for making deep copies (vs. clone()'s shallow copies) of objects. Objects are first serialized and then deserialized. Error checking is fairly minimal in this implementation. If an object is encountered that cannot be serialized (or that references an object that cannot be serialized) an error is printed to System.err and null is returned. Depending on your specific application, it might make more sense to have copy(...) re-throw the exception.

Definition at line **16** of file **DeepCopy.java**.

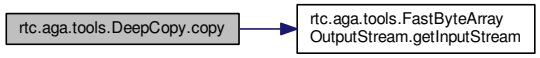
1.15.2 Member Function Documentation

1.15.2.1 static Object rtc.aga.tools.DeepCopy.copy (Object *orig*) [inline], [static]

Returns a copy of the object, or null if the object cannot be serialized.

Definition at line **22** of file **DeepCopy.java**.

Here is the call graph for this function:



The documentation for this class was generated from the following file:

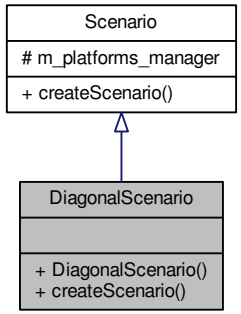
- RaViGEnTestConsole/RaViGEnTestConsole/src/rtc/aga/tools/DeepCopy.java

1.16 DiagonalScenario Class Reference

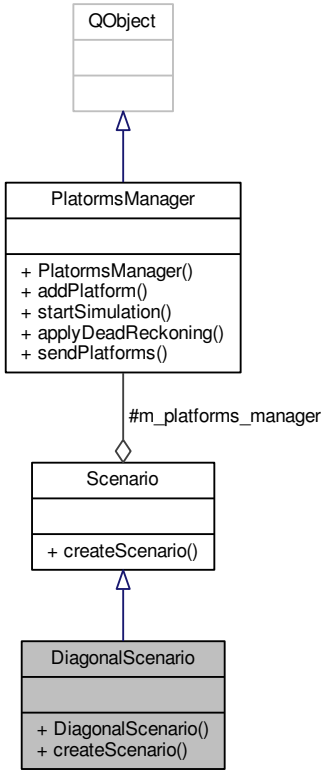
The **DiagonalScenario** (p. 30) class implements a scenario to test based on two ortogonal diagonals.

`#include <diagonalscenario.h>`

Inheritance diagram for DiagonalScenario:



Collaboration diagram for DiagonalScenario:



Public Member Functions

- **DiagonalScenario** (**PlatormsManager** ∗manager)

  DiagonalScenario *(p. 30) class method. Needs manager as paremeter to include the platforms in the scenario.*

- void **createScenario** (double range=0.0, unsigned num_platforms=10)

  createScenario creates a scenario based on range and num_platforms

Additional Inherited Members

1.16.1 Detailed Description

The **DiagonalScenario** (p. 30) class implements a scenario to test based on two ortogonal diagonals.

Definition at line **10** of file **diagonalscenario.h**.

The documentation for this class was generated from the following files:

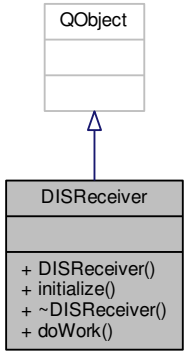
- PlatformsSimulator/diagonalscenario.h
- PlatformsSimulator/diagonalscenario.cpp

1.17 DISReceiver Class Reference

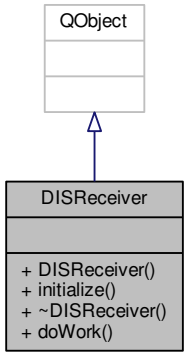
The **DISReceiver** (p. 32) class responsible of DIS traffic reception.

`#include <disreceiver.h>`

Inheritance diagram for DISReceiver:



Collaboration diagram for DISReceiver:



Public Slots

- void **doWork** ()

  doWork defines the mainloop of the worker thread

Signals

- void **pduReceived** (bool)

  pduReceived is emitted when a PDU is received

Public Member Functions

- **DISReceiver** (QString ip, int port)

  Class constructor to configure conection using ip and port.
- bool **initialize** (unsigned exercise_id)

  initialize Start the reception of the exercise_id
- ~**DISReceiver** ()

  Class destructor.

1.17.1 Detailed Description

The **DISReceiver** (p. 32) class responsible of DIS traffic reception.

Definition at line **18** of file **disreceiver.h**.

The documentation for this class was generated from the following files:

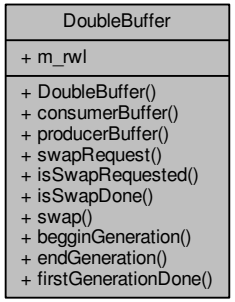
- libDISReceiver/disreceiver.h
- libDISReceiver/disreceiver.cpp

1.18 DoubleBuffer Class Reference

The **DoubleBuffer** (p. 34) class stores a two copies of ***Revolution*** *(p. 140)* to allow to the consumer buffer (sender) has a work copy of data meanwhile the producer (generator) generates new radius for the next radar cycle.

```
#include <doublebuffer.h>
```

Collaboration diagram for DoubleBuffer:



Public Member Functions

- **DoubleBuffer** (**Revolution** ∗read, **Revolution** ∗write)

  DoubleBuffer *(p. 34) constructor needs a read (consumer) and a write (producer) buffers.*
- **Revolution** ∗ **consumerBuffer** ()

  consumerBuffer returns the consumer buffers
- **Revolution** ∗ **producerBuffer** ()

  producerBuffer returns the producer buffers
- void **swapRequest** ()

  swapRequest mark that a swapping is required by producer or consumer
- bool **isSwapRequested** ()

  isSwapRequested allows to check if a swap is required
- bool **isSwapDone** ()

  isSwapDone allows to check if the swap is done
- void **swap** ()

  swap interchanges the producer and consumer buffers
- void **begginGeneration** ()

  begginGeneration mark the generation as started in the producer buffer
- void **endGeneration** ()

  begginGeneration mark the generation as done in the producer buffer
- bool **firstGenerationDone** ()

  firstGenerationDone allos to check if the first generation is done

Public Attributes

- QMutex **m_rwl**

1.18.1 Detailed Description

The **DoubleBuffer** (p. 34) class stores a two copies of ***Revolution*** *(p. 140)* to allow to the consumer buffer (sender) has a work copy of data meanwhile the producer (generator) generates new radius for the next radar cycle.

Definition at line **21** of file **doublebuffer.h**.

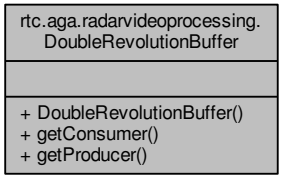
The documentation for this class was generated from the following files:

- RaViGEn/doublebuffer.h
- RaViGEn/doublebuffer.cpp

1.19 rtc.aga.radarvideoprocessing.DoubleRevolutionBuffer Class Reference

Specific implementation of producer - consumer problem to RaViGEn Test Console. It stores a producer and consumer copy but the consumer is always a copy of a producer data buffer. For this problem the most important thing is be able to access a copy of the most recent data to continue drawing the scene.

Collaboration diagram for rtc.aga.radarvideoprocessing.DoubleRevolutionBuffer:



Public Member Functions

- **DoubleRevolutionBuffer** (**Revolution** consumer, **Revolution** producer)

  Class constructor.
- synchronized **Revolution getConsumer** ()

  Get a copy of producer buffer in consumer data buffer.
- synchronized **Revolution getProducer** ()

1.19.1 Detailed Description

Specific implementation of producer - consumer problem to RaViGEn Test Console. It stores a producer and consumer copy but the consumer is always a copy of a producer data buffer. For this problem the most important thing is be able to access a copy of the most recent data to continue drawing the scene.

Definition at line **12** of file **DoubleRevolutionBuffer.java**.

1.19.2 Constructor & Destructor Documentation

1.19.2.1 rtc.aga.radarvideoprocessing.DoubleRevolutionBuffer.DoubleRevolutionBuffer (Revolution *consumer,* Revolution *producer*) [inline]

Class constructor.

Parameters

consumer	Consumer reference
producer	Producer reference

Definition at line **20** of file **DoubleRevolutionBuffer.java**.

1.19.3 Member Function Documentation

1.19.3.1 synchronized Revolution rtc.aga.radarvideoprocessing.DoubleRevolutionBuffer.getConsumer ( ) [inline]

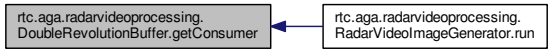
Get a copy of producer buffer in consumer data buffer.

Returns

Consumer buffer

Definition at line **30** of file **DoubleRevolutionBuffer.java**.

Here is the caller graph for this function:



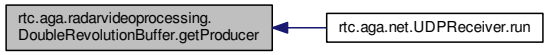
1.19.3.2 synchronized Revolution rtc.aga.radarvideoprocessing.DoubleRevolutionBuffer.getProducer ( ) [inline]

Returns

Reference to a producer buffer data

Definition at line **39** of file **DoubleRevolutionBuffer.java**.

Here is the caller graph for this function:



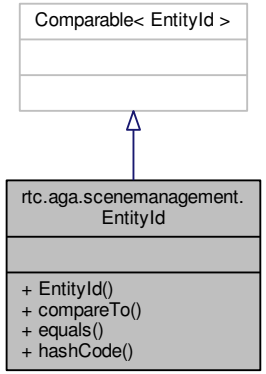
The documentation for this class was generated from the following file:

- RaViGEnTestConsole/RaViGEnTestConsole/src/rtc/aga/radarvideoprocessing/DoubleRevolution↩ Buffer.java

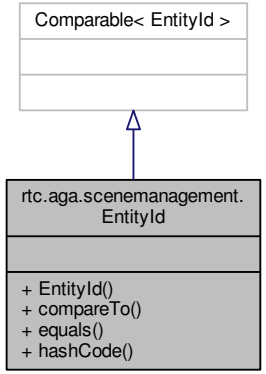
1.20 rtc.aga.scenemanagement.EntityId Class Reference

EntityId (p. 38) class implements a key value for java data Collections.

Inheritance diagram for rtc.aga.scenemanagement.EntityId:



Collaboration diagram for rtc.aga.scenemanagement.EntityId:



Public Member Functions

- **EntityId** (int site_id, int application_id, int entity_id)
  EntityId *(p. 38) class constructor.*
- int **compareTo** (**EntityId** another)
  Entity DIS simulation entity id.
- boolean **equals** (Object o)
  Comparator method.
- int **hashCode** ()
  Hash Code constructor based on an site and application id offset.

1.20.1 Detailed Description

EntityId (p. 38) class implements a key value for java data Collections.

Definition at line **6** of file **EntityId.java**.

1.20.2 Constructor & Destructor Documentation

1.20.2.1 rtc.aga.scenemanagement.EntityId.EntityId (int *site_id,* int *application_id,* int *entity_id*) [inline]

EntityId (p. 38) class constructor.

Parameters

site_id	
application_id	
entity_id	

Definition at line **14** of file **EntityId.java**.

Here is the caller graph for this function:

1.20.3 Member Function Documentation

1.20.3.1 int rtc.aga.scenemanagement.EntityId.compareTo (EntityId *another*) [inline]

Entity DIS simulation entity id.

Comparator method

Definition at line **29** of file **EntityId.java**.

The documentation for this class was generated from the following file:

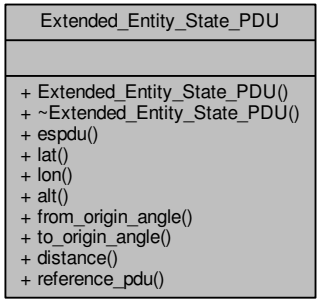
- RaViGEnTestConsole/RaViGEnTestConsole/src/rtc/aga/scenemanagement/EntityId.java

1.21 Extended_Entity_State_PDU Class Reference

The **Extended_Entity_State_PDU** (p. 40) class aggregate Entity State PDU with calculated parameters as latitude, longitude and altitude based on KDIS implementation.

`#include <extended_entity_state_pdu.h>`

Collaboration diagram for Extended_Entity_State_PDU:



Public Member Functions

- **Extended_Entity_State_PDU** (const KDIS::DATA_TYPE::EntityIdentifier &EI, KDIS::DATA_TYPE::ENUMS::ForceID ID, const KDIS::DATA_TYPE::EntityType &Type, const KDIS::DATA_TYPE::EntityType &AltType, const KDIS::DATA_TYPE::Vector &EntityLinearVelocity, const KDIS::DATA_TYPE::WorldCoordinates &EntityLocation, const KDIS::DATA_TYPE::EulerAngles &EntityOrientation, const KDIS::DATA_TYPE::EntityAppearance &EA, const KDIS::DATA_TYPE::DeadReckoningParameter &DRP, const KDIS::DATA_TYPE::EntityMarking &EM, const KDIS::DATA_TYPE::EntityCapabilities &EC, double **lat**, double **lon**, double **alt**, qlonglong insertion_time, bool **reference_pdu**, double **distance**=0.0, double **from_origin_angle**=0.0, double **to_origin_angle**=0.0)

  Extended_Entity_State_PDU *(p. 40) class constructor.*
- ∼**Extended_Entity_State_PDU** ()

  ∼*Extended_Entity_State_PDU class destructor*
- Entity_State_PDU ∗ **espdu** () const

  Retunr espdu base of Extended Entity.
- double **lat** () const

  lat returns latitude value of entity in degrees
- double **lon** () const

  lon returns longitude value of entity in degrees
- double **alt** () const

  alt returns altitude value of entity in meters
- double **from_origin_angle** () const

  from_origin_angle return the angle from reference entity to this platform in degrees
- double **to_origin_angle** () const

  from_origin_angle return the angle to reference entity from this platform in degrees
- double **distance** () const

  distance returns the distance to the reference platform in meters
- bool **reference_pdu** () const

  reference_pdu returns if this entity is the reference

1.21.1 Detailed Description

The **Extended_Entity_State_PDU** (p. 40) class aggregate Entity State PDU with calculated parameters as latitude, longitude and altitude based on KDIS implementation.

Definition at line **25** of file **extended_entity_state_pdu.h**.

The documentation for this class was generated from the following files:

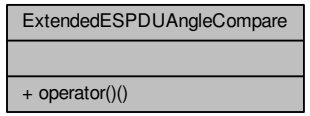
- libExtendedKDIS/extended_entity_state_pdu.h
- libExtendedKDIS/extended_entity_state_pdu.cpp

1.22 ExtendedESPDUAngleCompare Struct Reference

The **ExtendedESPDUAngleCompare** (p. 42) struct allows to Extended ESPDU be stored in ordered containers.

`#include <extended_entity_state_pdu.h>`

Collaboration diagram for ExtendedESPDUAngleCompare:



Public Member Functions

- bool **operator()** (const **Extended_Entity_State_PDU** ∗one, const **Extended_Entity_State_PDU** ∗another) const

1.22.1 Detailed Description

The **ExtendedESPDUAngleCompare** (p. 42) struct allows to Extended ESPDU be stored in ordered containers.

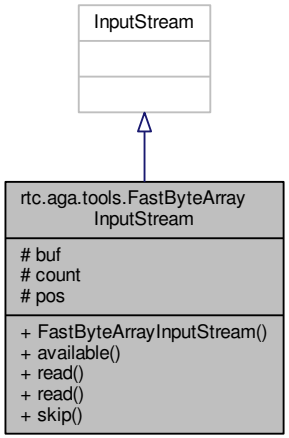
Definition at line **106** of file **extended_entity_state_pdu.h**.

The documentation for this struct was generated from the following file:

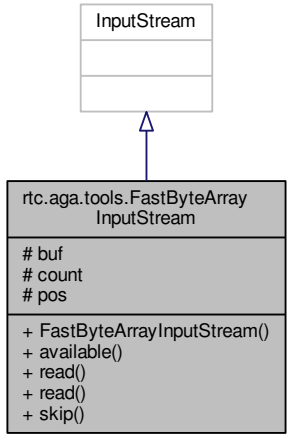
- libExtendedKDIS/extended_entity_state_pdu.h

1.23 rtc.aga.tools.FastByteArrayInputStream Class Reference

Inheritance diagram for rtc.aga.tools.FastByteArrayInputStream:



Collaboration diagram for rtc.aga.tools.FastByteArrayInputStream:



Public Member Functions

- **FastByteArrayInputStream** (byte[ ] **buf**, int **count**)
- final int **available** ()
- final int **read** ()
- final int **read** (byte[ ] b, int off, int len)
- final long **skip** (long n)

Protected Attributes

- byte[ ] **buf** = null
- int **count** = 0
- int **pos** = 0

1.23.1 Detailed Description

ByteArrayInputStream implementation that does not synchronize methods.

Definition at line **8** of file **FastByteArrayInputStream.java**.

1.23.2 Member Data Documentation

1.23.2.1 byte [ ] rtc.aga.tools.FastByteArrayInputStream.buf = null [protected]

Our byte buffer

Definition at line **13** of file **FastByteArrayInputStream.java**.

1.23.2.2 int rtc.aga.tools.FastByteArrayInputStream.count = 0 [protected]

Number of bytes that we can read from the buffer

Definition at line **18** of file **FastByteArrayInputStream.java**.

1.23.2.3 int rtc.aga.tools.FastByteArrayInputStream.pos = 0 [protected]

Number of bytes that have been read from the buffer

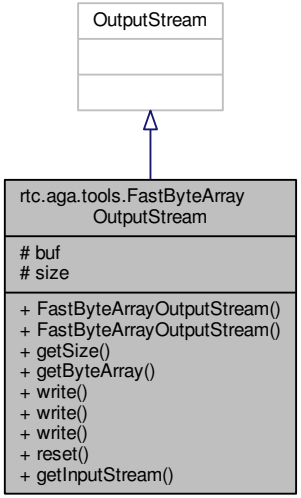
Definition at line **23** of file **FastByteArrayInputStream.java**.

The documentation for this class was generated from the following file:

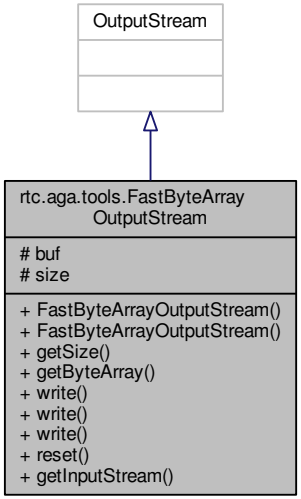
- RaViGEnTestConsole/RaViGEnTestConsole/src/rtc/aga/tools/FastByteArrayInputStream.java

1.24 rtc.aga.tools.FastByteArrayOutputStream Class Reference

Inheritance diagram for rtc.aga.tools.FastByteArrayOutputStream:



Collaboration diagram for rtc.aga.tools.FastByteArrayOutputStream:



Public Member Functions

- **FastByteArrayOutputStream** ()
- **FastByteArrayOutputStream** (int initSize)
- int **getSize** ()
- byte[ ] **getByteArray** ()
- final void **write** (byte b[ ])
- final void **write** (byte b[ ], int off, int len)
- final void **write** (int b)
- void **reset** ()
- InputStream **getInputStream** ()

Protected Attributes

- byte[ ] **buf** = null
- int **size** = 0

1.24.1 Detailed Description

ByteArrayOutputStream implementation that doesn't synchronize methods and doesn't copy the data on toByteArray().

Definition at line **10** of file **FastByteArrayOutputStream.java**.

1.24.2 Constructor & Destructor Documentation

1.24.2.1 rtc.aga.tools.FastByteArrayOutputStream.FastByteArrayOutputStream ( ) [inline]

Constructs a stream with buffer capacity size 5K

Definition at line **21** of file **FastByteArrayOutputStream.java**.

1.24.2.2 rtc.aga.tools.FastByteArrayOutputStream.FastByteArrayOutputStream (int *initSize*) [inline]

Constructs a stream with the given initial size

Definition at line **29** of file **FastByteArrayOutputStream.java**.

1.24.3 Member Function Documentation

1.24.3.1 byte [ ] rtc.aga.tools.FastByteArrayOutputStream.getByteArray ( ) [inline]

Returns the byte array containing the written data. Note that this array will almost always be larger than the amount of data actually written.

