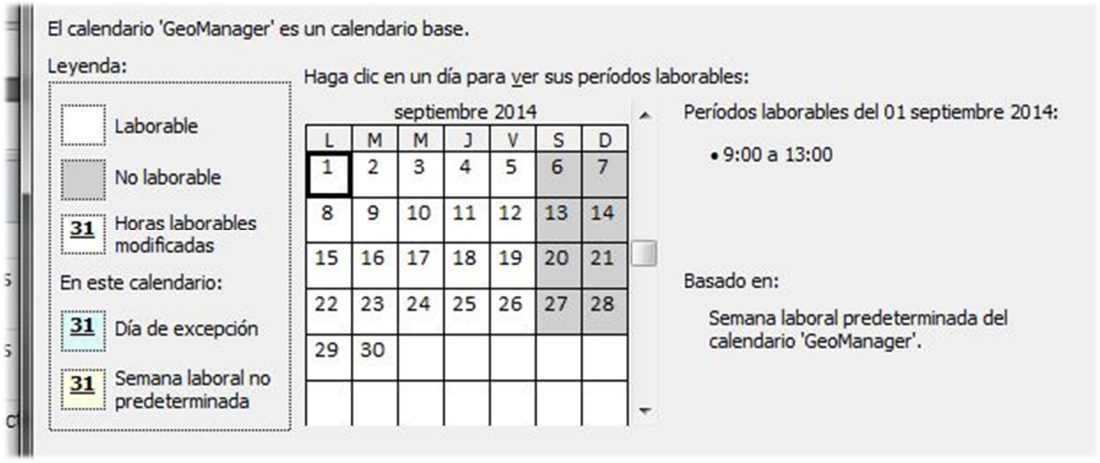
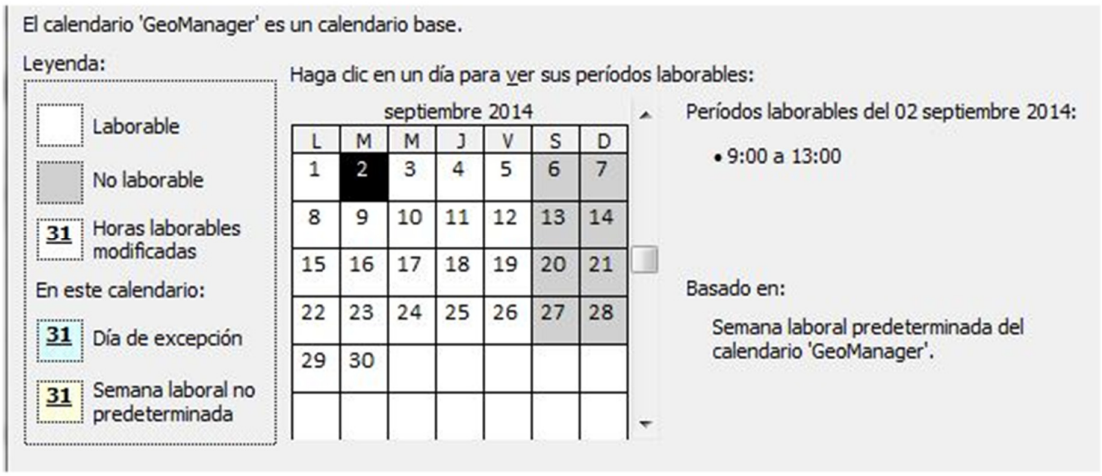
TERCERA PARTE. MODELO TEÓRICO

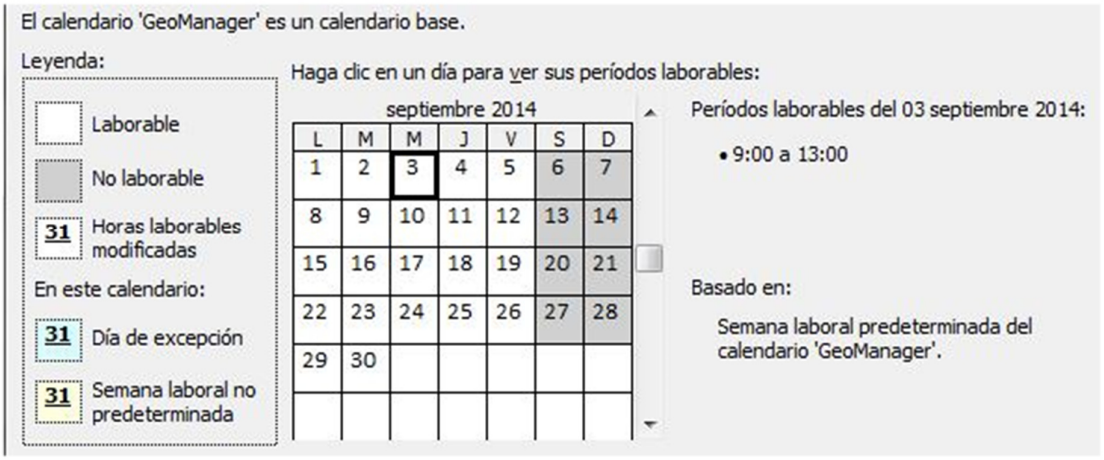
Planificación

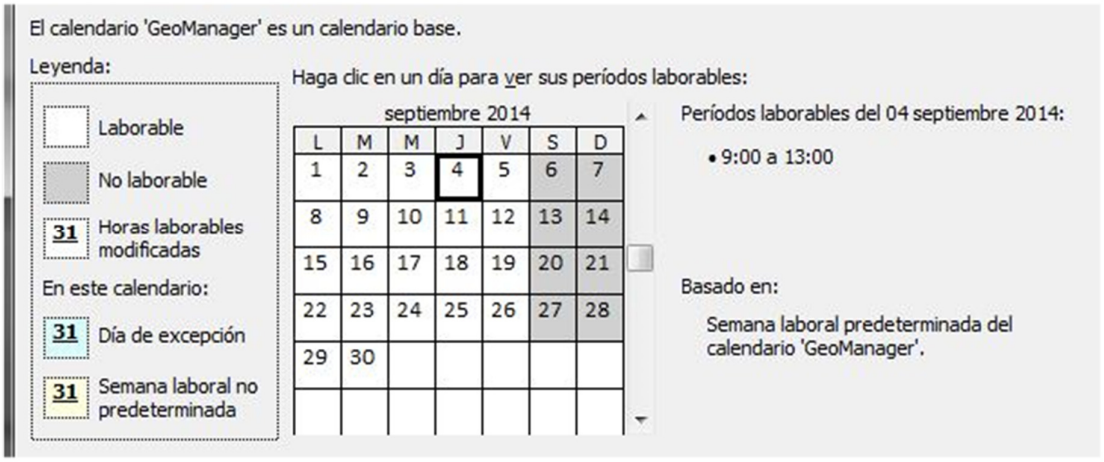
Aclaración: en este caso, el responsable de realizar todas las actividades del proyecto es el tesista/autor de este trabajo.

Calendario laboral






El calendario 'GeoManager' es un calendario base.
Leyenda:
Laborable
No laborable
31 Horas laborables modificadas
En este calendario:
31 Día de excepción
31 Semana laboral no predeterminada
Haga clic en un día para ver sus períodos laborables:
septiembre 2014
L M M J V S D
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
Períodos laborables del 03 septiembre 2014:
• 9:00 a 13:00
Basado en:
Semana laboral predeterminada del calendario 'GeoManager'.


El calendario 'GeoManager' es un calendario base.
Leyenda:
Laborable
No laborable
31 Horas laborables modificadas
En este calendario:
31 Día de excepción
31 Semana laboral no predeterminada
Haga clic en un día para ver sus períodos laborables:
septiembre 2014
L M M J V S D
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
Períodos laborables del 04 septiembre 2014:
• 9:00 a 13:00
Basado en:
Semana laboral predeterminada del calendario 'GeoManager'.