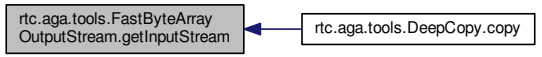
Definition at line **58** of file **FastByteArrayOutputStream.java**.

1.24.3.2 InputStream rtc.aga.tools.FastByteArrayOutputStream.getInputStream ( ) [inline]

Returns a ByteArrayInputStream for reading back the written data

Definition at line **91** of file **FastByteArrayOutputStream.java**.

Here is the caller graph for this function:



1.24.4 Member Data Documentation

1.24.4.1 byte [ ] rtc.aga.tools.FastByteArrayOutputStream.buf = null [protected]

Buffer and size

Definition at line **15** of file **FastByteArrayOutputStream.java**.

The documentation for this class was generated from the following file:

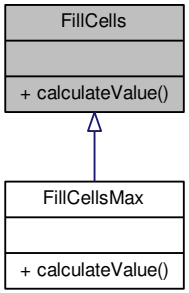
- RaViGEnTestConsole/RaViGEnTestConsole/src/rtc/aga/tools/FastByteArrayOutputStream.java

1.25 FillCells Class Reference

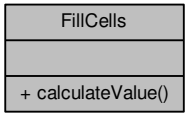
The **FillCells** (p. 49) interface is the base class to cell fillers.

`#include <fillcells.h>`

Inheritance diagram for FillCells:



Collaboration diagram for FillCells:



Public Member Functions

- virtual unsigned **calculateValue** (const QImage &image)=0
  calculateValue return a value for a cell applying a method that can be based on

1.25.1 Detailed Description

The **FillCells** (p. 49) interface is the base class to cell fillers.

Definition at line **10** of file **fillcells.h**.

1.25.2 Member Function Documentation

1.25.2.1 virtual unsigned FillCells::calculateValue (const QImage & *image*) [pure virtual]

calculateValue return a value for a cell applying a method that can be based on

Implemented in **FillCellsMax** (p. 52).

Here is the caller graph for this function:

The documentation for this class was generated from the following file:

- libModelStamper/fillcells.h

1.26 FillCellsFactory Class Reference

The **FillCellsFactory** (p. 50) class allows select a filler for every radar cell.

`#include <fillcellsfactory.h>`

Collaboration diagram for FillCellsFactory:

Static Public Member Functions

- static **FillCells** * **cellFiller** (QString filler)

  cellFiller selects a cells filler

1.26.1 Detailed Description

The **FillCellsFactory** (p. 50) class allows select a filler for every radar cell.

Definition at line **12** of file **fillcellsfactory.h**.

The documentation for this class was generated from the following files:

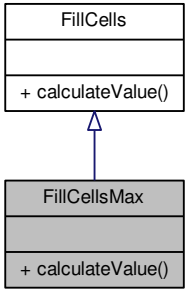
- libModelStamper/fillcellsfactory.h
- libModelStamper/fillcellsfactory.cpp

1.27 FillCellsMax Class Reference

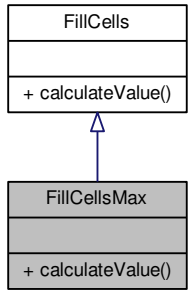
The **FillCellsMax** (p. 51) class implements a filler of cells based on max value of a bitmap represents by QImage.

```
#include <fillcellsmax.h>
```

Inheritance diagram for FillCellsMax:



Collaboration diagram for FillCellsMax:



Public Member Functions

- unsigned **calculateValue** (const QImage &image)

  calculateValue calculate the max value on image

1.27.1 Detailed Description

The **FillCellsMax** (p. 51) class implements a filler of cells based on max value of a bitmap represents by QImage.

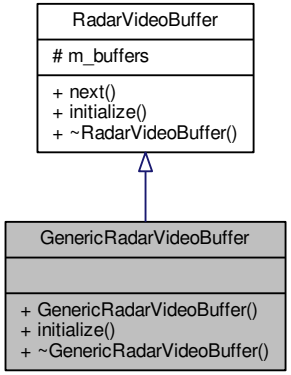
Definition at line **11** of file **fillcellsmax.h**.

The documentation for this class was generated from the following files:

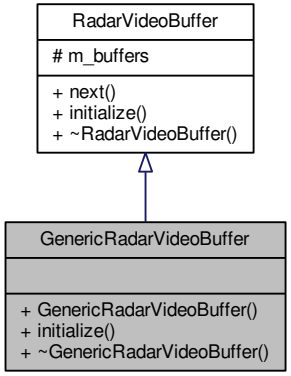
- libModelStamper/fillcellsmax.h
- libModelStamper/fillcellsmax.cpp

1.28 GenericRadarVideoBuffer Class Reference

Inheritance diagram for GenericRadarVideoBuffer:



Collaboration diagram for GenericRadarVideoBuffer:



Public Member Functions

- **GenericRadarVideoBuffer** ()

  GenericRadarVideoBuffer *(p. 53) Constructor.*
- void **initialize** (void *setup)

  initialize This method is used for the buffer properties initialization using ***GenericRadarVideoBuffer Settings*** *(p. 54) structure*
- ~**GenericRadarVideoBuffer** ()

Additional Inherited Members

1.28.1 Detailed Description

Definition at line **20** of file **genericradarvideobuffer.h**.

1.28.2 Constructor & Destructor Documentation

1.28.2.1 GenericRadarVideoBuffer::~GenericRadarVideoBuffer ( )

~GenericRadarVideoBuffer Desctructor

Definition at line **24** of file **genericradarvideobuffer.cpp**.

The documentation for this class was generated from the following files:

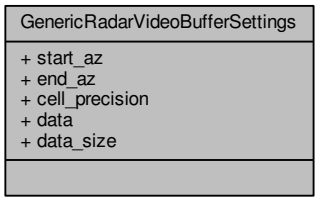
- libRadarVideoFormat/genericradarvideobuffer.h
- libRadarVideoFormat/genericradarvideobuffer.cpp

1.29 GenericRadarVideoBufferSettings Struct Reference

The **GenericRadarVideoBufferSettings** (p. 54) struct Properties definitifion of generic format video radar buffer.

`#include <genericradarvideobuffer.h>`

Collaboration diagram for GenericRadarVideoBufferSettings:



Public Attributes

- double **start_az**
- double **end_az**
  Initial angle in degrees.
- double **cell_precision**
  Final angle in degrees.
- unsigned * **data**
  Cell precision in meters.
- unsigned **data_size**
  Buffer data message.

1.29.1 Detailed Description

The **GenericRadarVideoBufferSettings** (p. 54) struct Properties definitifion of generic format video radar buffer.

Definition at line **10** of file **genericradarvideobuffer.h**.

The documentation for this struct was generated from the following file:

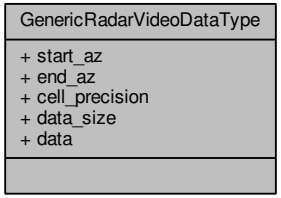
- libRadarVideoFormat/genericradarvideobuffer.h

1.30 GenericRadarVideoDataType Struct Reference

The **GenericRadarVideoDataType** (p. 55) struct stores experimental specification for radar message for developing and debugging.

```
#include <genericradarvideoformat.h>
```

Collaboration diagram for GenericRadarVideoDataType:



Public Attributes

- quint32 **start_az**
- quint32 **end_az**
  Start azimuth for the radar radius in degrees.
- quint32 **cell_precision**
  End azimuth for the radar radius in degrees.
- quint32 **data_size**
  Cell precision in meters.
- quint32 * **data**
  Data size of the radius in bytes.

1.30.1 Detailed Description

The **GenericRadarVideoDataType** (p. 55) struct stores experimental specification for radar message for developing and debugging.

Definition at line **12** of file **genericradarvideoformat.h**.

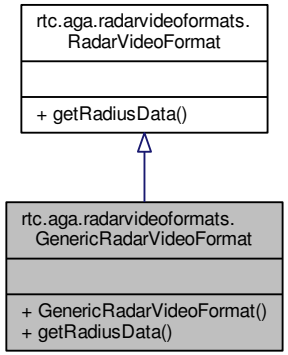
The documentation for this struct was generated from the following file:

- libRadarVideoFormat/genericradarvideoformat.h

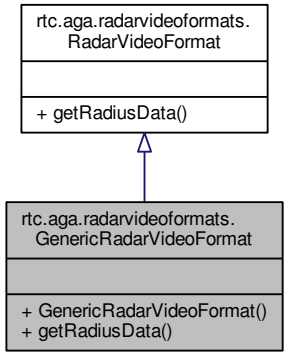
1.31 rtc.aga.radarvideoformats.GenericRadarVideoFormat Class Reference

GenericRadarVideoFormat (p. 56) implemts a generic radar video format to debug the RaViGEn project. It is based on **RadarVideoFormat** (p. 116) interface and it is in charge of deserialize the data message.

Inheritance diagram for rtc.aga.radarvideoformats.GenericRadarVideoFormat:



Collaboration diagram for rtc.aga.radarvideoformats.GenericRadarVideoFormat:



Public Member Functions

- **GenericRadarVideoFormat** (byte[ ] received_data)

  GenericRadarVideoFormat *(p. 56) recive the raw data of a radar video generic message deserializing it.*

- **GenericRadiusFormat getRadiusData** ()

  Returns the radius in generic format.

1.31.1 Detailed Description

GenericRadarVideoFormat (p. 56) implemts a generic radar video format to debug the RaViGEn project. It is based on **RadarVideoFormat** (p. 116) interface and it is in charge of deserialize the data message.

Definition at line **11** of file **GenericRadarVideoFormat.java**.

The documentation for this class was generated from the following file:

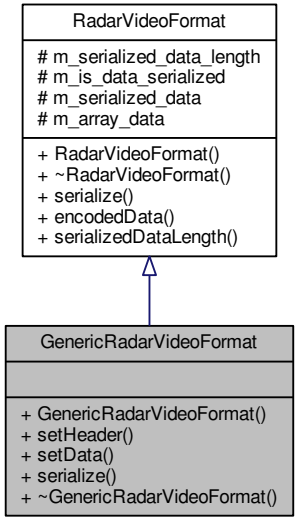
- RaViGEnTestConsole/RaViGEnTestConsole/src/rtc/aga/radarvideoformats/GenericRadarVideo↩ Format.java

1.32 GenericRadarVideoFormat Class Reference

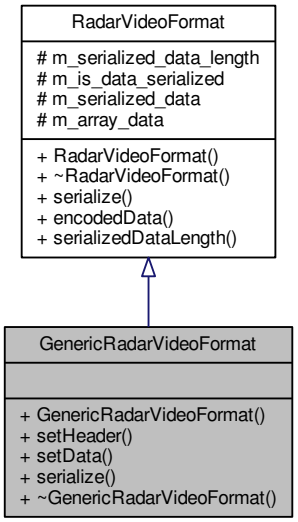
The **GenericRadarVideoFormat** (p. 58) class implemets version 0.1 of experimental common specification for radar video message developed for this project.

```
#include <genericradarvideoformat.h>
```

Inheritance diagram for GenericRadarVideoFormat:



Collaboration diagram for GenericRadarVideoFormat:



Public Member Functions

- **GenericRadarVideoFormat** ()
  GenericRadarVideoFormat *(p. 58) constructor.*
- void **setHeader** (double start_az, double end_az, double cell_precision)
  setHeader Describes the information included in the radar video message
- void **setData** (unsigned ∗data, unsigned data_size)
  setData Establish the data in the message
- void **serialize** ()
  serialize Marshall the data in buffer format for multicast
- ∼**GenericRadarVideoFormat** ()
  ∼GenericRadarVideoFormat Destructor

Additional Inherited Members

1.32.1 Detailed Description

The **GenericRadarVideoFormat** (p. 58) class implemets version 0.1 of experimental common specification for radar video message developed for this project.

Definition at line **27** of file **genericradarvideoformat.h**.

1.32.2 Member Function Documentation

1.32.2.1 void GenericRadarVideoFormat::setData (unsigned ∗ *data*, unsigned *data_size*)

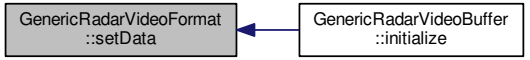
setData Establish the data in the message

Parameters

data	Buffer message
data_size	Size of buffer message

Definition at line **26** of file **genericradarvideoformat.cpp**.

Here is the caller graph for this function:



1.32.2.2 void GenericRadarVideoFormat::setHeader (double *start_az*, double *end_az*, double *cell_precision*)

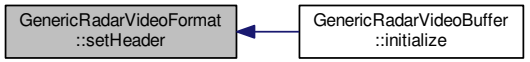
setHeader Describes the information included in the radar video message

Parameters

start_az	Initial angle in degrees
end_az	Final angle in degrees
cell_precision	Cell precision in meters

Definition at line **19** of file **genericradarvideoformat.cpp**.

Here is the caller graph for this function:



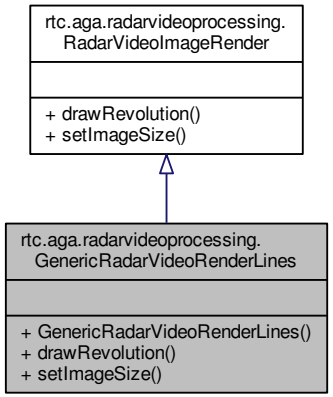
The documentation for this class was generated from the following files:

- libRadarVideoFormat/genericradarvideoformat.h
- libRadarVideoFormat/genericradarvideoformat.cpp

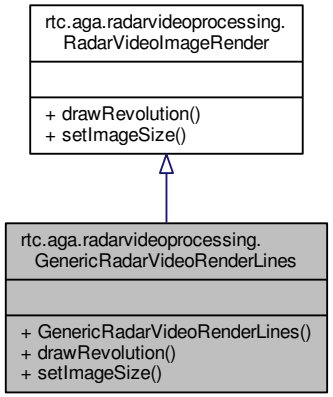
1.33 rtc.aga.radarvideoprocessing.GenericRadarVideoRenderLines Class Reference

GenericRadarVideoRenderLines (p. 61) implements the radar video render based on ray lines from the origin of the data to the end of the radar range.

Inheritance diagram for rtc.aga.radarvideoprocessing.GenericRadarVideoRenderLines:



Collaboration diagram for rtc.aga.radarvideoprocessing.GenericRadarVideoRenderLines:



Public Member Functions

- **GenericRadarVideoRenderLines** ()
  Class constructor.
- BufferedImage **drawRevolution** (**Revolution** rev)
  *Draw in BufferedImage the **Revolution** (p. 140) rev data.*
- void **setImageSize** (int imageSize)
  setImageSize establishes the height and width of the image

1.33.1 Detailed Description

GenericRadarVideoRenderLines (p. 61) implements the radar video render based on ray lines from the origin of the data to the end of the radar range.

Definition at line **15** of file **GenericRadarVideoRenderLines.java**.

1.33.2 Member Function Documentation

1.33.2.1 BufferedImage rtc.aga.radarvideoprocessing.GenericRadarVideoRenderLines.drawRevolution (Revolution *rev*) [inline]

Draw in *BufferedImage* the **Revolution** (p. 140) *rev* data.

Parameters

rev	*Radar* revolution

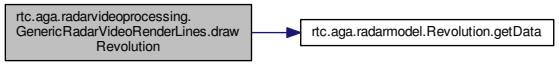
Returns

Buffered image based on a bitmap with the scene rendered

Implements **rtc.aga.radarvideoprocessing.RadarVideoImageRender** (p. 122).

Definition at line **30** of file **GenericRadarVideoRenderLines.java**.

Here is the call graph for this function:



1.33.2.2 void rtc.aga.radarvideoprocessing.GenericRadarVideoRenderLines.setImageSize (int *imageSize*) [inline]

setImageSize establishes the height and width of the image

Parameters

imageSize	Image height and widht in pixeles

Implements **rtc.aga.radarvideoprocessing.RadarVideoImageRender** (p. 122).

Definition at line **93** of file **GenericRadarVideoRenderLines.java**.

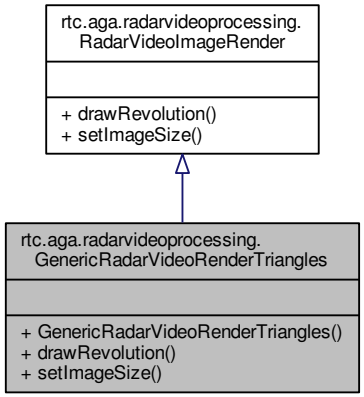
The documentation for this class was generated from the following file:

- RaViGEnTestConsole/RaViGEnTestConsole/src/rtc/aga/radarvideoprocessing/GenericRadar↩ VideoRenderLines.java

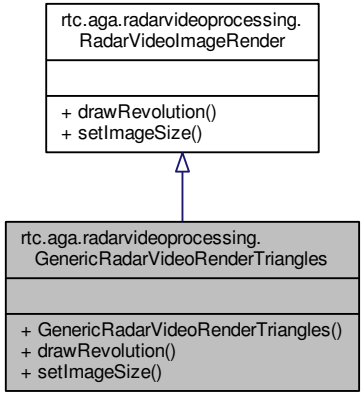
1.34 rtc.aga.radarvideoprocessing.GenericRadarVideoRenderTriangles Class Reference

GenericRadarVideoRenderTriangles (p. 63) implements the radar video render based on textured filled triangles built from the origin of the data to the end of the radar range.

Inheritance diagram for rtc.aga.radarvideoprocessing.GenericRadarVideoRenderTriangles:



Collaboration diagram for rtc.aga.radarvideoprocessing.GenericRadarVideoRenderTriangles:



Public Member Functions

- **GenericRadarVideoRenderTriangles** ()
  Class constructor.
- BufferedImage **drawRevolution** (**Revolution** rev)
  *Draw in BufferedImage the **Revolution** (p. 140) rev data.*
- void **setImageSize** (int imageSize)
  setImageSize establishes the height and width of the image

1.34.1 Detailed Description

GenericRadarVideoRenderTriangles (p. 63) implements the radar video render based on textured filled triangles built from the origin of the data to the end of the radar range.

Definition at line **19** of file **GenericRadarVideoRenderTriangles.java**.

1.34.2 Member Function Documentation

1.34.2.1 BufferedImage rtc.aga.radarvideoprocessing.GenericRadarVideoRenderTriangles.drawRevolution (Revolution *rev*) [inline]

Draw in *BufferedImage* the **Revolution** (p. 140) *rev* data.

Parameters

rev	*Radar* revolution

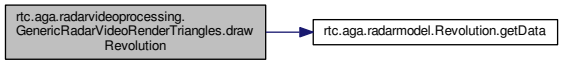
Returns

Buffered image based on a bitmap with the scene rendered

Implements **rtc.aga.radarvideoprocessing.RadarVideoImageRender** (p. 122).

Definition at line **34** of file **GenericRadarVideoRenderTriangles.java**.

Here is the call graph for this function:



1.34.2.2 void rtc.aga.radarvideoprocessing.GenericRadarVideoRenderTriangles.setImageSize (int *imageSize*) [inline]

setImageSize establishes the height and width of the image

Parameters

imageSize	Image height and widht in pixeles

Implements **rtc.aga.radarvideoprocessing.RadarVideoImageRender** (p. 122).

Definition at line **113** of file **GenericRadarVideoRenderTriangles.java**.

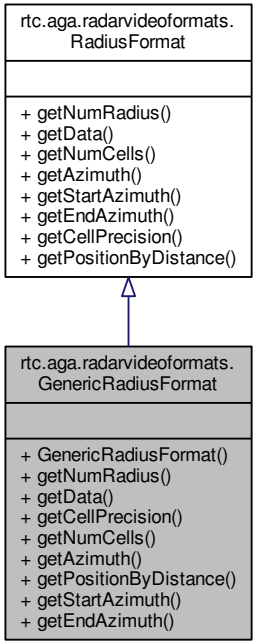
The documentation for this class was generated from the following file:

- RaViGEnTestConsole/RaViGEnTestConsole/src/rtc/aga/radarvideoprocessing/GenericRadar↩ VideoRenderTriangles.java

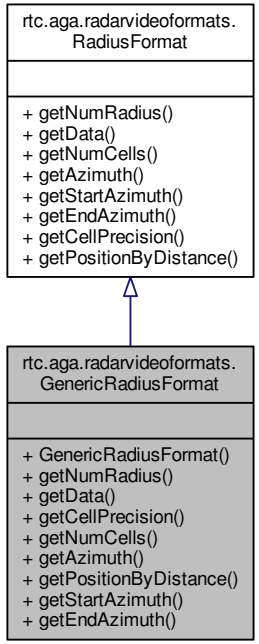
1.35 rtc.aga.radarvideoformats.GenericRadiusFormat Class Reference

Generic **Radius** (p. 126) format that are the base of radar ***Revolution*** *(p. 140)* and radar video format in its Generic implementation.

Inheritance diagram for rtc.aga.radarvideoformats.GenericRadiusFormat:



Collaboration diagram for rtc.aga.radarvideoformats.GenericRadiusFormat:



Public Member Functions

- **GenericRadiusFormat** (double startAz, double endAz, int cellPrecision, int numRadius, int data↩ Size, int[ ] data)

  Class constructor.
- int **getNumRadius** ()
- int[ ] **getData** ()
- int **getCellPrecision** ()
- int **getNumCells** ()
- double **getAzimuth** ()
- int **getPositionByDistance** (double distance, double range)

  Given a distance and a range scale return a position in the raw radius data.
- double **getStartAzimuth** ()
- double **getEndAzimuth** ()

1.35.1 Detailed Description

Generic **Radius** (p. 126) format that are the base of radar ***Revolution*** *(p. 140)* and radar video format in its Generic implementation.

Definition at line **7** of file **GenericRadiusFormat.java**.

1.35.2 Constructor & Destructor Documentation

1.35.2.1 rtc.aga.radarvideoformats.GenericRadiusFormat.GenericRadiusFormat (double *startAz,* double *endAz,* int *cellPrecision,* int *numRadius,* int *dataSize,* int[ ] *data*) [inline]

Class constructor.

Parameters

startAz	/// The start angle of the radius in degrees
endAz	/// The end angle of the radius in degrees
cellPrecision	/// The cell precision in meters
numRadius	/// Sequence number of the radius in a revolution
dataSize	/// The size of raw *data* in bytes
data	/// **Radius** (p. 126) data

Definition at line **19** of file **GenericRadiusFormat.java**.

1.35.3 Member Function Documentation

1.35.3.1 double rtc.aga.radarvideoformats.GenericRadiusFormat.getAzimuth ( ) [inline]

Returns

The middle angle of the radius in degrees

Implements **rtc.aga.radarvideoformats.RadiusFormat** (p. 129).

Definition at line **66** of file **GenericRadiusFormat.java**.

1.35.3.2 int rtc.aga.radarvideoformats.GenericRadiusFormat.getCellPrecision ( ) [inline]

Returns

The cell precision in meters

Implements **rtc.aga.radarvideoformats.RadiusFormat** (p. 130).

Definition at line **49** of file **GenericRadiusFormat.java**.

1.35.3.3 int [ ] rtc.aga.radarvideoformats.GenericRadiusFormat.getData ( ) [inline]

Returns

Returns the raw data of a radius

Implements **rtc.aga.radarvideoformats.RadiusFormat** (p. 130).

Definition at line **41** of file **GenericRadiusFormat.java**.

1.35.3.4 double rtc.aga.radarvideoformats.GenericRadiusFormat.getEndAzimuth ( ) [inline]

Returns

The end angle of the radius in degrees

Implements **rtc.aga.radarvideoformats.RadiusFormat** (p. 130).

Definition at line **97** of file **GenericRadiusFormat.java**.

1.35.3.5 int rtc.aga.radarvideoformats.GenericRadiusFormat.getNumCells ( ) [inline]

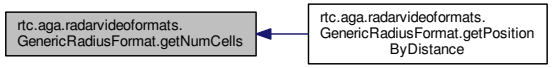
Returns

Number of cells in a radius

Implements **rtc.aga.radarvideoformats.RadiusFormat** (p. 130).

Definition at line **57** of file **GenericRadiusFormat.java**.

Here is the caller graph for this function:



1.35.3.6 int rtc.aga.radarvideoformats.GenericRadiusFormat.getNumRadius ( ) [inline]

Returns

Sequence number of the radius in a revolution

Implements **rtc.aga.radarvideoformats.RadiusFormat** (p. 130).

Definition at line **32** of file **GenericRadiusFormat.java**.

1.35.3.7 int rtc.aga.radarvideoformats.GenericRadiusFormat.getPositionByDistance (double *distance,* double *range*) [inline]

Given a *distance* and a *range* scale return a position in the raw radius data.

Parameters

distance	Distance to represent in meters
range	Scale of the distance

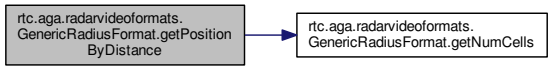
Returns

Position in the raw radius data

Implements **rtc.aga.radarvideoformats.RadiusFormat** (p. 130).

Definition at line **79** of file **GenericRadiusFormat.java**.

Here is the call graph for this function:



1.35.3.8 double rtc.aga.radarvideoformats.GenericRadiusFormat.getStartAzimuth ( ) [inline]

Returns

The start angle of the radius in degrees

Implements **rtc.aga.radarvideoformats.RadiusFormat** (p. 131).

Definition at line **88** of file **GenericRadiusFormat.java**.

The documentation for this class was generated from the following file:

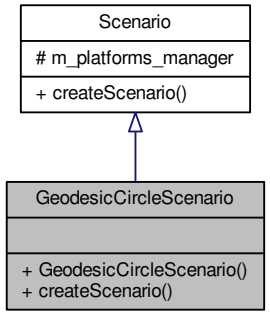
- RaViGEnTestConsole/RaViGEnTestConsole/src/rtc/aga/radarvideoformats/GenericRadius↩ Format.java

1.36 GeodesicCircleScenario Class Reference

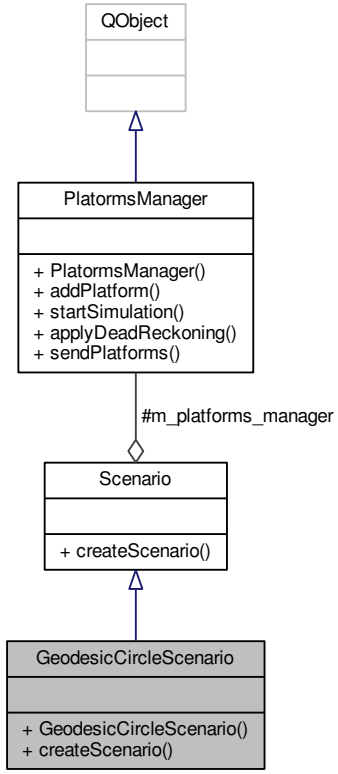
The **GeodesicCircleScenario** (p. 72) class implements a scenario that defines a circle around the reference platform.

`#include <geodesiccirclescenario.h>`

Inheritance diagram for GeodesicCircleScenario:



Collaboration diagram for GeodesicCircleScenario:



Public Member Functions

- **GeodesicCircleScenario** (**PlatormsManager** ∗manager)

  GeodesicCircleScenario *(p. 72) class method. Needs manager as paremeter to include the platforms in the scenario.*

- void **createScenario** (double range, unsigned num_platforms=10)

  createScenario creates a scenario based on range and num_platforms

Additional Inherited Members

1.36.1 Detailed Description

The **GeodesicCircleScenario** (p. 72) class implements a scenario that defines a circle around the reference platform.

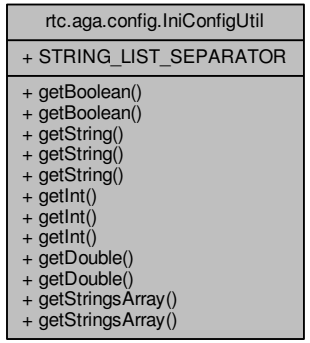
Definition at line **10** of file **geodesiccirclescenario.h**.

The documentation for this class was generated from the following files:

- PlatformsSimulator/geodesiccirclescenario.h
- PlatformsSimulator/geodesiccirclescenario.cpp

1.37 rtc.aga.config.IniConfigUtil Class Reference

Collaboration diagram for rtc.aga.config.IniConfigUtil:



Static Public Member Functions

- static boolean **getBoolean** (Ini.Section configSection, String propName, boolean defaultValue)
- static boolean **getBoolean** (Ini.Section configSection, String propName) throws Configuration↩ Exception
- static String **getString** (String filename, String sectionName, String propName, String defaultValue)
- static String **getString** (Ini.Section configSection, String propName, String defaultValue)

- static String **getString** (Ini.Section configSection, String propName) throws ConfigurationException
- static int **getInt** (Ini.Section configSection, String propName, int minValue, int maxValue) throws ConfigurationException
- static int **getInt** (String filename, String sectionName, String propName, int defaultValue, int min↩ Value, int maxValue)
- static int **getInt** (Ini.Section configSection, String propName, int defaultValue, int minValue, int maxValue)
- static double **getDouble** (String filename, String sectionName, String propName, double default↩ Value, double minValue, double maxValue)
- static double **getDouble** (Ini.Section configSection, String propName, double defaultValue, double minValue, double maxValue)
- static String[ ] **getStringsArray** (Ini.Section configSection, String propName) throws Configuration↩ Exception
- static String[ ] **getStringsArray** (Ini.Section configSection, String propName, String[ ] default↩ Values)

Static Public Attributes

- static final String **STRING_LIST_SEPARATOR** = " "

1.37.1 Detailed Description

Utilities for getting values for configuration files.

Definition at line **18** of file **IniConfigUtil.java**.

1.37.2 Member Function Documentation

1.37.2.1 static boolean rtc.aga.config.IniConfigUtil.getBoolean (Ini.Section *configSection,* String *propName,* boolean *defaultValue*) [inline],[static]

Extracts a boolean value from a configuration property. The values 'true', 'yes', and '1' are treated as true, the values 'false', 'no', '0' are treated as false, use case insensitive matching. If the value is anything else, or not present, the default value is used.

Parameters

configSection	configuration section from which to extract the attribute
propName	name of the configuration property
defaultValue	default value for the property

Returns

the value

Definition at line **39** of file **IniConfigUtil.java**.

Here is the call graph for this function:

1.37.2.2 static boolean rtc.aga.config.IniConfigUtil.getBoolean (Ini.Section *configSection,* String *propName*) throws ConfigurationException [inline],[static]

Extracts a boolean value from a configuration property. The values 'true', 'yes', and '1' are treated as true, the values 'false', 'no', '0' are treated as false, use case insensitive matching. If the value is anything else, or not present, the default value is used.

Parameters

configSection	configuration section from which to extract the attribute
propName	name of the configuration property

Returns

the value

Exceptions

ConfigurationException	thrown if given configuration section does not contain a property with the given name

Definition at line **73** of file **IniConfigUtil.java**.

Here is the call graph for this function:

1.37.2.3 static double rtc.aga.config.IniConfigUtil.getDouble (String *filename,* String *sectionName,* String *propName,* double *defaultValue,* double *minValue,* double *maxValue*) [inline],[static]

Extracts an double value from a configuration property given a FILENAME. If the property does not exist or has an invalid value the default value is returned.

Parameters

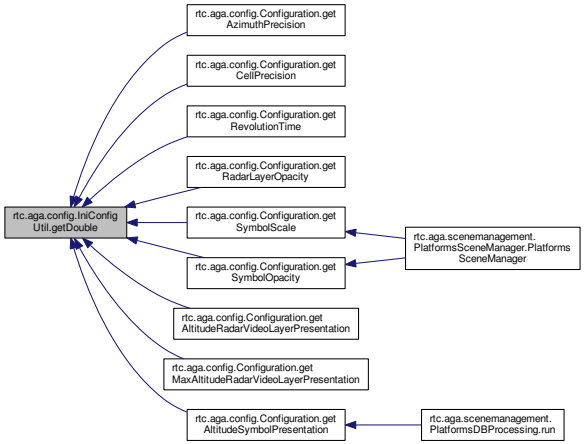
configSection	configuration section from which to extract the attribute
propName	name of the configuration property
defaultValue	default value of the property
minValue	minimum value of the property
maxValue	maximum value of the property

Returns

the value for the property (double)

Definition at line **343** of file **IniConfigUtil.java**.

Here is the caller graph for this function:



1.37.2.4 static double rtc.aga.config.IniConfigUtil.getDouble (Ini.Section *configSection,* String *propName,* double *defaultValue,* double *minValue,* double *maxValue*) [inline],[static]

Extracts an double value from a configuration property. If the property does not exist or has an invalid value the default value is returned.

Parameters

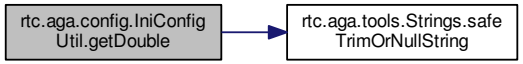
configSection	configuration section from which to extract the attribute
propName	name of the configuration property
defaultValue	default value of the property
minValue	minimum value of the property
maxValue	maximum value of the property

Returns

the value for the property

Definition at line **387** of file **IniConfigUtil.java**.

Here is the call graph for this function:



1.37.2.5 static int rtc.aga.config.IniConfigUtil.getInt (Ini.Section *configSection*, String *propName*, int *minValue*, int *maxValue*) throws ConfigurationException [inline], [static]

Extracts an integer value from a configuration property.

Parameters

configSection	configuration section from which to extract the attribute
propName	name of the configuration property
minValue	minimum value of the property
maxValue	maximum value of the property

Returns

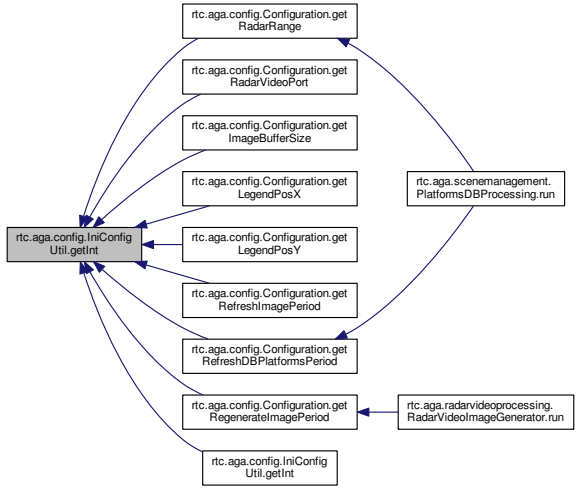
the value for the property

Exceptions

ConfigurationException	thrown if there is a problem getting the required integer value

Definition at line **210** of file **IniConfigUtil.java**.

Here is the caller graph for this function:



1.37.2.6 static int rtc.aga.config.IniConfigUtil.getInt (String *filename*, String *sectionName*, String *propName*, int *defaultValue*, int *minValue*, int *maxValue*) [inline], [static]

Extracts an integer value from a configuration property given a FILENAME. If the property does not exist or has an invalid value the default value is returned.

Parameters

configSection	configuration section from which to extract the attribute
propName	name of the configuration property
defaultValue	default value of the property
minValue	minimum value of the property
maxValue	maximum value of the property

Returns

the value for the property

Definition at line **254** of file **IniConfigUtil.java**.

Here is the call graph for this function:

1.37.2.7 static int rtc.aga.config.IniConfigUtil.getInt (Ini.Section *configSection*, String *propName*, int *defaultValue*, int *minValue*, int *maxValue*) [inline], [static]

Extracts an integer value from a configuration property. If the property does not exist or has an invalid value the default value is returned.

Parameters

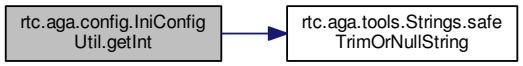
configSection	configuration section from which to extract the attribute
propName	name of the configuration property
defaultValue	default value of the property
minValue	minimum value of the property
maxValue	maximum value of the property

Returns

the value for the property

Definition at line **298** of file **IniConfigUtil.java**.

Here is the call graph for this function:



1.37.2.8 static String rtc.aga.config.IniConfigUtil.getString (String *filename*, String *sectionName*, String *propName*, String *defaultValue*) [inline], [static]

Extracts an String value from a configuration property given a FILENAME. If the property does not exist or has an invalid value the default value is returned.

Parameters

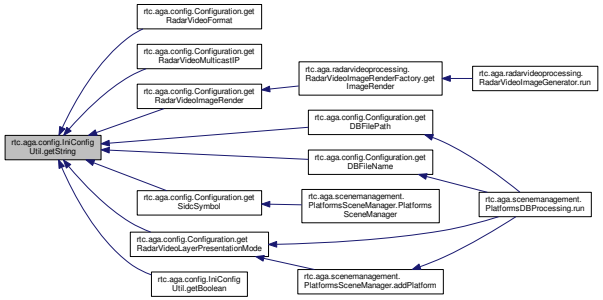
configSection	configuration section from which to extract the attribute
propName	name of the configuration property
defaultValue	default value of the property
minValue	minimum value of the property
maxValue	maximum value of the property

Returns

the value for the property (double)

Definition at line **110** of file **IniConfigUtil.java**.

Here is the caller graph for this function:



1.37.2.9 static String rtc.aga.config.IniConfigUtil.getString (Ini.Section *configSection*, String *propName*, String *defaultValue*) [inline], [static]

Extracts a string value from a configuration property. If the property does not exist or has a null or empty value the default value is used.

Parameters

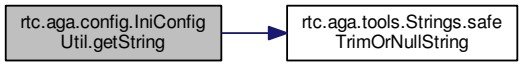
configSection	configuration section from which to extract the attribute
propName	name of the configuration property
defaultValue	default value for the property

Returns

the value of the property

Definition at line **152** of file **IniConfigUtil.java**.

Here is the call graph for this function:



1.37.2.10 static String rtc.aga.config.IniConfigUtil.getString (Ini.Section *configSection,* String *propName*) throws ConfigurationException [inline],[static]

Extracts a string value from a configuration property.

Parameters

configSection	configuration section from which to extract the attribute
propName	name of the configuration property

Returns

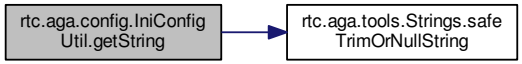
the value of the property

Exceptions

ConfigurationException	thrown if the value does not exist or has a null/empty value

Definition at line **179** of file **IniConfigUtil.java**.

Here is the call graph for this function:



1.37.2.11 static String [ ] rtc.aga.config.IniConfigUtil.getStringsArray (Ini.Section *configSection,* String *propName*) throws ConfigurationException [inline],[static]

Extracts a string list values from a configuration property, the values are separated with { **STRING_LIS↩ T_SEPARATOR** (p. 85)} (space).

Parameters

configSection	configuration section from which to extract the strings list
propName	name of the configuration property

Returns

the string values array of the property

Exceptions

ConfigurationException	thrown if the configuration property does not exist.

Definition at line **434** of file **IniConfigUtil.java**.

Here is the caller graph for this function:

1.37.2.12 static String [ ] rtc.aga.config.IniConfigUtil.getStringsArray (Ini.Section *configSection*, String *propName*, String[ ] *defaultValues*) [inline], [static]

Extracts a string list values from a configuration property, the values are separated with { **STRING_LIS↩ T_SEPARATOR** (p. 85)} (space).

Parameters

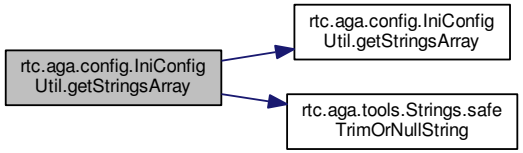
configSection	configuration section from which to extract the strings list
propName	name of the configuration property
defaultValues	the default list values to return if the configuration property does not exist.

Returns

the string values array of the property

Definition at line **452** of file **IniConfigUtil.java**.