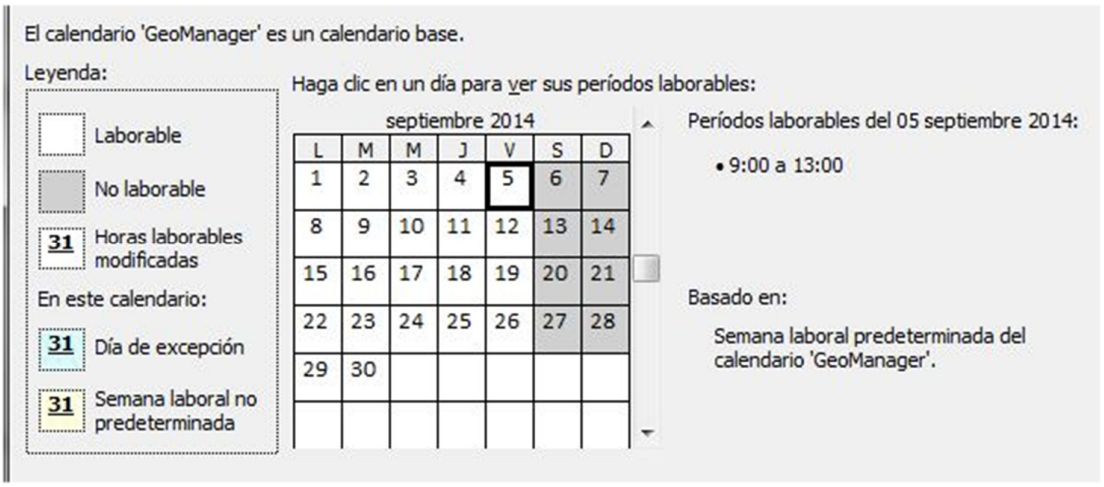
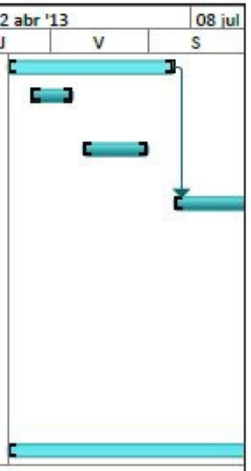

El calendario 'GeoManager' es un calendario base.
Leyenda:
Laborable
No laborable
31 Horas laborables modificadas
En este calendario:
31 Día de excepción
31 Semana laboral no predeterminada
Haga clic en un día para ver sus períodos laborables:
septiembre 2014
L M M J V S D
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
Períodos laborables del 05 septiembre 2014:
• 9:00 a 13:00
Basado en:
Semana laboral predeterminada del calendario 'GeoManager'.

Diagrama Gantt

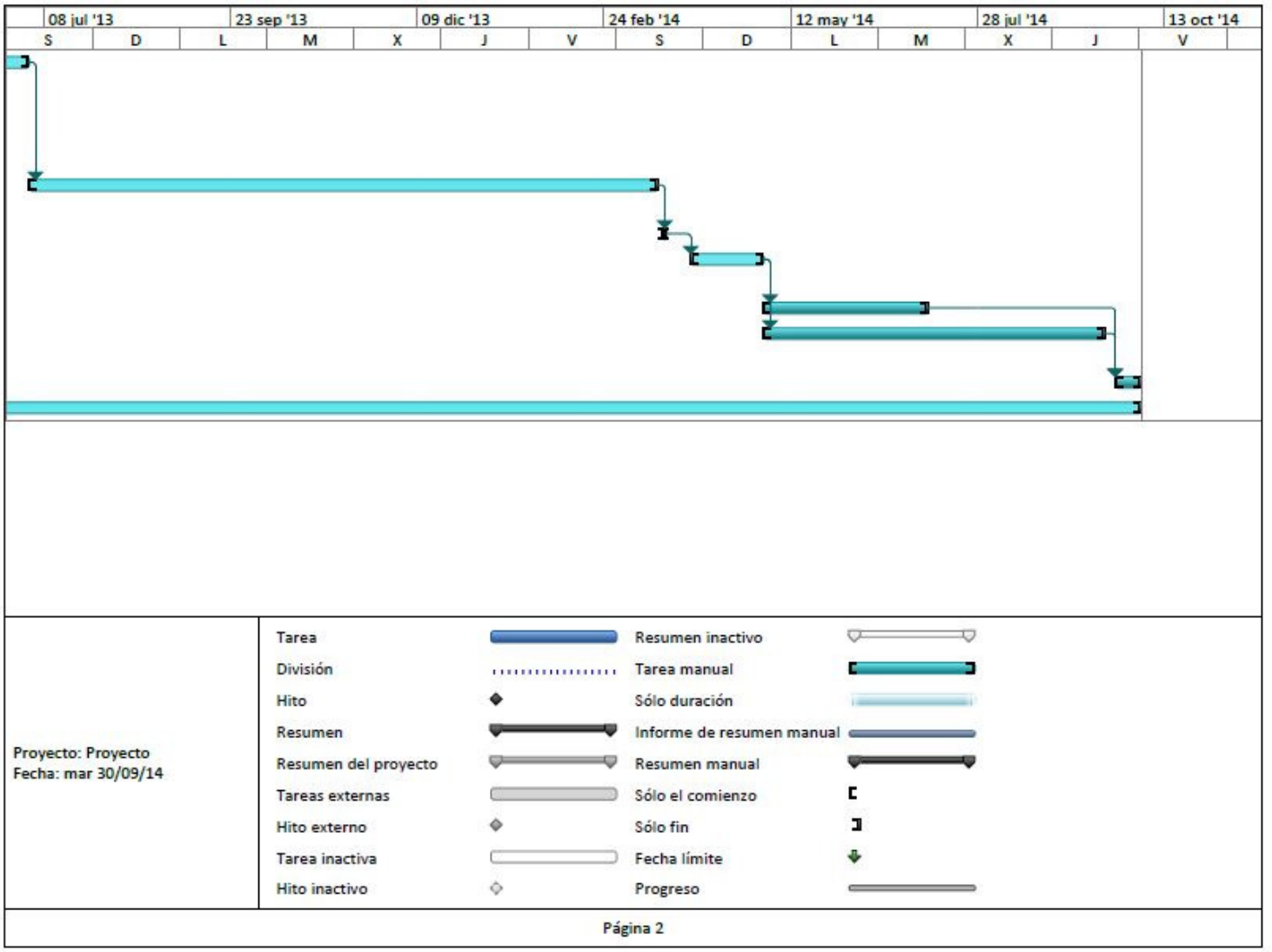
Id	Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1		Estudio de mercado	22,75 días	mié 01/05/13	lun 01/07/13	
2		Evaluación de los productos SkyPatrol	6,25 días	vie 10/05/13	vie 24/05/13	
3		Evaluación de los productos Virtec	9,75 días	mié 29/05/13	vie 21/06/13	
4		Profundización en el producto - pre compra	92,75 días	mar 02/07/13	mar 18/03/14	1
5		Compra y capacitación	1,75 días	jue 20/03/14	vie 21/03/14	4
6		Estudio de funcionalidades básicas	11,75 días	mar 01/04/14	mié 30/04/14	5
7		Desarrollo de aplicación Java	24,75 días	jue 01/05/14	lun 07/07/14	6
8		Desarrollo de aplicación GeneXus	51,25 días	jue 01/05/14	jue 18/09/14	6
9		Pruebas	5,25 días	mar 23/09/14	vie 03/10/14	7;8
10		Documentación	187,25 días	mié 01/05/13	vie 03/10/14	



Proyecto: Proyecto
Fecha: mar 30/09/14

Tarea		Resumen inactivo	
División		Tarea manual	
Hito		Sólo duración	
Resumen		Informe de resumen manual	
Resumen del proyecto		Resumen manual	
Tareas externas		Sólo el comienzo	
Hito externo		Sólo fin	
Tarea inactiva		Fecha límite	
Hito inactivo		Progreso	

Página 1


08 jul '13
23 sep '13
09 dic '13
24 feb '14
12 may '14
28 jul '14
13 oct '14
S D L M X J V S D L M X J V
Proyecto: Proyecto
Fecha: mar 30/09/14
Tarea
División
Hito
Resumen
Resumen del proyecto
Tareas externas
Hito externo
Tarea inactiva
Hito inactivo
Resumen inactivo
Tarea manual
Sólo duración
Informe de resumen manual
Resumen manual
Sólo el comienzo
Sólo fin
Fecha límite
Progreso
Página 2

Requerimientos

Actores

En el análisis de este prototipo (primera versión de GeoManager), he identificado únicamente dos actores, los cuales actúan como actores primarios ya que inician el CU.

En las versiones subsiguientes de GeoManager se tomará en cuenta al sistema GIC como actor secundario (ya que la intención de Legado IT es integrar ambas aplicaciones), pero por el momento este análisis se limita al estudio del prototipo software a desarrollar.