Here is the call graph for this function:



1.37.3 Member Data Documentation

1.37.3.1 final String rtc.aga.config.IniConfigUtil.STRING_LIST_SEPARATOR = " " [static]

Separator for the strings list elements

Definition at line **418** of file **IniConfigUtil.java**.

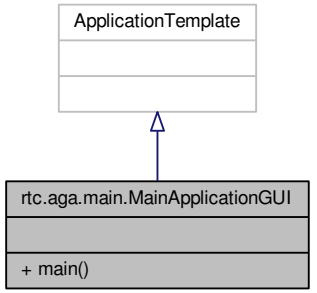
The documentation for this class was generated from the following file:

- RaViGEnTestConsole/RaViGEnTestConsole/src/rtc/aga/config/IniConfigUtil.java

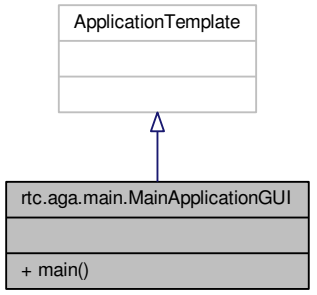
1.38 rtc.aga.main.MainApplicationGUI Class Reference

MainApplicationGUI (p. 86) iImplements a window responsible for displaying the video image radar geographic location. It has navigation tools, measurement and filter layers for debugging RaViGEn core.

Inheritance diagram for rtc.aga.main.MainApplicationGUI:



Collaboration diagram for rtc.aga.main.MainApplicationGUI:



Classes

- class **AppFrame**

Static Public Member Functions

- static void **main** (String[ ] args)

  Main method. Starts GUI application and establish the WorldWind network mode to off.

1.38.1 Detailed Description

MainApplicationGUI (p. 86) iImplements a window responsible for displaying the video image radar geographic location. It has navigation tools, measurement and filter layers for debugging RaViGEn core.

Definition at line **54** of file **MainApplicationGUI.java**.

The documentation for this class was generated from the following file:

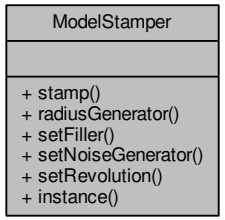
- RaViGEnTestConsole/RaViGEnTestConsole/src/rtc/aga/main/MainApplicationGUI.java

1.39 ModelStamper Class Reference

The **ModelStamper** (p. 87) class create the scene to be processed by radius generator to prepare the radius that will be sent.

```
#include <modelstamper.h>
```

Collaboration diagram for ModelStamper:



Public Member Functions

- void **stamp** (**Extended_Entity_State_PDU** ∗ees_pdu)

  stamp Includes the ees_pdu in the scene
- void **radiusGenerator** ()

  radiusGenerator creates the scene radius
- void **setFiller** (**FillCells** ∗filler)

  setFiller establishes the filler for each cell
- void **setNoiseGenerator** (**NoiseGenerator** ∗noiser)

  setNoiseGenerator establishes the noise generator noiser
- void **setRevolution** (**Revolution** ∗rev)

  setRevolution establishes the rev to proccess

Static Public Member Functions

- static **ModelStamper** ∗ **instance** ()

  instance Singleton implementation

1.39.1 Detailed Description

The **ModelStamper** (p. 87) class create the scene to be processed by radius generator to prepare the radius that will be sent.

Definition at line **21** of file **modelstamper.h**.

The documentation for this class was generated from the following files:

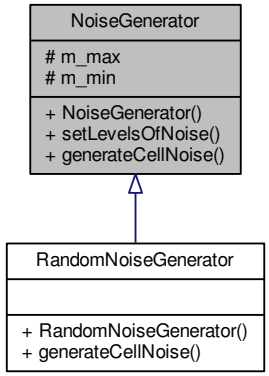
- libModelStamper/modelstamper.h
- libModelStamper/modelstamper.cpp

1.40 NoiseGenerator Class Reference

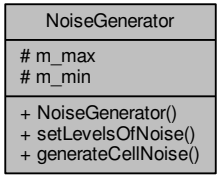
The **NoiseGenerator** (p. 88) abstract class is the base of noise generators.

```
#include <noisegenerator.h>
```

Inheritance diagram for NoiseGenerator:



Collaboration diagram for NoiseGenerator:



Public Member Functions

- **NoiseGenerator** ()

  NoiseGenerator *(p. 88) class constructor.*
- void **setLevelsOfNoise** (unsigned max, unsigned min)

  setLevelsOfNoise establish the limits of the generator usign max and min paremeters
- virtual unsigned **generateCellNoise** (unsigned pos, unsigned original_value)

  generateCellNoise in this implementation return original_value for each pos

Protected Attributes

- unsigned **m_max**
- unsigned **m_min**

1.40.1 Detailed Description

The **NoiseGenerator** (p. 88) abstract class is the base of noise generators.

Definition at line **10** of file **noisegenerator.h**.

The documentation for this class was generated from the following files:

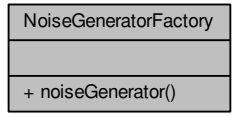
- libNoiseGenerator/noisegenerator.h
- libNoiseGenerator/noisegenerator.cpp

1.41 NoiseGeneratorFactory Class Reference

The **NoiseGeneratorFactory** (p. 90) class allows to RaViGEn choose a noise generator from a implemented ones.

`#include <noisegeneratorfactory.h>`

Collaboration diagram for NoiseGeneratorFactory:



Static Public Member Functions

- static **NoiseGenerator** ∗ **noiseGenerator** (QString generator)

  noiseGenerator returns a noise generator based on generator selection

1.41.1 Detailed Description

The **NoiseGeneratorFactory** (p. 90) class allows to RaViGEn choose a noise generator from a implemented ones.

Definition at line **14** of file **noisegeneratorfactory.h**.

The documentation for this class was generated from the following files:

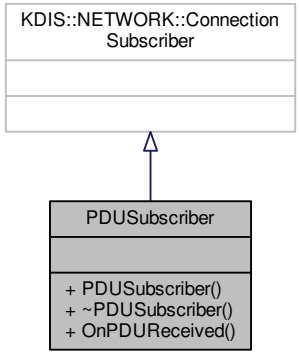
- libNoiseGenerator/noisegeneratorfactory.h
- libNoiseGenerator/noisegeneratorfactory.cpp

1.42 PDUSubscriber Class Reference

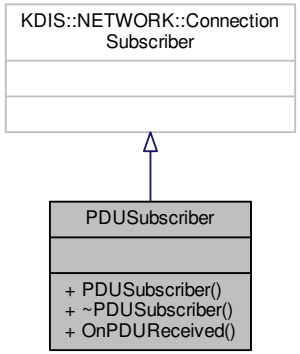
The **PDUSubscriber** (p. 91) class subscriber to PDUs DIS reception.

```
#include <pdusubscriber.h>
```

Inheritance diagram for PDUSubscriber:



Collaboration diagram for PDUSubscriber:



Public Member Functions

- **PDUSubscriber** ()

  PDUSubscriber *(p. 91) constructor.*
- ∼**PDUSubscriber** ()

  PDUSubscriber *(p. 91) destructor.*
- virtual void **OnPDUReceived** (const KDIS::PDU::Header ∗header)

  OnPDUReceived overloaded PDU Received function.

1.42.1 Detailed Description

The **PDUSubscriber** (p. 91) class subscriber to PDUs DIS reception.

Definition at line **14** of file **pdusubscriber.h**.

The documentation for this class was generated from the following files:

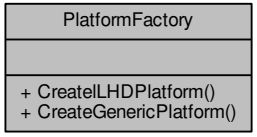
- libDISReceiver/pdusubscriber.h
- libDISReceiver/pdusubscriber.cpp

1.43 PlatformFactory Class Reference

The **PlatformFactory** (p. 93) class creates the platforms used in the scenarios.

`#include <platformfactory.h>`

Collaboration diagram for PlatformFactory:



Static Public Member Functions

- static Entity_State_PDU * **CreatelLHDPlatform** (double lat, double lon, double alt)

  CreatelLHDPlatform creates a LHD JC1 platform in a lat, lon and lat given position.
- static Entity_State_PDU * **CreateGenericPlatform** (double lat, double lon, double alt)

  CreateGenericPlatform creates a Generic platform in a lat, lon and lat given position.

1.43.1 Detailed Description

The **PlatformFactory** (p. 93) class creates the platforms used in the scenarios.

Definition at line **20** of file **platformfactory.h**.

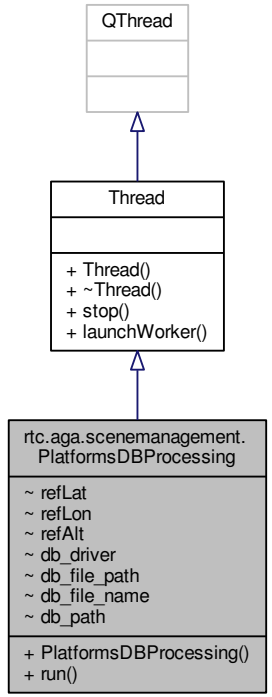
The documentation for this class was generated from the following files:

- PlatformsSimulator/platformfactory.h
- PlatformsSimulator/platformfactory.cpp

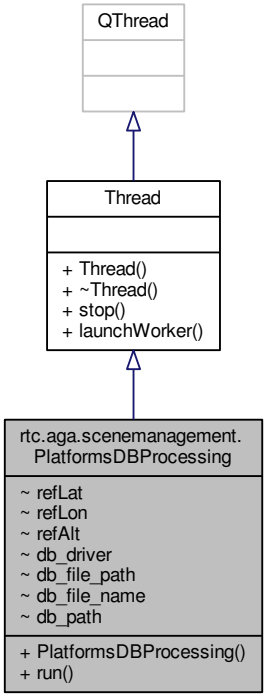
1.44 rtc.aga.scenemanagement.PlatformsDBProcessing Class Reference

PlatformsDBProcessing (p. 94) is the worker thread and it is responsible for updating the platforms read from database and drawing on the layer.

Inheritance diagram for rtc.aga.scenemanagement.PlatformsDBProcessing:



Collaboration diagram for rtc.aga.scenemanagement.PlatformsDBProcessing:



Public Member Functions

- **PlatformsDBProcessing** (RenderableLayer platformsDBLayer, AnalyticSurface radarVideo↩ Surface, Globe globe)

  Class constructor.
- void **run** ()

  Main loop of the worker thread.

Additional Inherited Members

1.44.1 Detailed Description

PlatformsDBProcessing (p. 94) is the worker thread and it is responsible for updating the platforms read from database and drawing on the layer.

Definition at line **25** of file **PlatformsDBProcessing.java**.

1.44.2 Constructor & Destructor Documentation

1.44.2.1 rtc.aga.scenemanagement.PlatformsDBProcessing.PlatformsDBProcessing (RenderableLayer *platformsDBLayer*, AnalyticSurface *radarVideoSurface*, Globe *globe*) [inline]

Class constructor.

Parameters

platformsDBLayer	Layer to add and draw the symbol platforms
radarVideoSurface	is necessary to produce a regeneration of the scene
globe	for geodesic calculations

Definition at line **34** of file **PlatformsDBProcessing.java**.

1.44.3 Member Function Documentation

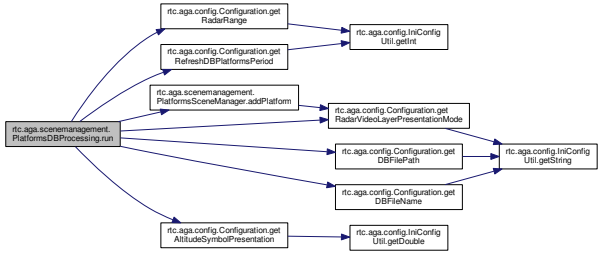
1.44.3.1 void rtc.aga.scenemanagement.PlatformsDBProcessing.run ( ) [inline]

Main loop of the worker thread.

- Read the platforms data base
- Generates the platforms symbols

Definition at line **53** of file **PlatformsDBProcessing.java**.

Here is the call graph for this function:



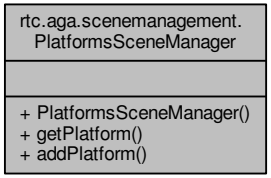
The documentation for this class was generated from the following file:

- RaViGEnTestConsole/RaViGEnTestConsole/src/rtc/aga/scenemanagement/PlatformsDBProcessing.↩ java

1.45 rtc.aga.scenemanagement.PlatformsSceneManager Class Reference

PlatformsSceneManager (p. 97) is the responsible of platforms managing:

Collaboration diagram for rtc.aga.scenemanagement.PlatformsSceneManager:



Public Member Functions

- **PlatformsSceneManager** ()
  Class constructor.
- TacticalSymbol **getPlatform** (int site_id, int application_id, int entity_id)
  Given a Entity ID returns a Symbol for the platform.
- TacticalSymbol **addPlatform** (**EntityId** id, int specificType, Position pos, RenderableLayer platformsDBLayer)
  addPlatform adds a new symbol to the platforms layer

1.45.1 Detailed Description

PlatformsSceneManager (p. 97) is the responsible of platforms managing:

- Adds new platforms
- Creates symbols
- Sets the 2D or 3D mode

Definition at line **23** of file **PlatformsSceneManager.java**.

1.45.2 Member Function Documentation

1.45.2.1 TacticalSymbol rtc.aga.scenemanagement.PlatformsSceneManager.addPlatform (EntityId *id*, int *specificType*, Position *pos*, RenderableLayer *platformsDBLayer*) [inline]

addPlatform adds a new symbol to the platforms layer

Parameters

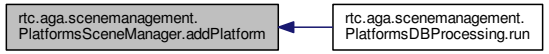
id	Identification of the platform
specificType	DIS type of the platform
pos	Latitude, Longitude and Altitude position of the symbol
platformsDBLayer	Layer of the platforms

Returns

Definition at line **59** of file **PlatformsSceneManager.java**.

Here is the call graph for this function:

Here is the caller graph for this function:



1.45.2.2 TacticalSymbol rtc.aga.scenemanagement.PlatformsSceneManager.getPlatform (int *site_id*, int *application_id*, int *entity_id*) [inline]

Given a Entity ID returns a Symbol for the platform.

Parameters

site_id	Entity ID specification
application_id	Entity ID specification
entity_id	Entity ID specification

Returns

Symbol for the platform

Definition at line **43** of file **PlatformsSceneManager.java**.

The documentation for this class was generated from the following file:

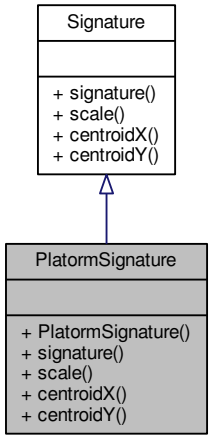
- RaViGEnTestConsole/RaViGEnTestConsole/src/rtc/aga/scenemanagement/PlatformsSceneManager.java

1.46 PlatormSignature Class Reference

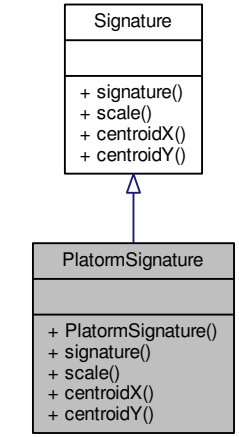
The PlatormsSignature class load an store the bitmap representation of the signal radar of a platform.

`#include <platormsignature.h>`

Inheritance diagram for PlatormSignature:



Collaboration diagram for PlatormSignature:



Public Member Functions

- **PlatormSignature** (const KDIS::DATA_TYPE::EntityType &entity_type)

  PlatormsSignature constructor initialize structs and data for the /a entity_type platform.
- QImage * **signature** () const

  signature() (p. *100*) *return the bitmap representation of radarsignature*
- double **scale** () const

  scale() (p. *100*) *return the Scale of the bitmap representation*
- int **centroidX** () const

  centroidX return the x center coordinate of the bitmap representation
- int **centroidY** () const

  centroidY return the y center coordinate of the bitmap representation

1.46.1 Detailed Description

The PlatormsSignature class load an store the bitmap representation of the signal radar of a platform.

Definition at line **20** of file **platormsignature.h**.

1.46.2 Member Function Documentation

1.46.2.1 int PlatormSignature::centroidY ( ) const [virtual]

centroidY return the y center coordinate of the bitmap representation

Returns

Implements **Signature** (p. 158).

Definition at line **59** of file **platormsignature.cpp**.

The documentation for this class was generated from the following files:

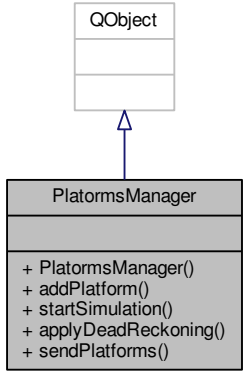
- libRadarSignature/platormsignature.h
- libRadarSignature/platormsignature.cpp

1.47 PlatormsManager Class Reference

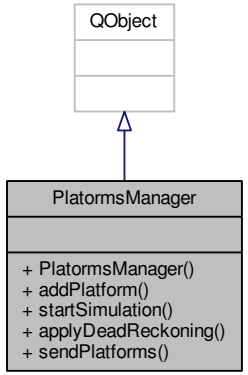
The **PlatormsManager** (p. 101) class manages, updates, and stores the simulated scenario.

`#include <platormsmanager.h>`

Inheritance diagram for PlatormsManager:



Collaboration diagram for PlatormsManager:



Public Slots

- void **startSimulation** ()
  start timers to begin with the simulation
- void **applyDeadReckoning** ()
  applyDeadReckoning applies periodically dead reckoning algorithm
- void **sendPlatforms** ()
  sendPlatforms send Entity State PDU periodically by multicast to keep live the simulation

Public Member Functions

- **PlatormsManager** ()
  PlatormsManager (p. *101) class constructor initializes timer values and creates structures.*
- void **addPlatform** (**SurfaceVehicle** ∗espdu)
  addPlatform adds a new espud platform to the scenario simulation

1.47.1 Detailed Description

The **PlatormsManager** (p. 101) class manages, updates, and stores the simulated scenario.

Definition at line **29** of file **platormsmanager.h**.

The documentation for this class was generated from the following files:

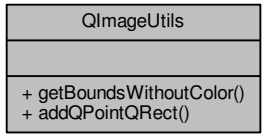
- PlatformsSimulator/platormsmanager.h
- PlatformsSimulator/platormsmanager.cpp

1.48 QImageUtils Class Reference

The **QImageUtils** (p. 103) class helps to model stamper in the operations with QImage Qt class.

`#include <qimageutils.h>`

Collaboration diagram for QImageUtils:



Static Public Member Functions

- static QRect **getBoundsWithoutColor** (const QImage &image, const QColor &exclusion_↩ color=QColor(Qt::transparent))

  getBoundsWithoutColor returns a bounding box adjusted to
- static QRect **addQPointQRect** (QPoint p, QRect r)

  addQPointQRect returns the QRect sum between p and r

1.48.1 Detailed Description

The **QImageUtils** (p. 103) class helps to model stamper in the operations with QImage Qt class.

Definition at line **12** of file **qimageutils.h**.

1.48.2 Member Function Documentation

1.48.2.1 QRect QImageUtils::getBoundsWithoutColor (const QImage & *image,* const QColor & *exclusion_color =* `QColor(Qt::transparent)`) [static]

getBoundsWithoutColor returns a bounding box adjusted to

bieng *exclusion* color the limit of this bounding box

Definition at line **3** of file **qimageutils.cpp**.

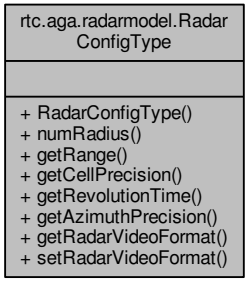
Here is the caller graph for this function:

The documentation for this class was generated from the following files:

- libModelStamper/qimageutils.h
- libModelStamper/qimageutils.cpp

1.49 rtc.aga.radarmodel.RadarConfigType Class Reference

Collaboration diagram for rtc.aga.radarmodel.RadarConfigType:



Public Member Functions

- **RadarConfigType** (int range, double azimuthPrecision, double cellPrecision, double revolution↩ Time, String radarVideoFormat)

  RadarConfigType *(p. 104) class constructor.*
- int **numRadius** ()
- int **getRange** ()
- double **getCellPrecision** ()
- double **getRevolutionTime** ()
- double **getAzimuthPrecision** ()
- String **getRadarVideoFormat** ()
- void **setRadarVideoFormat** (String radarVideoFormat)

1.49.1 Detailed Description

RadarConfigType (p. 104) stores the radar configuration:

- Range radar in meters
- Cell Precision in meters
- Azimuth precision in meters
- Radar Video Format
- Number of the radius in a cycle

Definition at line **11** of file **RadarConfigType.java**.

1.49.2 Constructor & Destructor Documentation

1.49.2.1 rtc.aga.radarmodel.RadarConfigType.RadarConfigType (int *range*, double *azimuthPrecision*, double *cellPrecision*, double *revolutionTime*, String *radarVideoFormat*) [inline]

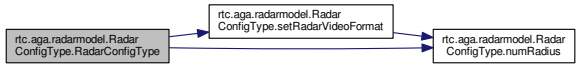
RadarConfigType (p. 104) class constructor.

Parameters

range	Radar range in meters
azimuthPrecision	Azimuth precision in degrees
cellPrecision	Cell precision in meters
revolutionTime	Time of a complete cycle radar
radarVideoFormat	Radar video format

Definition at line **22** of file **RadarConfigType.java**.

Here is the call graph for this function:



1.49.3 Member Function Documentation

1.49.3.1 double rtc.aga.radarmodel.RadarConfigType.getAzimuthPrecision ( ) [inline]

Returns

Azimuth precision of the radar in degrees

Definition at line **69** of file **RadarConfigType.java**.

1.49.3.2 double rtc.aga.radarmodel.RadarConfigType.getCellPrecision ( ) [inline]

Returns

Cell precision in meters

Definition at line **53** of file **RadarConfigType.java**.

1.49.3.3 String rtc.aga.radarmodel.RadarConfigType.getRadarVideoFormat ( ) [inline]

Returns

Radar video format

Definition at line **77** of file **RadarConfigType.java**.

1.49.3.4 int rtc.aga.radarmodel.RadarConfigType.getRange ( ) [inline]

Returns

Radar range in meters

Definition at line **45** of file **RadarConfigType.java**.

1.49.3.5 double rtc.aga.radarmodel.RadarConfigType.getRevolutionTime ( ) [inline]

Returns

Radar revolution time in milliseconds

Definition at line **61** of file **RadarConfigType.java**.

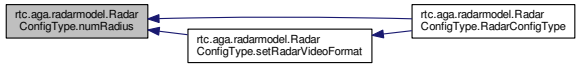
1.49.3.6 int rtc.aga.radarmodel.RadarConfigType.numRadius ( ) [inline]

Returns

Number of radius in a cycle radar

Definition at line **37** of file **RadarConfigType.java**.

Here is the caller graph for this function:



1.49.3.7 void rtc.aga.radarmodel.RadarConfigType.setRadarVideoFormat (String *radarVideoFormat*) [inline]

Establish the radar video format

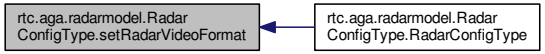
Parameters

radarVideoFormat	

Definition at line **86** of file **RadarConfigType.java**.

Here is the call graph for this function:

Here is the caller graph for this function:



The documentation for this class was generated from the following file:

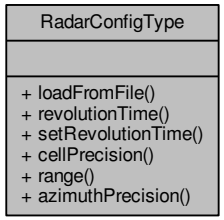
- RaViGEnTestConsole/RaViGEnTestConsole/src/rtc/aga/radarmodel/RadarConfigType.java

1.50 RadarConfigType Class Reference

The **RadarConfigType** (p. 108) class loads and stores the radar configuration.

`#include <radarconfig.h>`

Collaboration diagram for RadarConfigType:



Public Member Functions

- void **loadFromFile** ()

  loadFromFile Load from configuration file the parametres of radar
- double **revolutionTime** () const

  revolutionTime query method in ms
- void **setRevolutionTime** (double value)

setRevolutionTime modification method in ms

- double **cellPrecision** () const

  cellPrecision returns cell precision in meters

- unsigned int **range** () const

  range returns radar range in meters

- double **azimuthPrecision** () const

  azimuthPrecision returns azimuth precision in degress

1.50.1 Detailed Description

The **RadarConfigType** (p. 108) class loads and stores the radar configuration.

Definition at line **11** of file **radarconfig.h**.

The documentation for this class was generated from the following files:

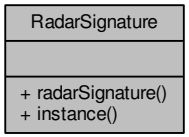
- libRadarModel/radarconfig.h
- libRadarModel/radarconfig.cpp

1.51 RadarSignature Class Reference

The **RadarSignature** (p. 109) singleton store and load the bitmaps that represents the radar signature of the every DIS entity.

`#include <radarsignature.h>`

Collaboration diagram for RadarSignature:



Public Member Functions

- **Signature** * **radarSignature** (const KDIS::DATA_TYPE::EntityType &entity_type)

  radarSignature returns bitmap radar signature of any DIS platform DIS pass by /a entity_type parameter

Static Public Member Functions

- static **RadarSignature** * **instance** ()

  *instance Implements Radar **Signature** (p. 157) singleton*

1.51.1 Detailed Description

The **RadarSignature** (p. 109) singleton store and load the bitmaps that represents the radar signature of the every DIS entity.

Definition at line **22** of file **radarsignature.h**.

1.51.2 Member Function Documentation

1.51.2.1 RadarSignature * RadarSignature::instance ( ) [static]

instance Implements Radar **Signature** (p. 157) singleton

Returns

Pointer to Radar **Signature** (p. 157) instance

Definition at line **15** of file **radarsignature.cpp**.

The documentation for this class was generated from the following files:

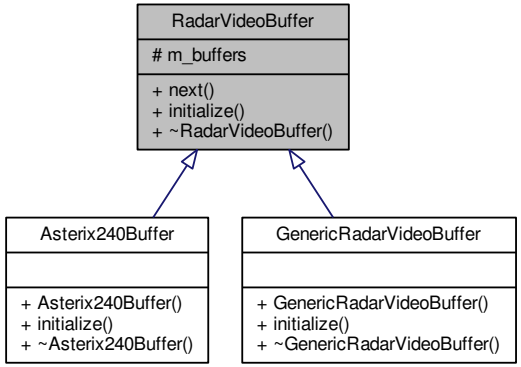
- libRadarSignature/radarsignature.h
- libRadarSignature/radarsignature.cpp

1.52 RadarVideoBuffer Class Reference

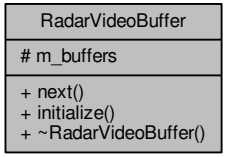
RadarVideoBuffer (p. 110) class represent the set of Radar Video Format messages that composes a revolution. Usually more than one messages is need for each radius.

`#include <radarvideobuffer.h>`

Inheritance diagram for RadarVideoBuffer:



Collaboration diagram for RadarVideoBuffer:



Public Member Functions

- virtual **RadarVideoFormat** ∗ **next** ()

  Implements next function in iterator pattern.
- virtual void **initialize** (void ∗setup)=0

  initialize This method is used for the buffer properties initialization
- virtual ∼**RadarVideoBuffer** ()

  ∼RadarVideoBuffer Virtual destructor

Protected Attributes

- std::queue< **RadarVideoFormat** ∗ > **m_buffers**

1.52.1 Detailed Description

RadarVideoBuffer (p. 110) class represent the set of Radar Video Format messages that composes a revolution. Usually more than one messages is need for each radius.

Definition at line **15** of file **radarvideobuffer.h**.

1.52.2 Member Function Documentation

1.52.2.1 virtual void RadarVideoBuffer::initialize (void ∗ *setup*) [pure virtual]

initialize This method is used for the buffer properties initialization

Parameters

setup	Initilization params

Implemented in **Asterix240Buffer** (p. 2), and **GenericRadarVideoBuffer** (p. 54).

Here is the caller graph for this function:

1.52.2.2 RadarVideoFormat ∗ RadarVideoBuffer::next ( ) [virtual]

Implements next function in iterator pattern.

Returns

Next Radar Video Format message

Definition at line **3** of file **radarvideobuffer.cpp**.

Here is the caller graph for this function:

The documentation for this class was generated from the following files:

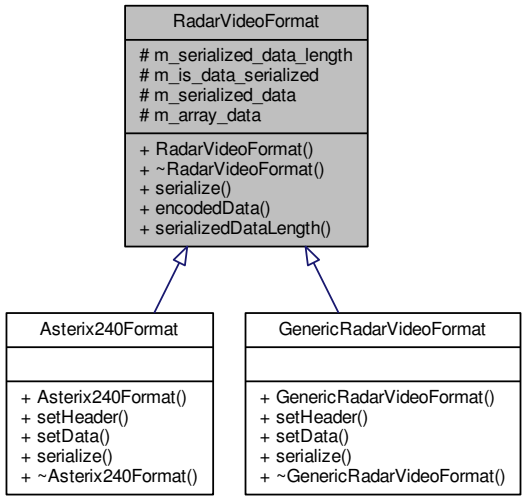
- libRadarVideoFormat/radarvideobuffer.h
- libRadarVideoFormat/radarvideobuffer.cpp

1.53 RadarVideoFormat Class Reference

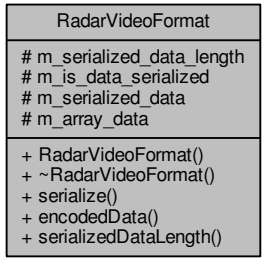
The **RadarVideoFormat** (p. 113) Abstract class interface used to generalizes radar video formats for data sender.

`#include <radarvideoformat.h>`

Inheritance diagram for RadarVideoFormat:



Collaboration diagram for RadarVideoFormat:



Public Member Functions

- **RadarVideoFormat** ()

RadarVideoFormat *(p. 113) initializes control data structures.*

- virtual ∼**RadarVideoFormat** ()

  ∼*RadarVideoFormat Destructor*
- virtual void **serialize** ()=0

  serialize Pure virtual function to be implemented by inherited classes
- virtual char ∗ **encodedData** ()

  encodedData Method to get the serialized data
- virtual unsigned **serializedDataLength** () const

  serializedDataLength

Protected Attributes

- unsigned **m_serialized_data_length**
- bool **m_is_data_serialized**

  Length of data serialized.
- QDataStream ∗ **m_serialized_data**

  Flag to know if data is serialized.
- QByteArray **m_array_data**

  Var to store data serialized.

1.53.1 Detailed Description

The **RadarVideoFormat** (p. 113) Abstract class interface used to generalizes radar video formats for data sender.

Definition at line **12** of file **radarvideoformat.h**.

1.53.2 Member Function Documentation

1.53.2.1 char ∗ RadarVideoFormat::encodedData ( ) [virtual]

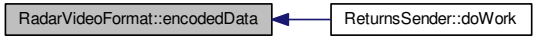
encodedData Method to get the serialized data

Returns

Buffer data to be sended by multicast

Definition at line **19** of file **radarvideoformat.cpp**.

Here is the caller graph for this function:



1.53.2.2 unsigned RadarVideoFormat::serializedDataLength ( ) const [virtual]

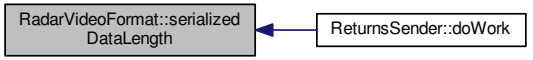
serializedDataLength

Returns

The lenght in bytes of data serialized

Definition at line **29** of file **radarvideoformat.cpp**.

Here is the caller graph for this function:



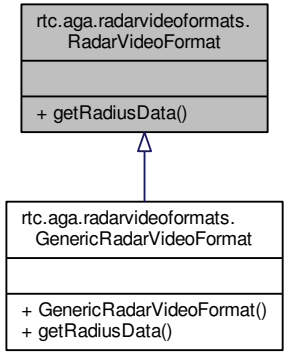
The documentation for this class was generated from the following files:

- libRadarVideoFormat/radarvideoformat.h
- libRadarVideoFormat/radarvideoformat.cpp

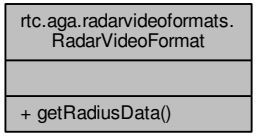
1.54 rtc.aga.radarvideoformats.RadarVideoFormat Interface Reference

RadiusVideoFormat interface allows to RaViGEn Test Console generalize the radar video message decoding.

Inheritance diagram for rtc.aga.radarvideoformats.RadarVideoFormat:



Collaboration diagram for rtc.aga.radarvideoformats.RadarVideoFormat:



Public Member Functions

- **RadiusFormat getRadiusData** ()

1.54.1 Detailed Description

RadiusVideoFormat interface allows to RaViGEn Test Console generalize the radar video message decoding.

Definition at line **7** of file **RadarVideoFormat.java**.

1.54.2 Member Function Documentation

1.54.2.1 RadiusFormat rtc.aga.radarvideoformats.RadarVideoFormat.getRadiusData ( )

Returns

Deserialize data and stores in **Radius** (p. 126) Format structure

Implemented in **rtc.aga.radarvideoformats.GenericRadarVideoFormat** (p. 57).

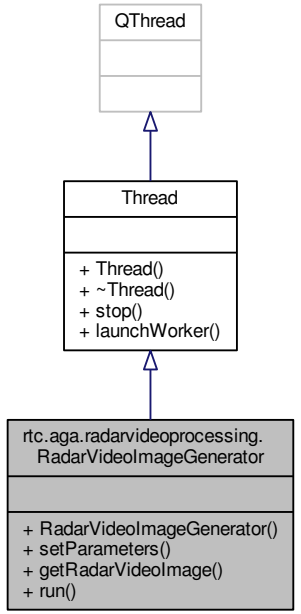
The documentation for this interface was generated from the following file:

- RaViGEnTestConsole/RaViGEnTestConsole/src/rtc/aga/radarvideoformats/RadarVideoFormat.↩ java

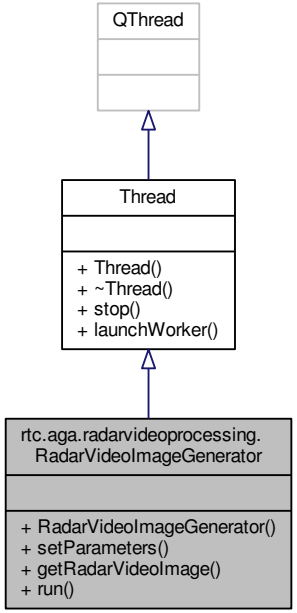
1.55 rtc.aga.radarvideoprocessing.RadarVideoImageGenerator Class Reference

RadarVideoImageGenerator (p. 118) is the worker thread in charge of generate the radar video image periodically in the given radar video image format and radar video format.

Inheritance diagram for rtc.aga.radarvideoprocessing.RadarVideoImageGenerator:



Collaboration diagram for rtc.aga.radarvideoprocessing.RadarVideoImageGenerator:



Public Member Functions

- **RadarVideoImageGenerator** (**DoubleRevolutionBuffer** doubleBuffer)

  RadarVideoImageGenerator *(p. 118) class constructor.*
- void **setParameters** (int imageSize)

  Establish the height and width of the radar image.
- BufferedImage **getRadarVideoImage** ()
- void **run** ()

  Main loop of the worker thread.

Additional Inherited Members

1.55.1 Detailed Description

RadarVideoImageGenerator (p. 118) is the worker thread in charge of generate the radar video image periodically in the given radar video image format and radar video format.

Definition at line **13** of file **RadarVideoImageGenerator.java**.

1.55.2 Constructor & Destructor Documentation

1.55.2.1 rtc.aga.radarvideoprocessing.RadarVideoImageGenerator.RadarVideoImageGenerator (DoubleRevolutionBuffer *doubleBuffer*) [inline]

RadarVideoImageGenerator (p. 118) class constructor.

Parameters

doubleBuffer	to resolve the consumer - producer problem

Definition at line **20** of file **RadarVideoImageGenerator.java**.

1.55.3 Member Function Documentation

1.55.3.1 BufferedImage rtc.aga.radarvideoprocessing.RadarVideoImageGenerator.getRadarVideoImage ( ) [inline]

Returns

Generated video radar imagen in a BufferedImage data type

Definition at line **38** of file **RadarVideoImageGenerator.java**.

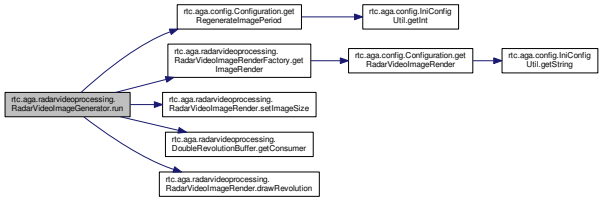
1.55.3.2 void rtc.aga.radarvideoprocessing.RadarVideoImageGenerator.run ( ) [inline]

Main loop of the worker thread.

start and end time to milliseconds

Definition at line **46** of file **RadarVideoImageGenerator.java**.

Here is the call graph for this function:



1.55.3.3 void rtc.aga.radarvideoprocessing.RadarVideoImageGenerator.setParameters (int *imageSize*) [inline]

Establish the height and width of the radar image.

Parameters

imageSize	in pixels

Definition at line **29** of file **RadarVideoImageGenerator.java**.

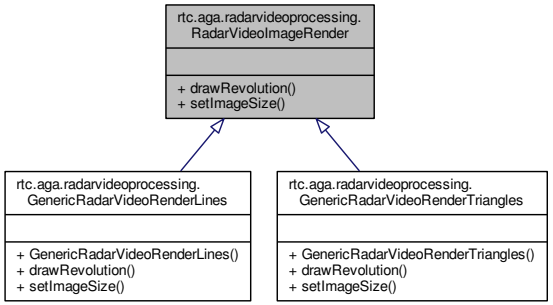
The documentation for this class was generated from the following file:

- RaViGEnTestConsole/RaViGEnTestConsole/src/rtc/aga/radarvideoprocessing/RadarVideo↩ ImageGenerator.java

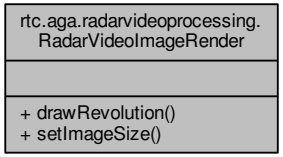
1.56 rtc.aga.radarvideoprocessing.RadarVideoImageRender Interface Reference

RadarVideoImageRender (p. 121) interface allows to RaViGEn Test Console generalize the radar video image render to accept a set of formats.

Inheritance diagram for rtc.aga.radarvideoprocessing.RadarVideoImageRender:



Collaboration diagram for rtc.aga.radarvideoprocessing.RadarVideoImageRender:



Public Member Functions

- BufferedImage **drawRevolution** (**Revolution** rev)

  *Draw in BufferedImage the **Revolution** (p. 140) rev data.*
- void **setImageSize** (int imageSize)

  setImageSize establishes the height and width of the image

1.56.1 Detailed Description

RadarVideoImageRender (p. 121) interface allows to RaViGEn Test Console generalize the radar video image render to accept a set of formats.

Definition at line **11** of file **RadarVideoImageRender.java**.

1.56.2 Member Function Documentation

1.56.2.1 BufferedImage rtc.aga.radarvideoprocessing.RadarVideoImageRender.drawRevolution (Revolution *rev*)

Draw in *BufferedImage* the **Revolution** (p. 140) *rev* data.

Parameters

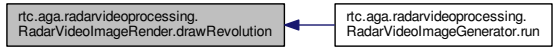
rev	*Radar* revolution

Returns

Buffered image based on a bitmap with the scene rendered

Implemented in **rtc.aga.radarvideoprocessing.GenericRadarVideoRenderTriangles** (p. 65), and **rtc.aga.radarvideoprocessing.GenericRadarVideoRenderLines** (p. 62).

Here is the caller graph for this function:



1.56.2.2 void rtc.aga.radarvideoprocessing.RadarVideoImageRender.setImageSize (int *imageSize*)

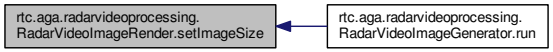
setImageSize establishes the height and width of the image

Parameters

imageSize	Image height and widht in pixeles

Implemented in **rtc.aga.radarvideoprocessing.GenericRadarVideoRenderTriangles** (p. 66), and **rtc.aga.radarvideoprocessing.GenericRadarVideoRenderLines** (p. 63).