Entonces,

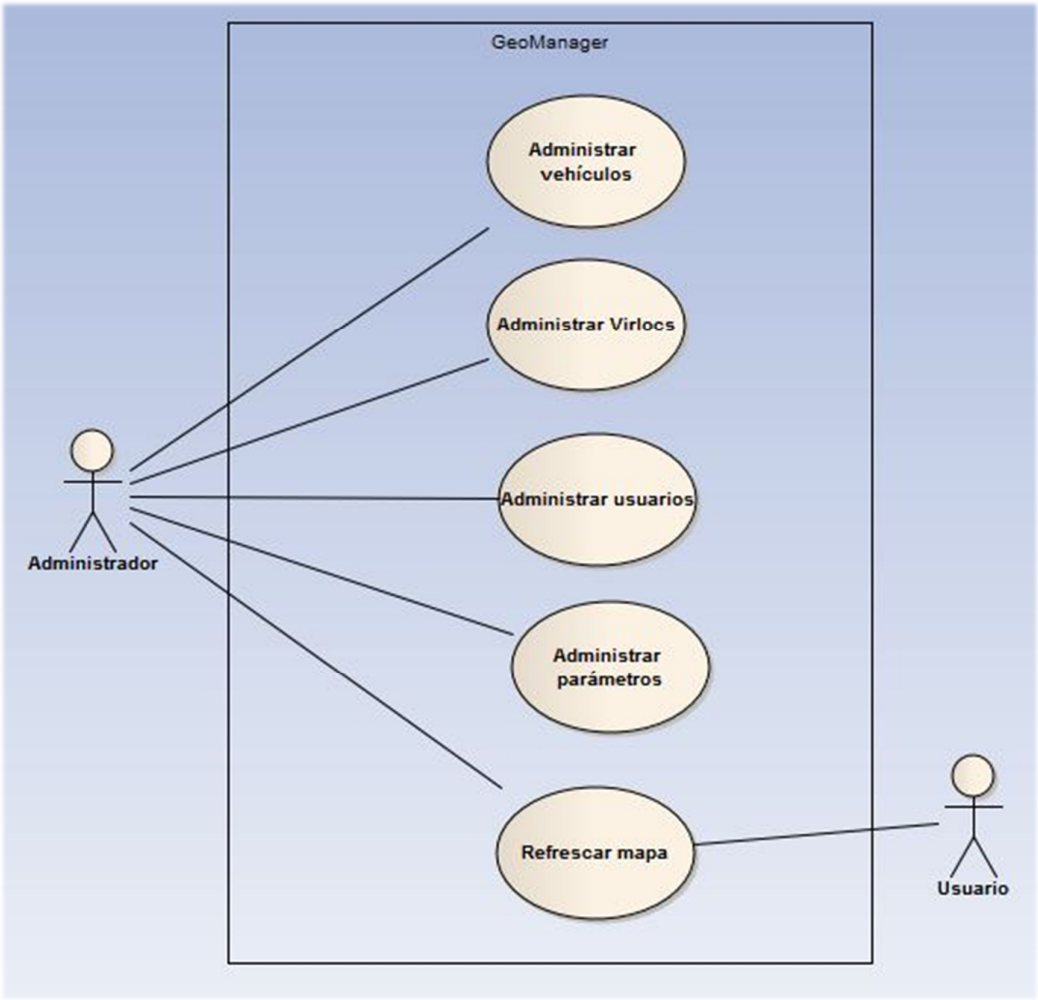
Administrador

Este actor tiene habilitados todos los permisos dentro de GeoManager y por lo tanto puede dar de alta, modificar y eliminar usuarios, vehículos, Virlocs y parámetros del sistema.

Usuario

Este actor solo puede visualizar el mapa que muestra información básica de cada móvil y además tiene habilitado el botón *Refrescar* que lo que hace es enviar mediante GPRS una consulta a las distintas unidades Virlocs para que actualicen su posición, velocidad, orientación, etc.

Modelo de casos de uso con actores



Caso de uso: Administrar vehículos

Sección principal

Caso de uso: **Administrar vehículos.**

Actor: Administrador.

Propósito: Dar de alta, modificar o eliminar los vehículos pertenecientes a la empresa.

Resumen: El administrador se encargará de dar de alta nuevos vehículos, modificar los datos de los mismos o darlos de baja según sea el caso.

Tipo: Imprescindible o primario.

Precondiciones: El administrador se encuentra logueado en el sistema.

Poscondiciones: Se administraron exitosamente los vehículos de la empresa.

Curso normal de los eventos:

Acciones de los actores	*Respuestas del sistema*
1. Este caso de uso comienza cuando el administrador selecciona el ítem "Administración" en la barra de menú, y dentro de éste, el sub-ítem "Vehículos".	2. Se muestra la grilla con los distintos vehículos ya previamente cargados por algún empleado de la empresa.
3. El administrador selecciona la tarea que desea llevar a cabo: a. Para el ingreso de un nuevo vehículo, seguir en sección alta vehículo. b. Para la baja de un vehículo, seguir en sección baja vehículo. c. Para la modificación de los datos de un vehículo, seguir en sección modificación vehículo. d. Para la visualización de los datos de un vehículo, seguir en sección visualización vehículo.	

Cursos alternativos:

- Línea 3: Si el administrador selecciona el campo "Última posición registrada" de alguno de los vehículos ya previamente cargados, se abre la aplicación Google Maps mostrando la ubicación del mismo.

Sección alta vehículo

Curso normal de los eventos:

Acciones de los actores	*Respuestas del sistema*
1. El administrador acciona el botón "Nuevo vehículo".	2. Se muestra la pantalla de ingreso de un nuevo vehículo con los campos a rellenar.
3. El administrador completa los datos del nuevo vehículo.	4. Se graba el nuevo vehículo relacionándolo a éste, a través de una clave foránea, con la empresa a la cual pertenece el administrador logueado.

Cursos alternativos:

- Línea 3: si no se ingresa la patente o la descripción del nuevo vehículo, se muestra un mensaje de error y no se puede grabar el móvil en la base de datos.

Sección baja vehículo

Curso normal de los eventos:

Acciones de los actores	*Respuestas del sistema*
1. El administrador busca el móvil mediante los filtros de la cabecera y selecciona la opción "Eliminar vehículo".	2. Se muestra la pantalla del vehículo seleccionado con todos sus datos y un mensaje advirtiendo sobre la potencial baja.
3. El administrador confirma la baja.	4. Se elimina al vehículo de la base de datos.

Sección modificación vehículo

Curso normal de los eventos:

Acciones de los actores	*Respuestas del sistema*
1. El administrador busca el móvil mediante los filtros de la cabecera y selecciona la opción "Modificar vehículo".	2. Se muestra la pantalla del vehículo seleccionado con todos sus datos y los campos habilitados para permitir la modificación, exceptuando la patente ya que ésta no puede ser actualizada al tratarse de la clave primaria de la tabla.
3. El administrador modifica los datos y confirma.	4. Se modifican los datos del vehículo.

Cursos alternativos:

- Línea 3: si la descripción del vehículo se deja en blanco, se informa mediante un mensaje de error y el administrador no puede confirmar la actualización.

Sección visualización vehículo

Curso normal de los eventos:

Acciones de los actores	*Respuestas del sistema*
1. El administrador busca el móvil mediante los filtros de la cabecera y selecciona la opción "Visualizar vehículo".	2. Se muestra la pantalla del vehículo seleccionado con los datos correspondientes.

Caso de uso: Administrar Virlocs
Sección principal

Caso de uso: **Administrar Virlocs.**

Actor: Administrador.

Propósito: Dar de alta, modificar o eliminar los Virlocs pertenecientes a la empresa.

Resumen: El administrador se encargará de dar de alta nuevos Virlocs, modificar los datos de los mismos o darlos de baja según sea el caso.

Tipo: Imprescindible o primario.

Precondiciones: El administrador se encuentra logueado en el sistema.

Poscondiciones: Se administraron exitosamente los Virlocs de la empresa.

Curso normal de los eventos:

Acciones de los actores	*Respuestas del sistema*
1. Este caso de uso comienza cuando el administrador selecciona el ítem "Administración" en la barra de menú, y dentro de éste, el sub-ítem "Virlocs".	2. Se muestra la grilla con los distintos Virlocs ya previamente cargados por algún empleado de la empresa.
3. El administrador selecciona la tarea que desea llevar a cabo: a. Para el ingreso de un nuevo Virloc, seguir en sección alta Virloc. b. Para la baja de un Virloc, seguir en sección baja Virloc. c. Para la modificación de los datos de un Virloc, seguir en sección modificación Virloc. d. Para la visualización de los datos de un Virloc, seguir en sección visualización Virloc.	

Cursos alternativos:

- Línea 3: Si el administrador selecciona la columna "Última posición registrada" de alguno de los Virlocs ya previamente cargados, se abre la aplicación Google Maps mostrando la ubicación del mismo.

Sección alta Virloc

Curso normal de los eventos:

Acciones de los actores	*Respuestas del sistema*
1. El administrador acciona el botón "Nuevo Virloc".	2. Se muestra la pantalla de alta de un nuevo Virloc con los campos a rellenar.

3. El administrador completa los datos del nuevo Virloc.	4. Se graba el nuevo Virloc relacionándolo a éste, a través de una clave foránea, con la empresa a la cual pertenece el administrador logueado. También se actualizan los datos del Virloc ingresado (velocidad, orientación, etc.) en caso de que ya se haya recibido algún reporte perteneciente a éste.

Cursos alternativos:

- Línea 3: si no se ingresa el ID del nuevo Virloc, se muestra un mensaje de error y no se puede grabar el Virloc en la base de datos.

Sección baja Virloc

Curso normal de los eventos:

Acciones de los actores	*Respuestas del sistema*
1. El administrador busca el Virloc mediante los filtros de la cabecera y el ordenamiento ubicado en el pie de página, y selecciona la opción "Eliminar Virloc".	2. Se muestra la pantalla del Virloc seleccionado con todos sus datos completos (no editables), y un mensaje advirtiendo sobre la potencial baja.
3. El administrador confirma la baja.	4. Se elimina al Virloc de la base de datos.

Cursos alternativos:

- Línea 2: si el Virloc se encuentra actualmente asociado a un vehículo, se muestra un mensaje de error al respecto y no se permite confirmar la operación.