Here is the caller graph for this function:



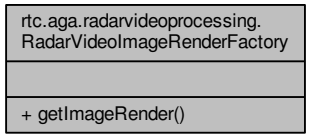
The documentation for this interface was generated from the following file:

- RaViGEnTestConsole/RaViGEnTestConsole/src/rtc/aga/radarvideoprocessing/RadarVideo↩ ImageRender.java

1.57 rtc.aga.radarvideoprocessing.RadarVideoImageRenderFactory Class Reference

RadarVideoImageRenderFactory (p. 123) allows to select the rendering method.

Collaboration diagram for rtc.aga.radarvideoprocessing.RadarVideoImageRenderFactory:



Static Public Member Functions

- static **RadarVideoImageRender getImageRender** ()

  getImageRender() *(p. 124) reads the render from the config file*

1.57.1 Detailed Description

RadarVideoImageRenderFactory (p. 123) allows to select the rendering method.

Definition at line **8** of file **RadarVideoImageRenderFactory.java**.

1.57.2 Member Function Documentation

1.57.2.1 static RadarVideoImageRender rtc.aga.radarvideoprocessing.RadarVideoImageRenderFactory.getImageRender ( ) [inline], [static]

getImageRender() (p. 124) reads the render from the config file

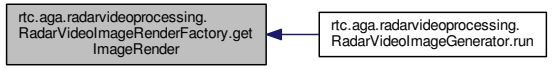
Returns

Radar video image render

Definition at line **14** of file **RadarVideoImageRenderFactory.java**.

Here is the call graph for this function:

Here is the caller graph for this function:



The documentation for this class was generated from the following file:

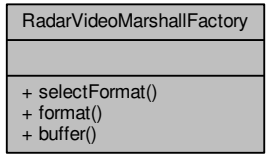
- RaViGEnTestConsole/RaViGEnTestConsole/src/rtc/aga/radarvideoprocessing/RadarVideo↩ ImageRenderFactory.java

1.58 RadarVideoMarshallFactory Class Reference

The **RadarVideoMarshallFactory** (p. 125) class implements the selection of which radar video format is going to be sended.

`#include <radarvideomarshallfactory.h>`

Collaboration diagram for RadarVideoMarshallFactory:



Static Public Member Functions

- static void **selectFormat** (QString **format**)

  selectFormat Establish the selected format in the format string
- static **RadarVideoFormat** ∗ **format** ()

  format Return The selected radar video format message
- static **RadarVideoBuffer** ∗ **buffer** ()

  buffer Return The selected radar video buffer format

1.58.1 Detailed Description

The **RadarVideoMarshallFactory** (p. 125) class implements the selection of which radar video format is going to be sended.

Definition at line **14** of file **radarvideomarshallfactory.h**.

The documentation for this class was generated from the following files:

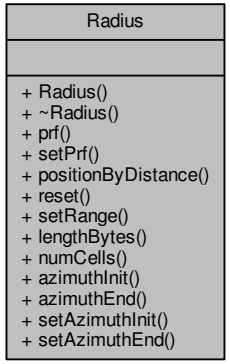
- libRadarVideoFormat/radarvideomarshallfactory.h
- libRadarVideoFormat/radarvideomarshallfactory.cpp

1.59 Radius Class Reference

The **Radius** (p. 126) class represent each beam of radar A Beam is defined by two angles and the radar range In each beam there is a fixed number of fills proportional to the cell resolution.

`#include <radius.h>`

Collaboration diagram for Radius:



Public Member Functions

- **Radius** (unsigned num_cells, unsigned range)

  Radius *(*p. *126) defines the radius based on num_cells and the range of radar.*
- ∼**Radius** ()

  Radius *(*p. *126) destructor.*
- unsigned ∗ **prf** ()

  prf represents the raw data of the radar. This method returns the prf as a pointer reference
- void **setPrf** (unsigned int ∗value)

  setPrf establish prf values of a radius
- unsigned **positionByDistance** (const double &distance, const double &scale)

  positionByDistance return the value of a position expressed in a scale
- void **reset** ()

  reset clean the radius data setting up to zero
- void **setRange** (const unsigned &meters)

  setRange establish the beam range to meters
- unsigned **lengthBytes** ()

lengthBytes returns the length in bytes of prf

- unsigned **numCells** () const

  numCells returns the number of cells in the beam

- double **azimuthInit** () const

  azimuthInit returns the initial angle of radius in degrees

- double **azimuthEnd** () const

  azimuthEnd returns the end angle of radius in degrees

- void **setAzimuthInit** (double azimuth_init)

  setAzimuthInit establish the initial angle of radius in degrees

- void **setAzimuthEnd** (double azimuth_end)

  setAzimuthEnd establish the end angle of radius in degrees

1.59.1 Detailed Description

The **Radius** (p. 126) class represent each beam of radar A Beam is defined by two angles and the radar range In each beam there is a fixed number of fills proportional to the cell resolution.

Definition at line **10** of file **radius.h**.

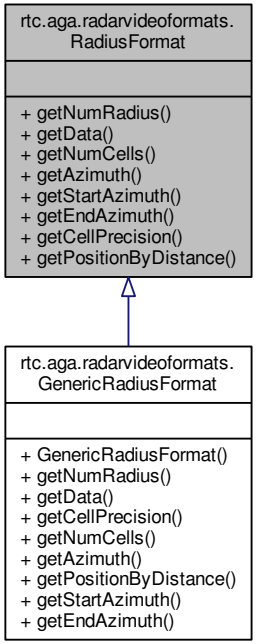
The documentation for this class was generated from the following files:

- libRadarModel/radius.h
- libRadarModel/radius.cpp

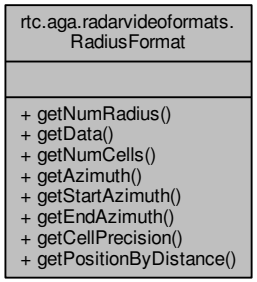
1.60 rtc.aga.radarvideoformats.RadiusFormat Interface Reference

This interface represents the generalization of **Radius** (p. 126) format that are the base of radar ***Revolution*** *(p. 140)* and radar video format.

Inheritance diagram for rtc.aga.radarvideoformats.RadiusFormat:



Collaboration diagram for rtc.aga.radarvideoformats.RadiusFormat:



Public Member Functions

- int **getNumRadius** ()
- int[ ] **getData** ()
- int **getNumCells** ()
- double **getAzimuth** ()
- double **getStartAzimuth** ()
- double **getEndAzimuth** ()
- int **getCellPrecision** ()
- int **getPositionByDistance** (double distance, double range)

  Given a distance and a range scale return a position in the raw radius data.

1.60.1 Detailed Description

This interface represents the generalization of **Radius** (p. 126) format that are the base of radar ***Revolution*** *(p. 140)* and radar video format.

Definition at line **8** of file **RadiusFormat.java**.

1.60.2 Member Function Documentation

1.60.2.1 double rtc.aga.radarvideoformats.RadiusFormat.getAzimuth ( )

Returns

The middle angle of the radius in degrees

Implemented in **rtc.aga.radarvideoformats.GenericRadiusFormat** (p. 69).

1.60.2.2 int rtc.aga.radarvideoformats.RadiusFormat.getCellPrecision ( )

Returns

The cell precision in meters

Implemented in **rtc.aga.radarvideoformats.GenericRadiusFormat** (p. 69).

1.60.2.3 int [ ] rtc.aga.radarvideoformats.RadiusFormat.getData ( )

Returns

Returns the raw data of a radius

Implemented in **rtc.aga.radarvideoformats.GenericRadiusFormat** (p. 70).

1.60.2.4 double rtc.aga.radarvideoformats.RadiusFormat.getEndAzimuth ( )

Returns

The end angle of the radius in degrees

Implemented in **rtc.aga.radarvideoformats.GenericRadiusFormat** (p. 70).

1.60.2.5 int rtc.aga.radarvideoformats.RadiusFormat.getNumCells ( )

Returns

Number of cells in a radius

Implemented in **rtc.aga.radarvideoformats.GenericRadiusFormat** (p. 70).

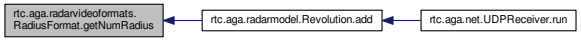
1.60.2.6 int rtc.aga.radarvideoformats.RadiusFormat.getNumRadius ( )

Returns

Sequence number of the radius in a revolution

Implemented in **rtc.aga.radarvideoformats.GenericRadiusFormat** (p. 70).

Here is the caller graph for this function:



1.60.2.7 int rtc.aga.radarvideoformats.RadiusFormat.getPositionByDistance (double *distance*, double *range*)

Given a *distance* and a *range* scale return a position in the raw radius data.

Parameters

distance	Distance to represent in meters
range	Scale of the distance

Returns

Position in the raw radius data

Implemented in **rtc.aga.radarvideoformats.GenericRadiusFormat** (p. 70).

1.60.2.8 double rtc.aga.radarvideoformats.RadiusFormat.getStartAzimuth ( )

Returns

The start angle of the radius in degrees

Implemented in **rtc.aga.radarvideoformats.GenericRadiusFormat** (p. 71).

The documentation for this interface was generated from the following file:

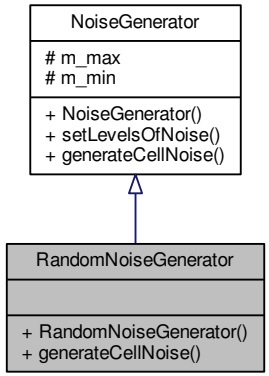
- RaViGEnTestConsole/RaViGEnTestConsole/src/rtc/aga/radarvideoformats/RadiusFormat.java

1.61 RandomNoiseGenerator Class Reference

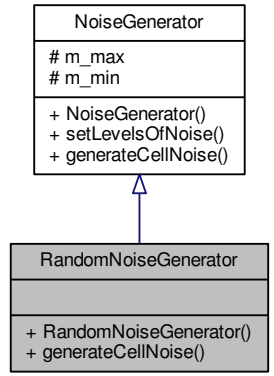
The **RandomNoiseGenerator** (p. 131) class generates aleatory noise.

`#include <randomnoisegenerator.h>`

Inheritance diagram for RandomNoiseGenerator:



Collaboration diagram for RandomNoiseGenerator:



Public Member Functions

- **RandomNoiseGenerator** ()

  RandomNoiseGenerator *(p. 131) class constructor.*
- unsigned **generateCellNoise** (unsigned pos, unsigned original_value)

  generateCellNoise to meet with base class, generate random noise for each pos based on position and sin/cos random values

Additional Inherited Members

1.61.1 Detailed Description

The **RandomNoiseGenerator** (p. 131) class generates aleatory noise.

Definition at line **13** of file **randomnoisegenerator.h**.

The documentation for this class was generated from the following files:

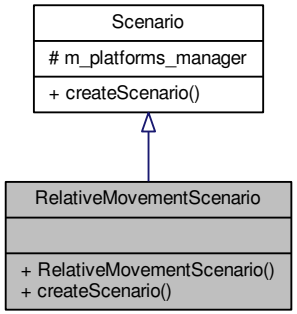
- libNoiseGenerator/randomnoisegenerator.h
- libNoiseGenerator/randomnoisegenerator.cpp

1.62 RelativeMovementScenario Class Reference

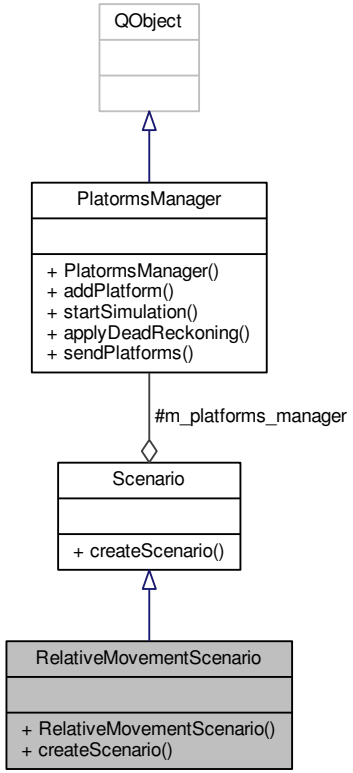
The **RelativeMovementScenario** (p. 133) class implements an circle scenario with platforms in movement.

```
#include <relativemovementscenario.h>
```

Inheritance diagram for RelativeMovementScenario:



Collaboration diagram for RelativeMovementScenario:



Public Member Functions

- **RelativeMovementScenario** (**PlatormsManager** ∗manager)

  RelativeMovementScenario *(p. 133) class method. Needs manager as paremeter to include the platforms in the scenario.*

- void **createScenario** (double range, unsigned num_platforms=10)

  createScenario creates a scenario based on range and num_platforms

Additional Inherited Members

1.62.1 Detailed Description

The **RelativeMovementScenario** (p. 133) class implements an circle scenario with platforms in movement.

Definition at line **11** of file **relativemovementscenario.h**.

The documentation for this class was generated from the following files:

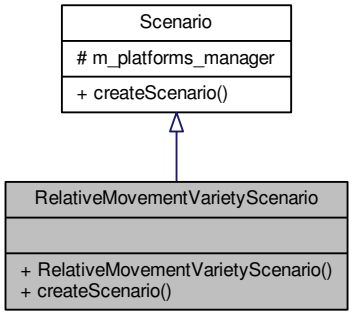
- PlatformsSimulator/relativemovementscenario.h
- PlatformsSimulator/relativemovementscenario.cpp

1.63 RelativeMovementVarietyScenario Class Reference

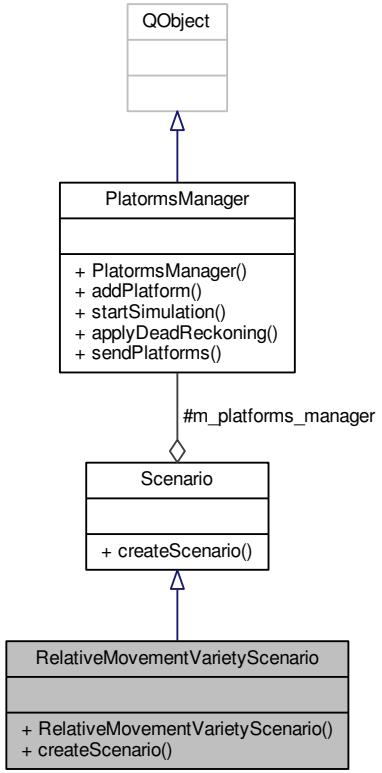
The **GeodesicCircleScenario** (p. 72) class implements a scenario that defines a circle around the reference platform with movement and a variety of platforms.

`#include <relativemovementvarietyscenario.h>`

Inheritance diagram for RelativeMovementVarietyScenario:



Collaboration diagram for RelativeMovementVarietyScenario:



Public Member Functions

- **RelativeMovementVarietyScenario** (**PlatormsManager** ∗manager)

  RelativeMovementVarietyScenario *(p. 135) class method. Needs manager as paremeter to include the platforms in the scenario.*

- void **createScenario** (double range, unsigned num_platforms=10)

  createScenario creates a scenario based on range and num_platforms

Additional Inherited Members

1.63.1 Detailed Description

The **GeodesicCircleScenario** (p. 72) class implements a scenario that defines a circle around the reference platform with movement and a variety of platforms.

Definition at line **10** of file **relativemovementvarietyscenario.h**.

The documentation for this class was generated from the following files:

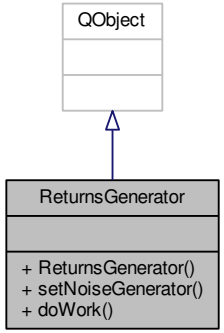
- PlatformsSimulator/relativemovementvarietyscenario.h
- PlatformsSimulator/relativemovementvarietyscenario.cpp

1.64 ReturnsGenerator Class Reference

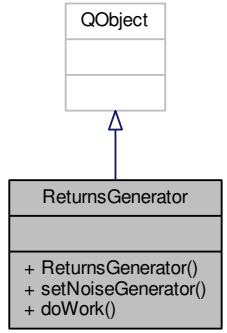
The **ReturnsGenerator** (p. 137) class is the worker is in char of generating radius to be sent for ***ReturnsSender*** *(p. 139)* applying a noise.

```
#include <returnsgenerator.h>
```

Inheritance diagram for ReturnsGenerator:



Collaboration diagram for ReturnsGenerator:



Public Slots

- void **doWork** ()

  doWork is the main loop of the worker thread

Public Member Functions

- **ReturnsGenerator** (**RadarConfigType** ∗rct, **DoubleBuffer** ∗double_buffer, QObject ∗parent=0)

  ReturnsGenerator *(p. 137) constructor needs the rct configuration of the radar, the definition of double_buffer,∗ parent is used by Qt.*
- void **setNoiseGenerator** (**NoiseGenerator** ∗noiser)

  setNoiseGenerator establishes the noise generator

1.64.1 Detailed Description

The **ReturnsGenerator** (p. 137) class is the worker is in char of generating radius to be sent for ***ReturnsSender*** *(p. 139)* applying a noise.

Definition at line **21** of file **returnsgenerator.h**.

The documentation for this class was generated from the following files:

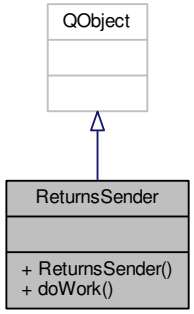
- RaViGEn/returnsgenerator.h
- RaViGEn/returnsgenerator.cpp

1.65 ReturnsSender Class Reference

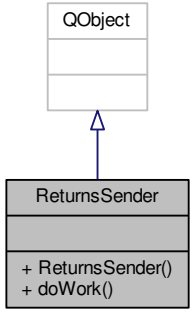
The **ReturnsSender** (p. 139) class is the worker in charge of send by multicast the radius builts by ReturnsGeneratos in previus steps.

`#include <returnssender.h>`

Inheritance diagram for ReturnsSender:



Collaboration diagram for ReturnsSender:



Public Slots

- void **doWork** ()

  doWork contains de main loop of the Returns Sender worker

Public Member Functions

- **ReturnsSender** (**RadarConfigType** ∗rct, **DoubleBuffer** ∗double_buffer, QObject ∗parent=0)

  ReturnsSender *(p. 139) Constructor Use the configuration passed in rct and the double_buffer to read the data that will be sent by work method.*

1.65.1 Detailed Description

The **ReturnsSender** (p. 139) class is the worker in charge of send by multicast the radius builts by ReturnsGeneratos in previus steps.

Definition at line **18** of file **returnssender.h**.

The documentation for this class was generated from the following files:

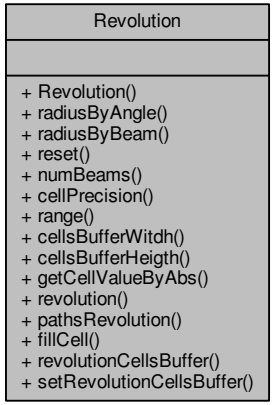
- RaViGEn/returnssender.h
- RaViGEn/returnssender.cpp

1.66 Revolution Class Reference

Revolution (p. 140) class represent a cycle of a detection radar A cycle or revolution is defined by radius and a detection cells buffers A radius is composed by cells, the minimun unit of a detection radar Each cells is a set of bits represented by a bitmap.