Sección modificación Virloc

Curso normal de los eventos:

Acciones de los actores	*Respuestas del sistema*
1. El administrador busca el Virloc mediante los filtros de la cabecera y el ordenamiento ubicado en el pie de página, y selecciona la opción "Modificar Virloc".	2. Se muestra la pantalla del Virloc seleccionado pero únicamente con los campos que pueden ser modificados por el administrador (que por el momento solo es la descripción), ya que el resto de los datos (velocidad, estado del GPS, etc.) se actualizan únicamente mediante la información recibida por GPRS.

3. El administrador modifica la descripción del Virloc y confirma.	4. Se modifica la descripción del VL en la base de datos.

Sección visualización Virloc

Curso normal de los eventos:

Acciones de los actores	*Respuestas del sistema*
1. El administrador busca el Virloc mediante los filtros de la cabecera y el ordenamiento ubicado en el pie de página, y selecciona la opción "Visualizar Virloc".	2. Se muestra la pantalla del Virloc seleccionado con todos sus datos completos (incluido estado del GPS, velocidad, orientación, etc.) en modo no editable.

Caso de uso: Administrar usuarios
Sección principal

Caso de uso: **Administrar usuarios.**

Actor: Administrador.

Propósito: Dar de alta, modificar o eliminar a los usuarios del sistema.

Resumen: El administrador se encargará de dar de alta nuevos usuarios, modificar los datos de los mismos o darlos de baja según sea el caso.

Tipo: Imprescindible o primario.

Precondiciones: El administrador se encuentra logueado en el sistema.

Poscondiciones: Se administraron exitosamente los usuarios del sistema.

Curso normal de los eventos:

Acciones de los actores	*Respuestas del sistema*
1. Este caso de uso comienza cuando el administrador selecciona el ítem "Administración" en la barra de menú, y dentro de éste, el sub-ítem "Usuarios".	2. Se muestra la grilla con los usuarios que han sido previamente cargados.
3. El administrador selecciona la tarea que desea llevar a cabo: a. Para el ingreso de un nuevo usuario, seguir en sección alta usuario. b. Para la baja de un usuario, seguir en sección baja usuario. c. Para la modificación de los datos de un usuario, seguir en	

sección modificación usuario. d. Para la visualización de los datos de un usuario, seguir en sección visualización usuario.	

Sección alta usuario

Curso normal de los eventos:

Acciones de los actores	*Respuestas del sistema*
1. El administrador acciona el botón "Nuevo usuario".	2. Se muestra la pantalla de ingreso de un nuevo usuario con los campos a rellenar.
3. El administrador completa los datos del nuevo usuario.	4. Se graba el nuevo usuario relacionándolo a éste, a través de una clave foránea, con la empresa a la cual pertenece el administrador logueado.

Cursos alternativos:

- Línea 3: si no se ingresa el ID, contraseña, DNI, nombre o apellido del nuevo usuario, se muestra un mensaje de error y no se lo puede grabar en la base de datos.

Sección baja usuario

Curso normal de los eventos:

Acciones de los actores	*Respuestas del sistema*
1. El administrador busca al usuario mediante los filtros de la cabecera y selecciona la opción "Eliminar usuario".	2. Se muestra la pantalla del usuario seleccionado con todos sus datos y un mensaje advirtiendo sobre la potencial baja.
3. El administrador confirma la baja.	4. Se da de baja al usuario seleccionado.

Sección modificación usuario

Curso normal de los eventos:

Acciones de los actores	*Respuestas del sistema*
1. El administrador busca al usuario mediante los filtros de la cabecera y selecciona la opción "Modificar usuario".	2. Se muestra la pantalla del usuario seleccionado con todos sus datos y los campos habilitados para permitir la modificación, excepto el ID ya que ésta no puede ser actualizado al tratarse de la clave primaria de la tabla.
3. El administrador modifica los datos y confirma.	4. Se modifican los datos del usuario en cuestión.

Cursos alternativos:

- Línea 3: si la contraseña, el DNI, el nombre o el apellido del usuario se deja en blanco, se informa mediante un mensaje de error y el administrador no puede confirmar la actualización.

Sección visualización usuario

Curso normal de los eventos:

Acciones de los actores	*Respuestas del sistema*
1. El administrador busca al usuario mediante los filtros de la cabecera y selecciona la opción "Visualizar usuario".	2. Se muestra la pantalla del usuario seleccionado con los datos correspondientes.

Caso de uso: Administrar parámetros

Sección principal

Caso de uso: **Administrar parámetros.**

Actor: Administrador.

Propósito: Dar de alta, modificar o eliminar los parámetros del sistema.

Resumen: El administrador se encargará de dar de alta nuevos parámetros, modificarlos o darlos de baja según sea el caso.

Tipo: Secundario.

Precondiciones: El administrador se encuentra logueado en el sistema.

Poscondiciones: Se administraron exitosamente los parámetros del sistema.

Curso normal de los eventos:

Acciones de los actores	*Respuestas del sistema*
1. Este caso de uso comienza cuando el administrador selecciona el ítem "Administración" en la barra de menú, y dentro de éste, el sub-ítem "Parámetros".	2. Se muestra la grilla con los distintos parámetros del sistema ya previamente cargados.
3. El administrador selecciona la tarea que desea llevar a cabo: a. Para el ingreso de un nuevo parámetro, seguir en sección alta parámetro.	

b. Para la baja de un parámetro, seguir en sección baja parámetro. c. Para la modificación de un parámetro, seguir en sección modificación parámetro. d. Para la visualización de un parámetro, seguir en sección visualización parámetro.	

Sección alta parámetro

Curso normal de los eventos:

Acciones de los actores	*Respuestas del sistema*
1. El administrador acciona el botón "Nuevo parámetro".	2. Se muestra la pantalla de ingreso de un nuevo parámetro con los campos a rellenar.
3. El administrador completa los datos del nuevo parámetro.	4. Se graba el nuevo parámetro relacionándolo a éste, a través de una clave foránea, con la empresa a la cual pertenece el administrador logueado.

Cursos alternativos:

- Línea 3: si no se ingresa la descripción del nuevo parámetro, se muestra un mensaje de error y no se puede grabar el parámetro en la base de datos.

Sección baja parámetro

Curso normal de los eventos:

Acciones de los actores	*Respuestas del sistema*
1. El administrador busca el parámetro mediante el filtro de la cabecera y selecciona la opción "Eliminar parámetro".	2. Se muestra la pantalla del parámetro seleccionado y un mensaje advirtiendo sobre la potencial baja.
3. El administrador confirma la baja.	4. Se elimina el parámetro de la base de datos.

Sección modificación parámetro

Curso normal de los eventos:

Acciones de los actores	*Respuestas del sistema*
1. El administrador busca el parámetro mediante el filtro de la cabecera y selecciona la opción "Modificar parámetro".	2. Se muestra el parámetro seleccionado en pantalla con los campos habilitados para permitir la modificación.
3. El administrador modifica los datos y confirma.	4. Se actualiza el parámetro.

Cursos alternativos:

- Línea 3: si la descripción del parámetro se deja en blanco, se informa mediante un mensaje de error y el administrador no puede confirmar la actualización.

Sección visualización parámetro

Curso normal de los eventos:

Acciones de los actores	*Respuestas del sistema*
1. El administrador busca el parámetro mediante el filtro de la cabecera y selecciona la opción "Visualizar parámetro".	2. Se muestra el parámetro seleccionado en pantalla.

Caso de uso: Refrescar mapa
Sección principal

Caso de uso: **Refrescar mapa.**

Actor: Administrador o Usuario.

Propósito: Ver la información actualizada en el mapa general de la aplicación.

Resumen: El usuario envía un comando de actualización a todos los Virlocs cargados en el sistema, de forma tal que se muestre la posición actual de cada uno en el mapa.

Tipo: Imprescindible o primario.

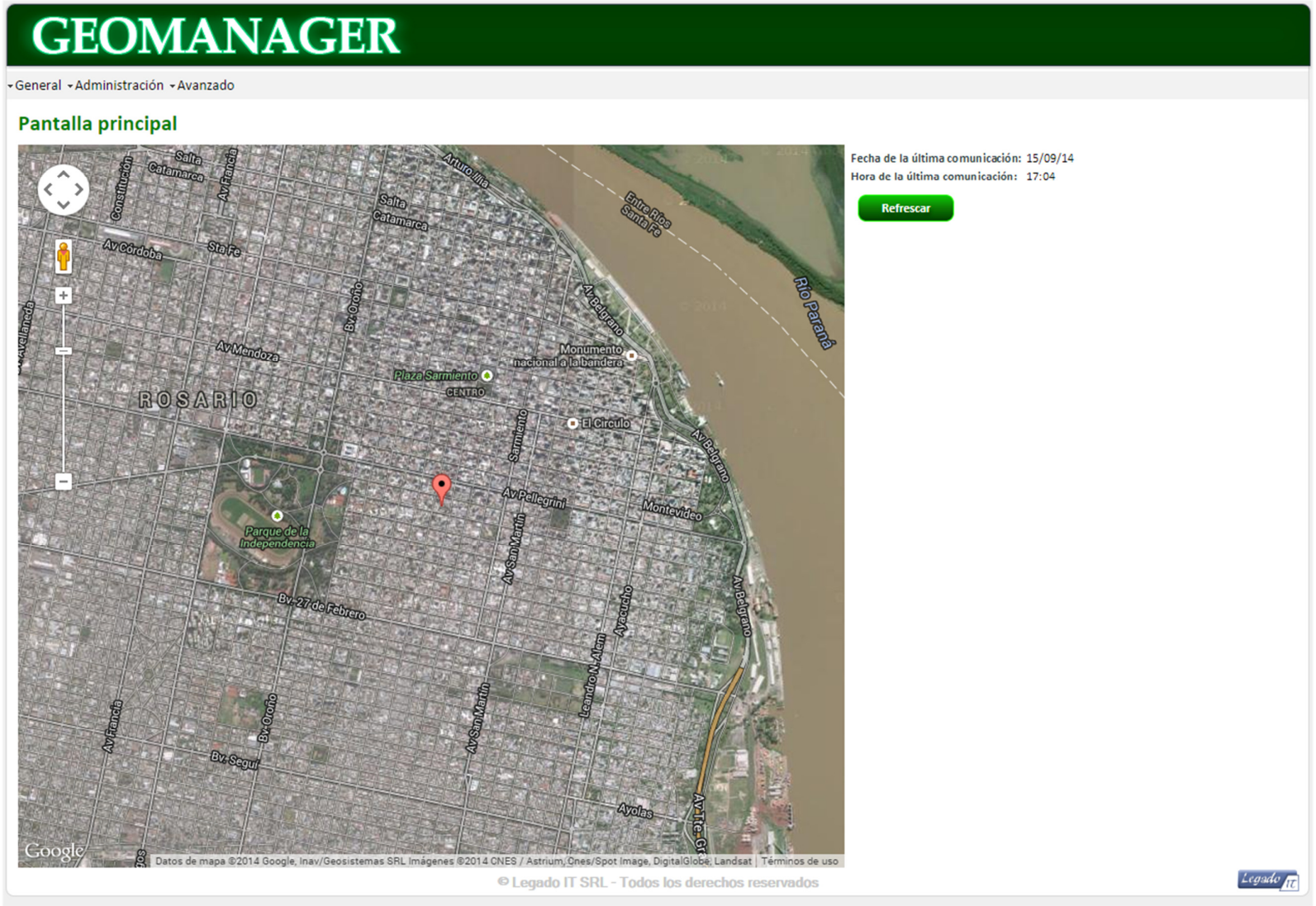
Precondiciones: El usuario se encuentra logueado en el sistema.

Poscondiciones: Se actualizó exitosamente el mapa general de la aplicación.

Curso normal de los eventos:

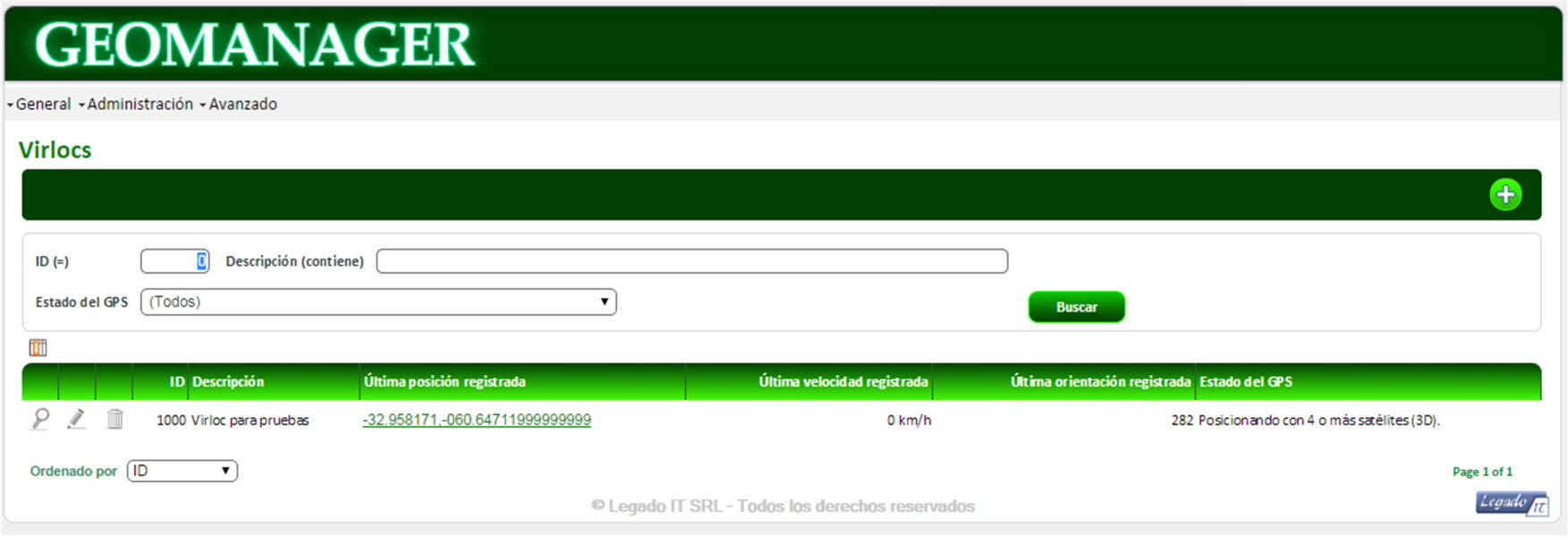
Acciones de los actores	*Respuestas del sistema*
1. Este caso de uso comienza cuando el usuario selecciona el ítem "General" en la barra de menú, y dentro de éste, el sub-ítem "Mapa".	2. Se muestra el mapa general de la aplicación con la ubicación y foto de los vehículos pertenecientes a la empresa.
3. El usuario acciona el botón "Refrescar".	4. Se envía un comando de consulta de posicionamiento global a todos los Virlocs registrados, lo cual actualiza la información en el mapa general y también así la etiqueta que muestra la fecha y hora de la última comunicación con los Virlocs.