`#include <revolution.h>`

Collaboration diagram for Revolution:



Public Member Functions

- **Revolution** (unsigned int **range**, double azimuth_precision, unsigned int **cellPrecision**)

  Revolution (p. *140) constructor* ***Revolution*** (p. *140) defined by a range in meters, an azimuth_precision degress and cellPrecision in meters.*
- **Radius** ∗ **radiusByAngle** (double angle)

  radiusByAngle Return the ***Radius*** (p. *126) given by an /a angle in degrees*
- **Radius** ∗ **radiusByBeam** (unsigned i)

  radiusByBeam Return the ***Radius*** (p. *126) given by an /a i secuence number of beam*
- void **reset** ()

  reset Clean every data structure of the class
- unsigned **numBeams** () const

  numBeams
- unsigned **cellPrecision** () const

  cellPrecision
- unsigned **range** () const

  range
- int **cellsBufferWitdh** () const

  cellsBufferWitdh
- int **cellsBufferHeigth** () const

  cellsBufferHeigth
- unsigned **getCellValueByAbs** (int absolute_position) const

  getCellValueByAbs Given an absolute position return the cell in this position. The absolution_position reference the secuence position if all the cells were in the same radius
- std::vector< **Radius** ∗ > **revolution** () const

  revolution
- std::vector< QPainterPath ∗ > ∗ **pathsRevolution** ()

  pathsRevolution
- void **fillCell** (int absolute_position, unsigned int value)

  fillCell Fill with value the detection cell in the absolute_position
- QImage ∗ **revolutionCellsBuffer** () const

  revolutionCellsBuffer
- void **setRevolutionCellsBuffer** (QImage ∗revolution_cellsbuffer)

  setRevolutionCellsBuffer

1.66.1 Detailed Description

Revolution (p. 140) class represent a cycle of a detection radar A cycle or revolution is defined by radius and a detection cells buffers A radius is composed by cells, the minimun unit of a detection radar Each cells is a set of bits represented by a bitmap.

Definition at line **20** of file **revolution.h**.

1.66.2 Constructor & Destructor Documentation

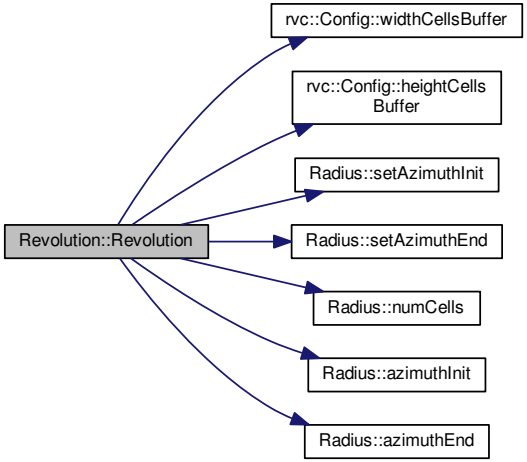
1.66.2.1 Revolution::Revolution (unsigned int *range*, double *azimuth_precision*, unsigned int *cellPrecision*)

Revolution (p. 140) constructor **Revolution** (p. 140) defined by a *range* in meters, an *azimuth_precision* degress and *cellPrecision* in meters.

Every class data is initilizate by this constructor

Definition at line **28** of file **revolution.cpp**.

Here is the call graph for this function:



1.66.3 Member Function Documentation

1.66.3.1 unsigned Revolution::cellPrecision ( ) const

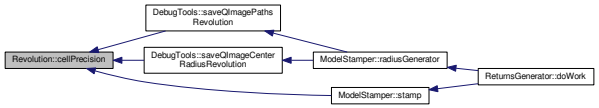
cellPrecision

Returns

Cell precision in meters

Definition at line **136** of file **revolution.cpp**.

Here is the caller graph for this function:



1.66.3.2 int Revolution::cellsBufferHeigth ( ) const

cellsBufferHeigth

Returns

Buffer heigth in bits

Definition at line **183** of file **revolution.cpp**.

1.66.3.3 int Revolution::cellsBufferWitdh ( ) const

cellsBufferWitdh

Returns

Buffer width in bits

Definition at line **178** of file **revolution.cpp**.

1.66.3.4 unsigned Revolution::numBeams ( ) const

numBeams

Returns

Number of total secuence beams

Definition at line **111** of file **revolution.cpp**.

Here is the caller graph for this function:

1.66.3.5 std::vector< QPainterPath * > * Revolution::pathsRevolution ( )

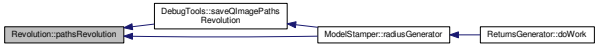
pathsRevolution

Returns

A vector will all the paths that compose the cell detection revolution

Definition at line **149** of file **revolution.cpp**.

Here is the caller graph for this function:



1.66.3.6 unsigned Revolution::range ( ) const

range

Returns

Range in meters

Definition at line **173** of file **revolution.cpp**.

Here is the caller graph for this function:

1.66.3.7 std::vector< Radius * > Revolution::revolution ( ) const

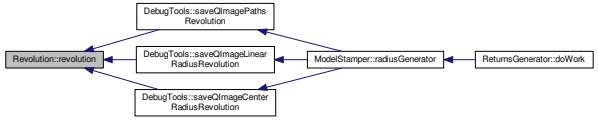
revolution

Returns

A vector with all the radius that compose the revolution

Definition at line **141** of file **revolution.cpp**.

Here is the caller graph for this function:



1.66.3.8 QImage * Revolution::revolutionCellsBuffer ( ) const

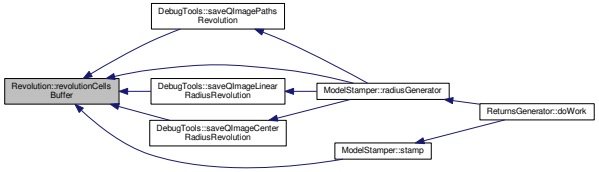
revolutionCellsBuffer

Returns

QImage bitmap representing the cell buffer detection

Definition at line **189** of file **revolution.cpp**.

Here is the caller graph for this function:



1.66.3.9 void Revolution::setRevolutionCellsBuffer (QImage * *revolution_cellsbuffer*)

setRevolutionCellsBuffer

Parameters

Sets	the revolution_cellsbuffer tha represents the cell buffer detection

Definition at line **194** of file **revolution.cpp**.

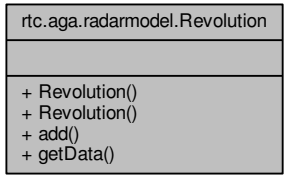
The documentation for this class was generated from the following files:

- libRadarModel/revolution.h
- libRadarModel/revolution.cpp

1.67 rtc.aga.radarmodel.Revolution Class Reference

Revolution (p. 147) class represents a cycle radar as an array of radius.

Collaboration diagram for rtc.aga.radarmodel.Revolution:



Public Member Functions

- **Revolution** (int numOfTotalRadius)
  Initializes the number of radius in a revolution.
- **Revolution** (**Revolution** another)
  Copy constructor.
- void **add** (**RadiusFormat** rf)
  Add a radius in its defined position.
- **RadiusFormat**[ ] **getData** ()
  *Query method to get the data of a **Revolution** (p. 147).*

1.67.1 Detailed Description

Revolution (p. 147) class represents a cycle radar as an array of radius.

Definition at line **8** of file **Revolution.java**.

1.67.2 Constructor & Destructor Documentation

1.67.2.1 rtc.aga.radarmodel.Revolution.Revolution (int *numOfTotalRadius*) [inline]

Initializes the number of radius in a revolution.

Parameters

numOfTotalRadius	Number of radius in a cycle or a revolution radar

Definition at line **15** of file **Revolution.java**.

1.67.2.2 rtc.aga.radarmodel.Revolution.Revolution (**Revolution** *another*) [inline]

Copy constructor.

Parameters

another	**Revolution** (p. 147) to copy

Definition at line **24** of file **Revolution.java**.

1.67.3 Member Function Documentation

1.67.3.1 void rtc.aga.radarmodel.Revolution.add (**RadiusFormat** *rf*) [inline]

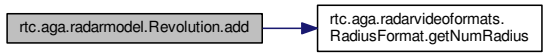
Add a radius in its defined position.

Parameters

rf	**Radius** (p. 126)

Definition at line **33** of file **Revolution.java**.

Here is the call graph for this function:



Here is the caller graph for this function:

1.67.3.2 RadiusFormat [ ] rtc.aga.radarmodel.Revolution.getData ( ) [inline]

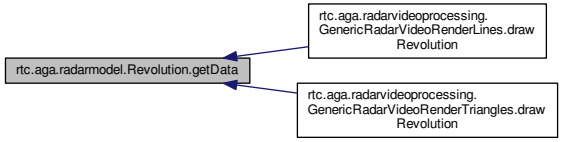
Query method to get the data of a **Revolution** (p. 147).

Returns

Revolution (p. 147) data

Definition at line **42** of file **Revolution.java**.

Here is the caller graph for this function:



The documentation for this class was generated from the following file:

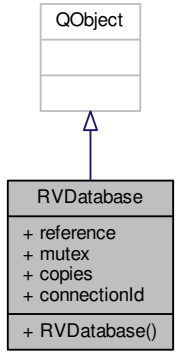
- RaViGEnTestConsole/RaViGEnTestConsole/src/rtc/aga/radarmodel/Revolution.java

1.68 RVDatabase Class Reference

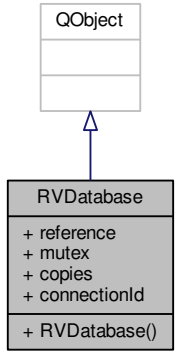
The **RVDatabase** (p. 150) class represents a set of connections to a database.

`#include <rvdatabase.h>`

Inheritance diagram for RVDatabase:



Collaboration diagram for RVDatabase:



Public Member Functions

- **RVDatabase** (QObject *parent=0)

Public Attributes

- QSqlDatabase **reference**
- QMutex **mutex**

  Reference to the database of the main thread.
- QMap< QThread *, QSqlDatabase > **copies**

  Mutex to control the acces to the copies.
- qint64 **connectionId**

  Set of connections.

1.68.1 Detailed Description

The **RVDatabase** (p. 150) class represents a set of connections to a database.

The **RVDatabase** (p. 150) provides the functionality to have a different connection to a database for each thread of a process. This class allows to execute queries in a thread-safe way.

Definition at line **21** of file **rvdatabase.h**.

1.68.2 Constructor & Destructor Documentation

1.68.2.1 RVDatabase::RVDatabase (QObject * *parent* = 0)

Constructor.

Definition at line **5** of file **rvdatabase.cpp**.

The documentation for this class was generated from the following files:

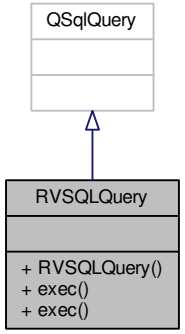
- libRVDB/rvdatabase.h
- libRVDB/rvdatabase.cpp

1.69 RVSQLQuery Class Reference

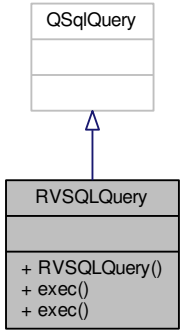
This class provides provides a means of executing and manipulating SQL statements.

`#include <rvsqlquery.h>`

Inheritance diagram for RVSQLQuery:



Collaboration diagram for RVSQLQuery:



Public Member Functions

- **RVSQLQuery** (QSqlDatabase db)

  Class constructor.
- bool **exec** ()

  *Executes a previously prepared SQL query. Note that the last error for this query is reset when **exec()** (p. 153) is called.*
- bool **exec** (QString query)

  *Executes the SQL in query. The query string must use syntax appropriate for the SQL database being queried (for example, standard SQL). Note that the last error for this query is reset when **exec()** (p. 153) is called.*

1.69.1 Detailed Description

This class provides provides a means of executing and manipulating SQL statements.

DESCRIPTION This class provides provides a means of executing and manipulating SQL statements.

RESPONSIBILITIES The member functions provides functionality to perform the following:

- Execute and manipulate SQL statements.

Definition at line **16** of file **rvsqlquery.h**.

1.69.2 Constructor & Destructor Documentation

1.69.2.1 RVSQLQuery::RVSQLQuery (QSqlDatabase *db*)

Class constructor.

Parameters

db	The database where to execute the queries.

Definition at line **14** of file **rvsqlquery.cpp**.

1.69.3 Member Function Documentation

1.69.3.1 bool RVSQLQuery::exec ( )

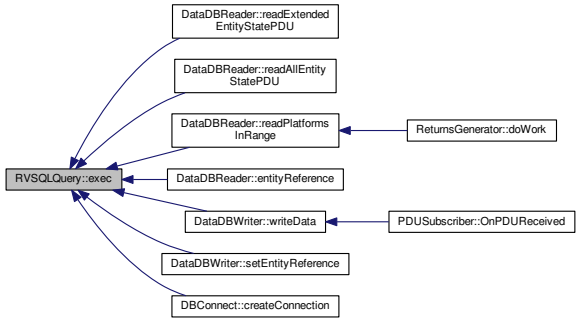
Executes a previously prepared SQL query. Note that the last error for this query is reset when **exec()** (p. 153) is called.

Returns

Returns true if the query executed successfully; otherwise returns false.

Definition at line **19** of file **rvsqlquery.cpp**.

Here is the caller graph for this function:



1.69.3.2 bool RVSQLQuery::exec (QString *query*)

Executes the SQL in query. The query string must use syntax appropriate for the SQL database being queried (for example, standard SQL). Note that the last error for this query is reset when **exec()** (p. 153) is called.

Parameters

query	The query to execute.

Returns

Returns true if the query executed successfully; otherwise returns false.

Definition at line **41** of file **rvsqlquery.cpp**.

The documentation for this class was generated from the following files:

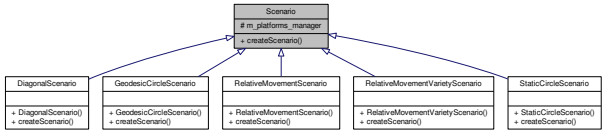
- libRVDB/rvsqlquery.h
- libRVDB/rvsqlquery.cpp

1.70 Scenario Class Reference

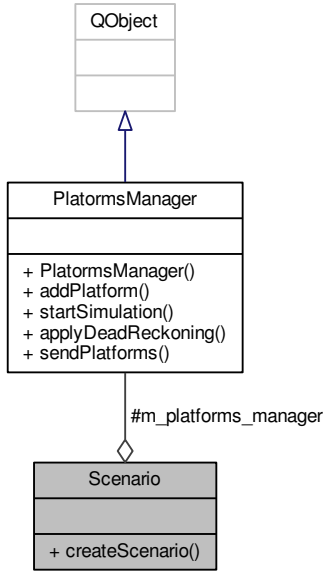
The **Scenario** (p. 155) interface is the base of the scenarios implemented in the Platforms Simulator.

`#include <scenario.h>`

Inheritance diagram for Scenario:



Collaboration diagram for Scenario:



Public Member Functions

- virtual void **createScenario** (double range=0.0, unsigned num_platforms=10)=0
  createScenario class method. Needs manager as paremeter to include the platforms in the scenario

Protected Attributes

- **PlatormsManager** ∗ **m_platforms_manager**

1.70.1 Detailed Description

The **Scenario** (p. 155) interface is the base of the scenarios implemented in the Platforms Simulator.

Definition at line **10** of file **scenario.h**.

The documentation for this class was generated from the following file:

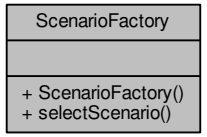
- PlatformsSimulator/scenario.h

1.71 ScenarioFactory Class Reference

The **ScenarioFactory** (p. 156) class implements the selection method of scenario.

`#include <scenariofactory.h>`

Collaboration diagram for ScenarioFactory:



Public Member Functions

- **ScenarioFactory** (**PlatormsManager** ∗manager)
  ScenarioFactory *(p. 156) needs a to create the scenarios.*
- void **selectScenario** (const QString &name)
  selectScenario establish the name scenario to be executed

1.71.1 Detailed Description

The **ScenarioFactory** (p. 156) class implements the selection method of scenario.

Definition at line **13** of file **scenariofactory.h**.

The documentation for this class was generated from the following files:

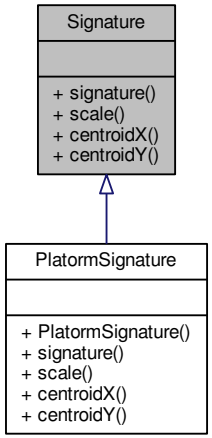
- PlatformsSimulator/scenariofactory.h
- PlatformsSimulator/scenariofactory.cpp

1.72 Signature Class Reference

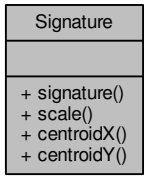
The **Signature** (p. 157) interface represent the radar signature of a platform.

`#include <signature.h>`

Inheritance diagram for Signature:



Collaboration diagram for Signature:



Public Member Functions

- virtual QImage ∗ **signature** () const =0
  signature() *(p. 158) return the bitmap representation of radarsignature*
- virtual double **scale** () const =0
  scale() *(p. 158) return the Scale of the bitmap representation*
- virtual int **centroidX** () const =0
  centroidX return the x center coordinate of the bitmap representation
- virtual int **centroidY** () const =0
  centroidY return the y center coordinate of the bitmap representation

1.72.1 Detailed Description

The **Signature** (p. 157) interface represent the radar signature of a platform.

Definition at line **9** of file **signature.h**.

1.72.2 Member Function Documentation

1.72.2.1 virtual int Signature::centroidY ( ) const `[pure virtual]`

centroidY return the y center coordinate of the bitmap representation

Returns

Implemented in **PlatormSignature** (p. 101).

The documentation for this class was generated from the following file:

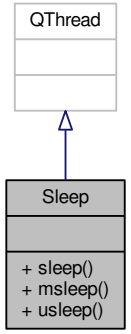
- libRadarSignature/signature.h

1.73 Sleep Class Reference

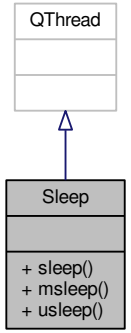
Class to be able to sleep a process for a while.

`#include <sleep.h>`

Inheritance diagram for Sleep:



Collaboration diagram for Sleep:



Static Public Member Functions

- static void **sleep** (unsigned long secs)
  Forces the current thread to sleep for secs seconds.
- static void **msleep** (unsigned long msecs)
  Forces the current thread to sleep for milliseconds.
- static void **usleep** (unsigned long usecs)
  Forces the current thread to sleep for useconds.

1.73.1 Detailed Description

Class to be able to sleep a process for a while.

The **Sleep** (p. 159) class used to be abble to force a process sleep a process for a while

Definition at line **13** of file **sleep.h**.

The documentation for this class was generated from the following files:

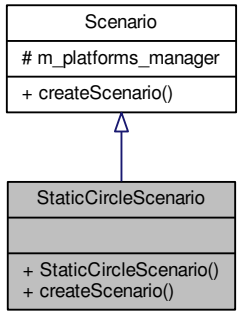
- RaViGEn/sleep.h
- RaViGEn/sleep.cpp

1.74 StaticCircleScenario Class Reference

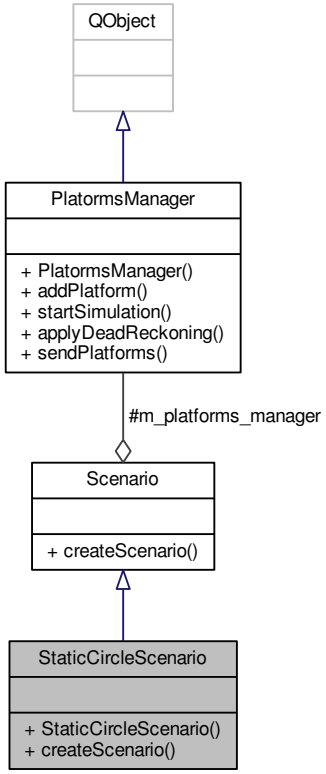
The **StaticCircleScenario** (p. 160) class implements a static scenario that defines a circle around the reference platform.

`#include <staticcirclescenario.h>`

Inheritance diagram for StaticCircleScenario:



Collaboration diagram for StaticCircleScenario:



Public Member Functions

- **StaticCircleScenario** (**PlatormsManager** ∗manager)

  StaticCircleScenario *(p. 160) class method. Needs manager as paremeter to include the platforms in the scenario.*

- void **createScenario** (double range=0.0, unsigned num_platforms=10)

  createScenario creates a scenario based on range and num_platforms

Additional Inherited Members

1.74.1 Detailed Description

The **StaticCircleScenario** (p. 160) class implements a static scenario that defines a circle around the reference platform.