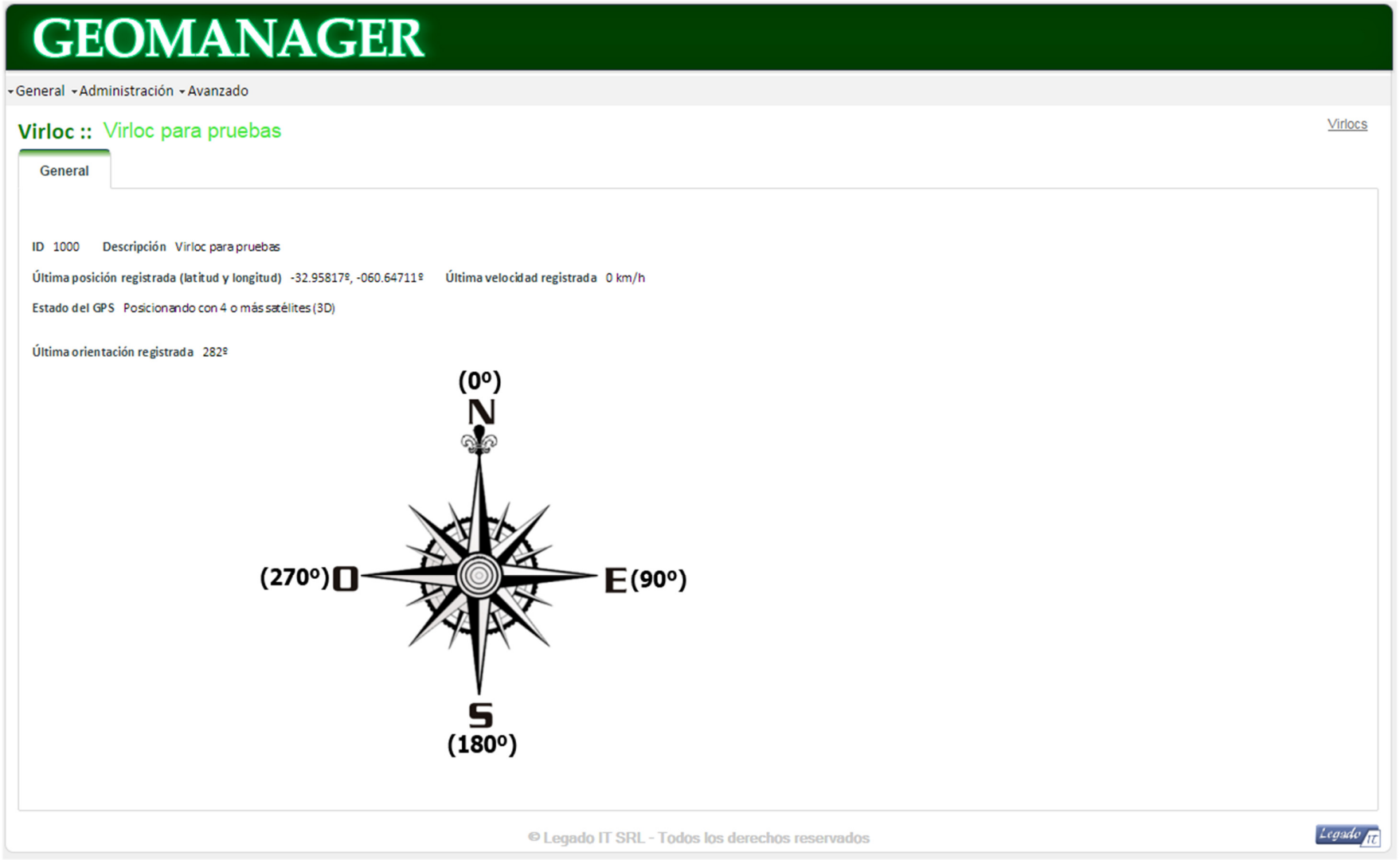
Capturas de pantalla


GEOMANAGER
General
Administración
Avanzado
Pantalla principal
Fecha de la última comunicación: 15/09/14
Hora de la última comunicación: 17:04
Refrescar
ROSARIO
Plaza Sarmiento
CENTRO
Monumento nacional a la bandera
El Circulo
Parque de la Independencia
Río Paraná
Entre Ríos Santa Fe
Av Belgrano
Av Pellegrini
Montevideo
Av San Martín
Sarmiento
Av Mendoza
Av Córdoba
Bv. 27 de Febrero
Bv. Oroño
Bv. Seguí
Ayolas
Ayacucho
Leandro N. Alem
Av Francia
Salta Catamarca
Sta Fe
Arturo Illia
Constitución
Google
Datos de mapa ©2014 Google, Inav/Geosistemas SRL Imágenes ©2014 CNES / Astrium, Cnes/Spot Image, DigitalGlobe, Landsat | Términos de uso
© Legado IT SRL - Todos los derechos reservados

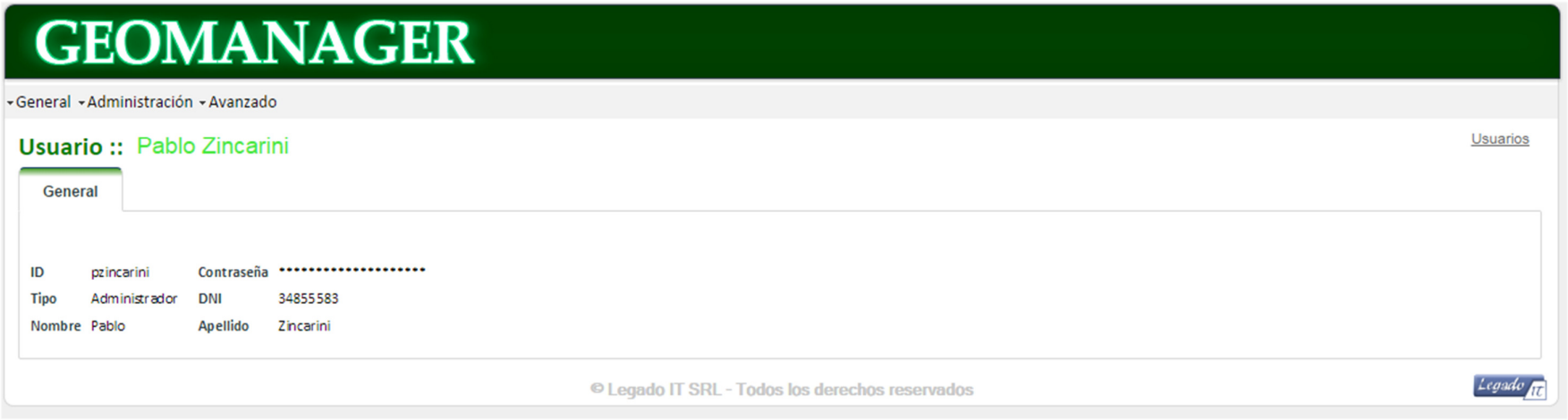

GEOMANAGER
General
Administración
Avanzado
Vehículos
Patente (contiene)
Descripcion (contiene)
Buscar
Patente
Descripción
¿El vehículo tiene Virloc asignado?
Última posición registrada
CXP504
Prueba
Sí
-32.958171,-060.64711999999999
Page 1 of 1
© Legado IT SRL - Todos los derechos reservados
Legado IT


GEOMANAGER
General
Administración
Avanzado
Vehículo :: Prueba
Vehículos
General
Patente CXP504
Descripción Prueba ID Virloc 1000
Foto
© Legado IT SRL - Todos los derechos reservados
Legado IT


GEOMANAGER
General
Administración
Avanzado
Virlocs
ID (=)
Descripción (contiene)
Estado del GPS
(Todos)
Buscar
ID
Descripción
Última posición registrada
Última velocidad registrada
Última orientación registrada
Estado del GPS
1000
Virloc para pruebas
-32.958171,-060.64711999999999
0 km/h
282
Posicionando con 4 o más satélites (3D).
Ordenado por
ID
Page 1 of 1
© Legado IT SRL - Todos los derechos reservados


GEOMANAGER
General
Administración
Avanzado
Virloc :: Virloc para pruebas
Virlocs
General
ID 1000 Descripción Virloc para pruebas
Última posición registrada (latitud y longitud) -32.95817º, -060.64711º Última velocidad registrada 0 km/h
Estado del GPS Posicionando con 4 o más satélites (3D)
Última orientación registrada 282º
(0º)
N
(270º) O
E (90º)
S
(180º)
© Legado IT SRL - Todos los derechos reservados
Legado IT


GEOMANAGER
General Administración Avanzado
Usuarios
ID (contiene)
Tipo
(Todos)
Nombre (contiene)
DNI (=)
0
Buscar
ID
Tipo
Nombre
DNI
pzincarini
Administrador
Pablo Zincarini
34855583
Page 1 of 1
© Legado IT SRL - Todos los derechos reservados


GEOMANAGER
General Administración Avanzado
Usuario :: Pablo Zincarini
Usuarios
General
ID pzincarini
Contraseña
Tipo Administrador
DNI 34855583
Nombre Pablo
Apellido Zincarini
© Legado IT SRL - Todos los derechos reservados


GEOMANAGER
General
Administración
Avanzado
Parámetros
Descripción (contiene)
Buscar
Descripción
Valor numérico
Valor alfanumérico
Page 0 of 0
© Legado IT SRL - Todos los derechos reservados
Legado IT

Requisitos adicionales (o requerimiento no funcionales)

Administración de usuarios			
(detalle) Permite determinar *inicialmente* dos categorías para los usuarios del sistema. Una para administradores (acceso a todos los módulos del sistema) y otra para el resto de los usuarios (acceso a funciones básicas del sistema).			Evidente
Aprobado	10 horas-persona	Prioridad: Crítico	Riesgo: Ordinario

Entorno web			
(detalle) El sistema debe ser desarrollado para ambiente web, de forma tal que los distintos clientes de la empresa puedan acceder a la aplicación a través del servidor principal de Legado IT.			Evidente
Aprobado	35 horas-persona	Prioridad: Crítico	Riesgo: Ordinario

Back-up			
(detalle) Se debe realizar un back-up constante del sistema (más específicamente de los datos sensibles) para prevenir cualquier falla o desperfecto técnico.			Oculta
Aprobado	15 horas-persona	Prioridad: Importante	Riesgo: Crítico

Sistema parametrizable[1]			
(detalle) Cuenta con la opción de ajustar los parámetros técnicos, comunes a todos los clientes.			Oculta
Aprobado	15 horas-persona	Prioridad: Crítico	Riesgo: Significativo

[1] Este requerimiento es distintos al mencionado en el CU Administrar parámetros, ya que allí me refiero a parámetros propios para el cliente (y gestionados por él mismo), en cambio aquí me refiero a parámetros que determinan distintas características del servidor general, como ser por ejemplo el puerto de escucha UDP de la aplicación.