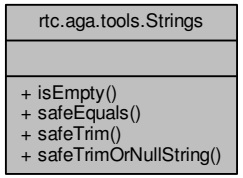
Definition at line **10** of file **staticcirclescenario.h**.

The documentation for this class was generated from the following files:

- PlatformsSimulator/staticcirclescenario.h
- PlatformsSimulator/staticcirclescenario.cpp

1.75 rtc.aga.tools.Strings Class Reference

Collaboration diagram for rtc.aga.tools.Strings:



Static Public Member Functions

- static boolean **isEmpty** (String s)
- static< T > boolean **safeEquals** (T s1, T s2)
- static String **safeTrim** (String s)
- static String **safeTrimOrNullString** (String s)

1.75.1 Detailed Description

Helper class for working with **Strings** (p. 162). [1] `http://www.programcreek.com/java-api-examples/index.php?source_dir=` argus-pdp-pep-common-master/src/main/java/org/glite/authz/common/util/**Strings.java** (p. **??**)

Definition at line **10** of file **Strings.java**.

1.75.2 Member Function Documentation

1.75.2.1 static boolean rtc.aga.tools.Strings.isEmpty (String *s*) [inline], [static]

A "safe" null/empty check for strings.

Parameters

s	The string to check

Returns

true if the string is null or the trimmed string is length zero

Definition at line **24** of file **Strings.java**.

1.75.2.2 static <T> boolean rtc.aga.tools.Strings.safeEquals (T *s1*, T *s2*) [inline], [static]

Compares two strings for equality, allowing for nulls.

Parameters

<T>	type of object to compare
s1	The first operand
s2	The second operand

Returns

true if both are null or both are non-null and the same string value

Definition at line **44** of file **Strings.java**.

1.75.2.3 static String rtc.aga.tools.Strings.safeTrim (String *s*) [inline], [static]

A safe string trim that handles nulls.

Parameters

s	the string to trim

Returns

the trimmed string or null if the given string was null

Definition at line **59** of file **Strings.java**.

1.75.2.4 static String rtc.aga.tools.Strings.safeTrimOrNullString (String *s*) [inline], [static]

Removes preceding or proceeding whitespace from a string or return null if the string is null or of zero length after trimming (i.e. if the string only contained whitespace).

Parameters

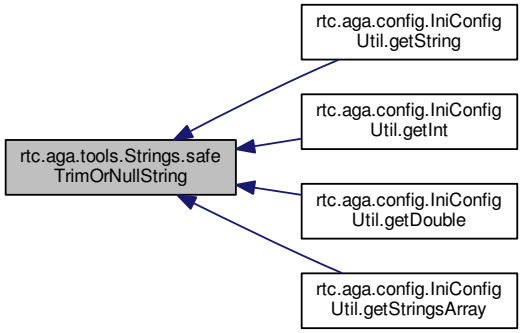
s	the string to trim

Returns

the trimmed string or null

Definition at line **75** of file **Strings.java**.

Here is the caller graph for this function:



The documentation for this class was generated from the following file:

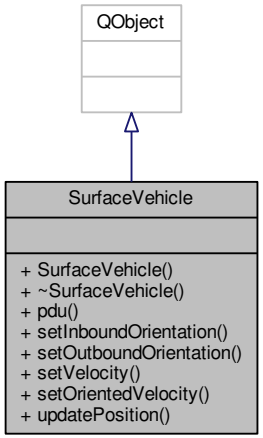
- RaViGEnTestConsole/RaViGEnTestConsole/src/rtc/aga/tools/Strings.java

1.76 SurfaceVehicle Class Reference

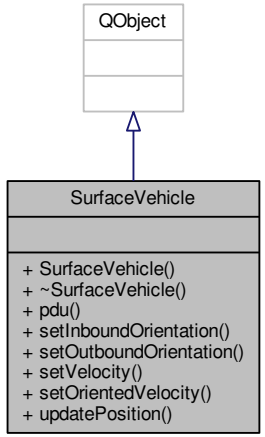
The **SurfaceVehicle** (p. 165) class represents a ship platform for simulation purpose.

```
#include <surfacevehicle.h>
```

Inheritance diagram for SurfaceVehicle:



Collaboration diagram for SurfaceVehicle:



Public Slots

- void **updatePosition** (double dt)

  updatePosition update the position for dt time applying a zero acceleration algorithm included in DIS speficiation

Public Member Functions

- **SurfaceVehicle** (Entity_State_PDU *espdu)

  SurfaceVehicle *(p. 165) is created from espdu Entity_State_PDU.*
- ~**SurfaceVehicle** ()
- Entity_State_PDU * **pdu** ()
- void **setInboundOrientation** (WorldCoordinates reference)

  setInboundOrientation sets initial orientation to reference
- void **setOutboundOrientation** (WorldCoordinates reference)

  setOutboundOrientation sets initial orientation from reference
- void **setVelocity** (double velocity)

  setVelocity establishes the velocity of platform in m/s to the north
- void **setOrientedVelocity** (double velocity)

  setVelocity establishes the velocity of platform in m/s to the heading of the surface vehicle

1.76.1 Detailed Description

The **SurfaceVehicle** (p. 165) class represents a ship platform for simulation purpose.

Definition at line **25** of file **surfacevehicle.h**.

1.76.2 Constructor & Destructor Documentation

1.76.2.1 SurfaceVehicle::~SurfaceVehicle ( )

~SurfaceVehicle class destructor

Definition at line **15** of file **surfacevehicle.cpp**.

The documentation for this class was generated from the following files:

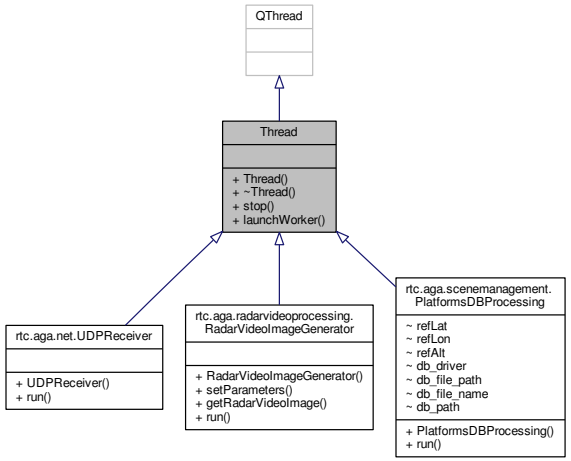
- PlatformsSimulator/surfacevehicle.h
- PlatformsSimulator/surfacevehicle.cpp

1.77 Thread Class Reference

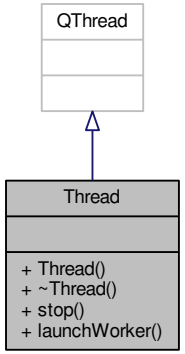
Thread (p. 167) QThread derived class with additional capability to move a QObject to the new thread, to stop the thread and move the QObject back to the thread where it came from.

`#include <thread.h>`

Inheritance diagram for Thread:



Collaboration diagram for Thread:



Signals

- void **aboutToStop** ()

  aboutToStop helper to stop worker thread

Public Member Functions

- **Thread** (QObject ∗parent=0)
- void **stop** ()

  stop puts a command to stop processing in the event queue of worker thread
- void **launchWorker** (QObject ∗worker)

  launchWorker starts thread, moves worker to this thread and blocks

1.77.1 Detailed Description

Thread (p. 167) QThread derived class with additional capability to move a QObject to the new thread, to stop the thread and move the QObject back to the thread where it came from.

Definition at line **12** of file **thread.h**.

The documentation for this class was generated from the following files:

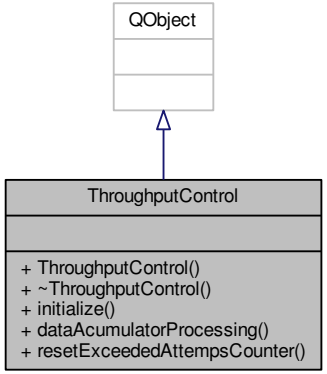
- RaViGEn/thread.h
- RaViGEn/thread.cpp

1.78 ThroughputControl Class Reference

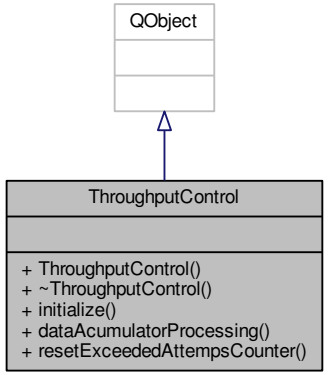
The **ThroughputControl** (p. 169) class is responsible for controlling the flow of data by reducing the amount of data to send as needed. Depends on the parameters of the configuration file.

```
#include <throughputcontrol.h>
```

Inheritance diagram for ThroughputControl:



Collaboration diagram for ThroughputControl:



Public Slots

- void **dataAcumulatorProcessing** ()

  *dataAcumulatorProcessing ask to **DataWriter** (p. 22) the amount of data processed per unit time. If the amount of data sended is greater than throughput threshold and the number of ocurrences is a certain amount per unit time revolution_time is relaxed to achieve the objective of the amount the sending data*

- void **resetExceededAttempsCounter** ()

  resetExceededAttempsCounter sets the counter to zero threshold overruns

Public Member Functions

- **ThroughputControl** (**RadarConfigType** ∗rct, **DataWriter** ∗data_writer)

  ***ThroughputControl** (p. 169) Constructor **ThroughputControl** (p. 169) modifies rct.revolution_time to adjust the amount of data to send. The data_writer parameter is used to query the data sent per time interval.*

- ∼**ThroughputControl** ()

  ∼ThroughputControl Destructor

- void **initialize** ()

  initialize starts the timers used in the class for the implemented adaptive algorithm according to the parameters defined in the configuration file.

1.78.1 Detailed Description

The **ThroughputControl** (p. 169) class is responsible for controlling the flow of data by reducing the amount of data to send as needed. Depends on the parameters of the configuration file.

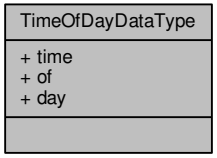
Definition at line **16** of file **throughputcontrol.h**.

The documentation for this class was generated from the following files:

- libThroughputControl/throughputcontrol.h
- libThroughputControl/throughputcontrol.cpp

1.79 TimeOfDayDataType Struct Reference

Collaboration diagram for TimeOfDayDataType:



Public Attributes

- quint8 **time**
- quint8 **of**
- quint8 **day**

1.79.1 Detailed Description

Definition at line **68** of file **asterix240format.h**.

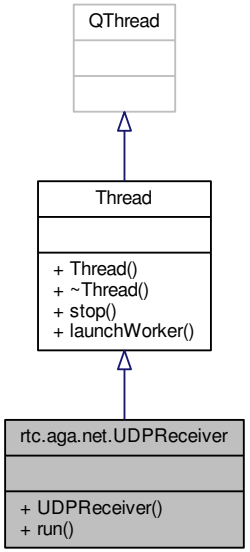
The documentation for this struct was generated from the following file:

- libRadarVideoFormat/asterix240format.h

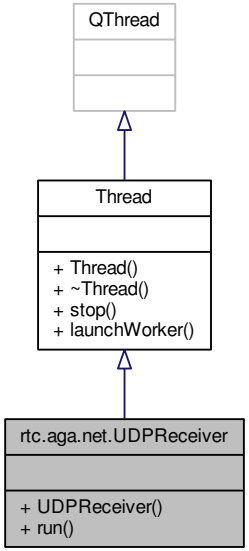
1.80 rtc.aga.net.UDPReceiver Class Reference

UDPReceiver (p. 172) is the worker thread in charge of receive the radar information and stores it in the producer copy of the buffer data.

Inheritance diagram for rtc.aga.net.UDPReceiver:



Collaboration diagram for rtc.aga.net.UDPReceiver:



Public Member Functions

- **UDPReceiver** (int port, InetAddress address, **DoubleRevolutionBuffer** doubleBuffer) throws SocketException
- void **run** ()

Additional Inherited Members

1.80.1 Detailed Description

UDPReceiver (p. 172) is the worker thread in charge of receive the radar information and stores it in the producer copy of the buffer data.

Definition at line **17** of file **UDPReceiver.java**.

1.80.2 Constructor & Destructor Documentation

1.80.2.1 rtc.aga.net.UDPReceiver.UDPReceiver (int *port*, InetAddress *address*, DoubleRevolutionBuffer *doubleBuffer*) throws SocketException [inline]

Class constructor

Parameters

port	Port to receive data
address	IP Multicast address to receive data
doubleBuffer	Reference to double buffer producer - consumer implementation

Exceptions

SocketException	Manages the socket problems / bad configurations and reusing ports problems

Definition at line **28** of file **UDPReceiver.java**.

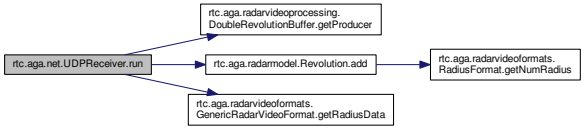
1.80.3 Member Function Documentation

1.80.3.1 void rtc.aga.net.UDPReceiver.run ( ) [inline]

Main loop of the worker thread in charge of receive data and stores in the producer copy of the double buffer implementation

Definition at line **40** of file **UDPReceiver.java**.

Here is the call graph for this function:

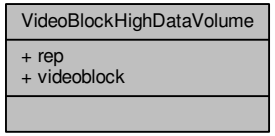


The documentation for this class was generated from the following file:

- RaViGEnTestConsole/RaViGEnTestConsole/src/rtc/aga/net/UDPReceiver.java

1.81 VideoBlockHighDataVolume Struct Reference

Collaboration diagram for VideoBlockHighDataVolume:



Public Attributes

- quint8 **rep**
- quint8 **videoblock** [kOCTECTSPERA240MESSAGE]

1.81.1 Detailed Description

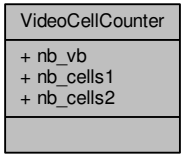
Definition at line **62** of file **asterix240format.h**.

The documentation for this struct was generated from the following file:

- libRadarVideoFormat/asterix240format.h

1.82 VideoCellCounter Struct Reference

Collaboration diagram for VideoCellCounter:



Public Attributes

- quint16 **nb_vb**
- quint16 **nb_cells1**
- quint8 **nb_cells2**

1.82.1 Detailed Description

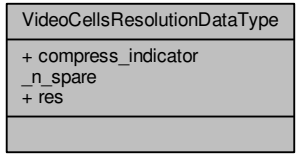
Definition at line **55** of file **asterix240format.h**.

The documentation for this struct was generated from the following file:

- libRadarVideoFormat/asterix240format.h

1.83 VideoCellsResolutionDataType Struct Reference

Collaboration diagram for VideoCellsResolutionDataType:



Public Attributes

- quint8 **compress_indicator_n_spare**
- quint8 **res**

1.83.1 Detailed Description

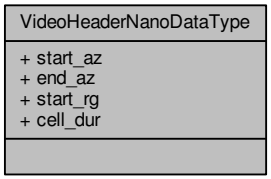
Definition at line **49** of file **asterix240format.h**.

The documentation for this struct was generated from the following file:

- libRadarVideoFormat/asterix240format.h

1.84 VideoHeaderNanoDataType Struct Reference

Collaboration diagram for VideoHeaderNanoDataType:



Public Attributes

- quint16 **start_az**
- quint16 **end_az**
- quint32 **start_rg**
- quint32 **cell_dur**

1.84.1 Detailed Description

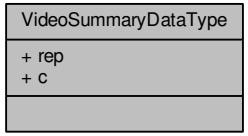
Definition at line **28** of file **asterix240format.h**.

The documentation for this struct was generated from the following file:

- libRadarVideoFormat/asterix240format.h

1.85 VideoSummaryDataType Struct Reference

Collaboration diagram for VideoSummaryDataType:



Public Attributes

- quint8 **rep**
- quint8 **c**

1.85.1 Detailed Description

Definition at line **22** of file **asterix240format.h**.

The documentation for this struct was generated from the following file:

- libRadarVideoFormat/asterix240format.h

Index

Bibliografía

[CCOO, 2010] CCOO (2010). Tablas salariales 2010 IV Convenio Colectivo. http://www.uca.es/sindicato/ccoo/documentos/tabla-salarial-pas-laboral-2010.pdf.

[David L. Parnas and Kwan, 1990] David L. Parnas, A. J. v. S. and Kwan, S. P. (1990). Evaluation of Safety-Critical Software. *Communications of the ACM*, 33(6).

[Direction generale de l'armament, 2010] Direction generale de l'armament (2010). Recommendations for a minimal implementation of DIS.

[Eckel, 2000] Eckel, B. (2000). *Thinking in C++; 2nd edition*. Prentice Hall.

[Erich Gamma, 1995] Erich Gamma, Richard Helm, R. J. J. V. (1995). *Design Patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.

[EUROCONTROL, 2009] EUROCONTROL (2009). Transmission of Monoradar Target Reports. Category 240.

[EUROCONTROL, 2012] EUROCONTROL (2012). Transmission of Monoradar Target Reports. Part 4 : Category 048.

[Firmantes Manifiesto Ágil, 2001] Firmantes Manifiesto Ágil (2001). Manifiesto por el Desarrollo Ágil de Software. http://www.agilemanifesto.org/iso/es/manifesto.html. Fecha de publicación 17 de febrero de 2001, última comprobación 28 de agosoto de 2016.

[Hodson and Gehl, 2008] Hodson, D. D. and Gehl, D. P. (2008). *Design and Implementation of Virtual Simulations*. TSEC.

[Infodefensa, 2009] Infodefensa (2009). La quinta fragata de la clase Álvaro de Bazán será botada con el nombe de Cristobal Cólon. https://www.infodefensa.com/es/2009/08/26/noticia-la-quinta-fragata-de-la-clase-alvaro-de-bazan-f-105-sera-botada-con-e.html. Fecha de publicación 26 de agosto de 2009, última comprobación 28 de agosoto de 2016.

[J.-F. Nouvel, 2004] J.-F. Nouvel, A. Herique, W. K. A. S. (2004). Radar signal simulation: Surface modeling with the Facet Method. *Radio Science*, 39(1).

[Jones, 2006] Jones, C. (2006). Social and Technical Reasons for Software Project Failure. *CrossTalk, Defense Software Engineering*, June.

[Kelly J. Hayhurst, 2001] Kelly J. Hayhurst, Dan S. Veerhusen, J. J. C. L. K. R. (2001). *A Practical Tutorial on Modified Condition/Decision Coverage.* National Aeronautics and Space Administration.

[Leffinegwell, 2010] Leffinegwell, D. (2010). *Agile Software Requirements. Lean Requirements Practices for Teams, Programs, and the Enterprise.* Ed. Addison-Wesley.

[NATO Standardization Agency, 1996] NATO Standardization Agency (1996). Recommended Practice for Distributed Interactive Simulation - Exercise Management and Feedback.

[Ramusson, 2010] Ramusson, J. (2010). *The Agile Samurai. How Agile Masters Deliver Great Software.* Ed. The Pramatic Bookshelf.

[Tim Mattson, 2009] Tim Mattson, L. M. (2009). A Hands-on Introduction to OpenMP. http://openmp.org/mp-documents/omp-hands-on-SC08.pdf.

[United States Naval Academy, 1958] United States Naval Academy (1958). *Radar and Optics.* U. S. Government.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

   As used herein, "this License" refers to version 3 of the GNU Lesser General Public License, and the "GNU GPL" refers to version 3 of the GNU General Public License.

   "The Library" refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

   An "Application" is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

   A "Combined Work" is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the "Linked Version".

   The "Minimal Corresponding Source" for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

   The "Corresponding Application Code" for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

   You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

   If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

   a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or

   b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.

b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

   You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

   a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.

   b) Accompany the Combined Work with a copy of the GNU GPL and this license document.

   c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.

   d) Do one of the following:

      0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.

      1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.

   e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

   You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.

b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

   The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

   If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.