Datos privados			
(detalle) No se pueden utilizar los datos de la empresa cliente (Federada SALUD en este primer caso) para otros fines que no sean los que ella especifique.			Oculta
Aprobado	10 horas-persona	Prioridad: Crítico	Riesgo: Ordinario

Datos seguros			
(detalle) Se implementan medidas de seguridad para evitar la sustracción de datos críticos de la empresa cliente por parte de cualquier delincuente informático.			Oculta
Aprobado	120 horas-persona	Prioridad: Crítico	Riesgo: Significativo

Auditoría			
(detalle) Se implementan medidas de seguridad para evitar la manipulación incorrecta o con fines malignos de los datos. Esto se refiere al registro de todas las transacciones realizadas por los usuarios.			Oculta
Propuesto	200 horas-persona	Prioridad: Importante	Riesgo: Significativo

Estética visual			
(detalle) Las interfaces gráficas deben tener un aspecto atractivo y amigable. Preferentemente deberían ser diseñadas utilizando CSS3.			Evidente
Aprobado	120 horas-persona	Prioridad: Importante	Riesgo: Ordinario

Lenguaje de programación			
(detalle) El sistema GeoManager debe ser desarrollado con GeneXus, complementando con otro lenguaje para poder trabajar a bajo nivel.			Oculta
Aprobado	200 horas-persona	Prioridad: Importante	Riesgo: Ordinario

Metodología de desarrollo			
(detalle) El proyecto debe ser llevado a cabo utilizando Proceso Unificado como metodología de desarrollo.			Oculta
Aprobado	100 horas-persona	Prioridad: Crítico	Riesgo: Ordinario

Tiempo de respuesta de la BD			
(detalle) El sistema debe responder las consultas a la base de datos en cinco segundos o menos.			Evidente
Aprobado	25 horas-persona	Prioridad: Importante	Riesgo: Importante

Tiempo de respuesta del sistema			
(detalle) El sistema debe responder a los cálculos y procedimientos solicitados (propios del mismo sistema y sin involucrar a la base de datos) en tres segundos o menos.			Evidente
Aprobado	25 horas-persona	Prioridad: Importante	Riesgo: Importante

Base de datos			
(detalle) El sistema GeoManager debe utilizar como motor de base de datos SQL Server, aunque en el prototipo de pruebas está permitido también utilizar PostgreSQL.			Oculta
Aprobado	95 horas-persona	Prioridad: Importante	Riesgo: Ordinario

Lenguaje generado			
(detalle) El lenguaje en que será generada la aplicación utilizando GeneXus debe ser .NET – C#.			Oculta
Aprobado	10 horas-persona	Prioridad: Importante	Riesgo: Ordinario

Análisis y Diseño

Alternativas del diseño general

Cuando se comenzó con el proyecto, surgieron dos alternativas sobre el diseño del software que debe interactuar con los equipos Virloc, las cuales eran muy parecidas entre sí pero con algunas diferencias esenciales:

1. El desarrollo de dos softwares de almacenamiento. El primero de ellos se ocupa de ser una especie de "tabla buffer" que recibe los paquetes y los almacena en estado puro dentro de una tabla de la Base de Datos.

   A su vez este mismo software releva una segunda tabla donde el cliente (aplicación GeneXus en este caso) pondrá las consultas o seteos que requiere de enviar a los equipos.

   El segundo software se encarga de tomar los paquetes puros de la "tabla buffer", parsearlos convenientemente e insertarlos en la Base de Datos del cliente (aplicación GeneXus).

   También se consideró la opción de entregar el paquete parseado a un Store Procedure determinado, en vez de insertarlo directamente en la tabla.
2. Se desarrolla un único software de interacción o comunicación, el cual utiliza dos hebras o hilos. La primera hebra "escucha" sobre el puerto al cual se envían los paquetes provenientes de los equipos Virloc, y automáticamente responde al equipo que envió el paquete con un mensaje ACK necesario para que no se repita la emisión del mismo paquete. A su vez guarda la cadena en estado puro en una tabla de la base de datos del cliente.

   La segunda hebra recorre otra tabla de la misma base de datos, donde se almacenan los comandos a ser enviados desde la base (aplicación GeneXus). Cuando se encuentra con un comando que aún no ha sido enviado, "le coloca una marca" y lo envía al Virloc correspondiente.

   En este caso la aplicación cliente es la encargada de parsear la información que se encuentra en la tabla de comandos recibidos.

Luego de realizar un análisis sobre ambas alternativas, decidí optar por la segunda ya que además de mejorar la performance en general, se desarrolla un único programa lo cual disminuye las posibles situaciones de conflicto.

Administración del riesgo

Considero este acápite como el más importante de todo el trabajo, ya que aquí expondré las distintas decisiones de diseño que se tuvieron que tomar con el fin de mitigar los riesgos identificados tanto inicialmente como durante el desarrollo del prototipo. La gran mayoría de ellos son de aspecto técnico y todos requirieron un análisis investigativo para poder seguir adelante y alcanzar los objetivos propuestos.

Por lo tanto, los riesgos identificados al inicio del proyecto fueron los siguientes[2]:

I. ***Comunicación vía UDP utilizando Java.*** El primer riesgo identificado fue el hecho de transmitir información mediante paquetes UDP, lo cual posibilitaría comunicarnos con el VL utilizando el protocolo GPRS.

II. ***Conversión de .jar a servicio de Windows.*** El segundo riesgo identificado inicialmente fue la transformación de un programa desarrollado en Java (que fue el lenguaje elegido desde el comienzo para el trabajo a bajo nivel) en un servicio de Windows, SO del servidor de Producción que alojaría la aplicación final.

III. ***Energización del Virloc.*** Este riesgo se refiere al hecho de suministrar energía eléctrica al VL sin que esto conlleve adquirir una batería de automóvil para las pruebas realizadas en laboratorio.

IV. ***Comunicación vía Serial.*** Se optó por realizar las primeras pruebas utilizando el protocolo Serial de forma tal que se pudiera estudiar el formato exacto en que debían enviarse los mensajes al dispositivo, y anulando así cualquier posibilidad de conflicto que provenga de la comunicación GPRS, al ser este último un protocolo más complejo en cuanto a su implementación. Dicho de otra forma, utilizando una comunicación Serial se aisló el estudio solo al formato de los comandos emitidos, por lo cual,

[2] La numeración utilizada aquí no está fundamentada en la clasificación expuesta (riesgos iniciales – riesgos posteriores) sino en el orden en que fueron mitigados los distintos riesgos, que es el mismo que llevará la explicación posterior.

cualquier problema que apareciese en la comunicación se debería únicamente a la cadena enviada.

V. ***Apertura de puerto UDP 4097.*** Habilitación del puerto necesario, tanto en el servidor de Desarrollo (equipo ubicado dentro de la LAN) como en el servidor de Producción (servidor virtual), para la comunicación con el VL.

VII. ***Transformación de la generación del Check SUM desde C hacia Java.*** La empresa productora del VL me otorgó un modelo a seguir para poder generar el Check SUM de los mensajes en protocolo XVM, pero tal modelo se encontraba codificado en C y por lo tanto se debió realizar la respectiva conversión de lenguaje.

IX. ***Generación del número de mensaje para comandos emitidos desde la base.*** Los mensajes enviados al VL deben llevar su correspondiente número de mensaje, y éste debe encontrarse en formato hexadecimal dentro de un rango del 8000 al FFFF. Este riesgo se relaciona fundamentalmente al desarrollo de la aplicación Java desde la cual tuve que realizar los respectivos controles y conversiones para la correcta emisión de los mensajes desde la base.

X. ***Implementación de las funcionalidades requeridas en la aplicación GeneXus.*** Muchas de las funcionalidades requeridas fueron necesarias implementarlas en GeneXus. Debido a que este IDE no es el más flexible en comparación con el resto de los ambientes de desarrollo, tuve que recurrir a los denominados User Control para poder alcanzar los objetivos propuestos para el sistema.

XI. ***Energización del VL para pruebas en ciudad.*** El VL cuenta *en la práctica* con una autonomía de 3 hs. utilizando su batería interna. Para poder realizar las pruebas en ciudad debí encontrar algún medio que me permitiera superar ese límite sin tener que conectarlo obligatoriamente a la batería del automóvil.

Y los que surgieron durante el avance del trabajo fueron estos otros:

VI. ***Filtrado por puerto de recepción.*** Problema que surgió cuando me encontraba en la primera etapa de desarrollo al percibir que el VL estaba rechazando los mensajes que se enviaban desde la base.

VIII. ***Gestión de usuarios de la BD.*** Este quizás fue el problema más grave que surgió durante el desarrollo del software. Fue un conflicto producido por la interacción de las distintas hebras que conformar el componente Java al acceder simultáneamente a la BD. Se logró salir adelante con una solución provisoria pero el caso aún no ha sido cerrado de forma definitiva.

XII. ***Continuidad en la comunicación por GPRS.*** Durante las pruebas finales se comprobó que en determinadas franjas horarias, debido a distintos factores el VL no logra hacer llegar correctamente sus reportes periódicos (o las respuestas a consultas o seteos) a la base.

Para esta explicación voy a seguir una secuencia u orden que no se base en la clasificación expuesta, sino en el orden en que fueron mitigados.

Antes de comenzar, reitero sobre la articulación de los sistemas que componen GeoManager con el fin de ilustrar la idea pensada inicialmente.

En primer lugar aclaro que el sistema GeoManager está compuesto por dos aplicaciones independientes y que son a su vez complementarias. Por un lado tenemos un programa desarrollado en Java (.jar) que es multihebra y corre en segundo plano dentro del servidor de la aplicación. Este software Java se encarga de la comunicación entre la aplicación principal (parte complementaria) y todos los Virlocs de los distintos clientes de Legado IT. Por otro lado, contamos con la aplicación principal, generada mediante el IDE GeneXus en código C# y que es con la cual interactúan los usuarios pertenecientes a los distintos clientes.

Entonces, mediante la primera hebra de la aplicación Java, escuchamos permanentemente sobre el puerto UDP 4097, que es por defecto al cual el Virloc envía los reportes periódicos utilizando el protocolo GPRS, y cuando recibimos información, la guardamos en la base de datos y respondemos con un mensaje ACK a la misma IP y mismo puerto desde el cual recibimos el paquete. Con esto lo que logramos es “avisarle” al VL que hemos recibido la información correctamente y que no debe reenviar el reporte. Por otro lado, desde la aplicación generada con GeneXus, se lee periódicamente la tabla en la cual almacenamos la información recuperada por la aplicación Java y actualizamos la información correspondiente. A su vez, si los usuario del sistema desean hacer una consulta o seteo en particular, luego de utilizar el control diseñado para esta función, la

aplicación GeneXus almacena el comando correspondiente en una tabla aparte, la cual es recorrida por la segunda hebra de la aplicación Java y si encuentra que algún comando aún no ha sido enviado, se encarga de realizar tal operación y de actualizar el parámetro que mantiene el valor del último comando enviado desde la base. Por lo tanto, cuando la primera hebra de la aplicación Java reciba el paquete de respuesta a la consulta o seteo enviado desde la base (aplicación GeneXus), se encargará de almacenarlo en la tabla y de responder nuevamente con el mensaje ACK, tal cual como con los reportes periódicos enviados de forma particular por los VL.

Por último, como ya se ha mencionado en el acápite anterior, la aplicación GeneXus es la encargada de parsear la información almacenada en la tabla de comandos recibidos, de forma tal que le resulte útil al usuario.

Comunicación vía UDP utilizando Java

Este riesgo en particular fue eliminado desarrollando un prototipo que simulaba la comunicación entre el servidor de la aplicación y el equipo Virloc. Más explícitamente, desarrollé dos aplicaciones Java, una que simulaba al Virloc, llamada ClienteUDP, y otra que simulaba el comportamiento del servidor, llamada ServerUDP. Entonces, desde el cliente abría un determinado puerto UDP, por ejemplo el 9876, utilizando la clase DatagramSocket, y mediante la interfaz gráfica de este programa determinaba la IP del equipo al cual se enviaría el paquete, el puerto (obviamente también UDP) y el texto a ser enviado.

Mientras tanto, dentro de la misma LAN, preparé un equipo que tenía IP estática (la misma que se ingresaba en la interfaz gráfica del cliente) y donde se encontraba corriendo el programa ServerUDP. Este programa escuchaba sobre un puerto UDP, por defecto el 9876 ya que fue el que especifiqué en el cliente, y respondía a la misma IP y mismo puerto del que recibía información, enviando el mensaje *Paquete recibido correctamente*.

Aclaro que la apertura del socket para escuchar se realiza de la misma forma que para enviar, es decir, utilizando la clase DatagramSocket que incluye el JDK de Java.

Con este prototipo se consideró mitigado el riesgo, y el próximo paso fue convertir el programa ServerUDP a un servicio de Windows.

Conversión de .jar a servicio de Windows

¿Por qué deseaba realizar esta conversión? Como expliqué párrafos anteriores, la aplicación deberá correr sobre un servidor virtual que tiene instalado como sistema operativo Windows Server, por lo cual, si se ejecuta el .jar para habilitar el servicio de escucha sobre el puerto UDP, el mismo ocuparía un espacio dentro de la barra de tareas y además habría que ejecutarlo cada vez que se reinicie el servidor (más allá de que esta operación se lleve a cabo muy pocas veces al año). Por estos dos motivos, el dueño de Legado IT pidió explícitamente que se encontrara la forma de convertir el .jar a un servicio de Windows (y de esta forma que quede el sistema más "prolijo").

Luego de la evaluación sobre las distintas alternativas con que contábamos, tomando en cuenta tanto opciones pagas como gratuitas, se optó por utilizar la librería WinRun4J, la cual cuenta con licencia CPL, y ejecutando distintos comandos a través de la consola del sistema operativo, logré convertir el programa a un servicio de Windows.

Además de esto, se configuró que tal servicio sea de tipo *Automático (inicio retrasado)*. Con esto lo que se logra es que el servicio se inicie 30 segundos luego del arranque del SO. ¿Por qué este retardo en el arranque del servicio? Debido a que nuestro programa actúa directamente sobre el servidor de bases de datos (en un principio PostgreSQL y posteriormente SQL Server). Entonces si se da el caso de que luego de un posible reinicio del servidor, el servicio de escucha arranca antes que el servicio correspondiente al DBMS, y el primero trata de guardar información o acceder a la BD (reiterando, ***antes de que el DBMS haya iniciado***) se generaría una excepción lo cual pondría en peligro la integridad del sistema. Configurando el servicio de esta forma me aseguro de que el mismo se inicie automáticamente luego de cada reinicio del servidor, pero que previamente se haya iniciado el DBMS.

Energización del Virloc

Aquí lo que se debía encontrar era una forma alternativa de energizar al VL para realizar las pruebas en laboratorio sin que esto me lleve adquirir una batería de automóvil, ya que además de que tal operación insumiría un poco más de tiempo, utilizarla para las pruebas afectaría la facilidad de movimiento del dispositivo.

Por tales motivos, luego de investigar sobre el tema surgió la alternativa de utilizar una fuente de corriente continua, la cual debía ser de 12 V – 1 A, que son los niveles de tensión y corriente con los que trabaja el VL.

Una vez adquirida dicha fuente, lo único que hizo falta fue conectar (y soldar para fijar los empalmes) ambos negativos (el del VL y el de la fuente) y el positivo de la fuente con el positivo del VL y su salida de ignición[3].

[3] La salida de ignición hace referencia a una señal que informa si el automóvil se encuentra en marcha, para luego desde la programación XVM poder aplicar reglas que actúen sobre las salidas digitales del VL según sea el caso. Para ver un diagrama general ir a Diagrama en Bloques.

Con esto ya era posible mantener al VL con energía a través de la red eléctrica domiciliaria y simular que el vehículo se encontraba en marcha gracias a la conexión de la salida de ignición.

Por último, agrego la siguiente tabla que me ayudó a determinar el estado del VL a través de su LED de GSM (el cual también actúa como indicador del estado del cargador – energía suministrada):

Estado de LED de GSM

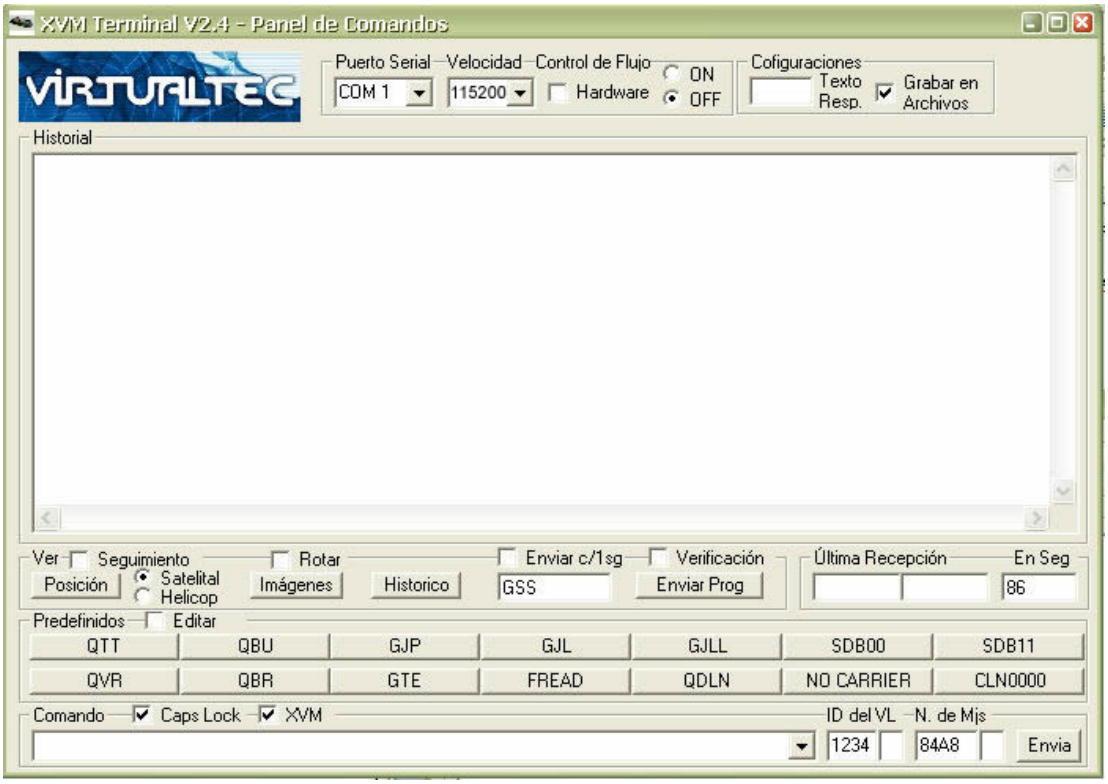
Comunicación vía Serial

Llegado a este punto, lo que necesitaba era conocer el formato en que debía enviarle las consultar al VL desde Java, es decir, si se necesitaban caracteres especiales al principio y/o final del mensaje, si los comandos debían escribirse en mayúsculas, minúsculas, o era indistinto, etc.

Gracias a la capacitación presenciada durante la compra del hardware, ya sabía que para programar los equipos bastaba con enviarle los comandos de configuración de a uno con cualquier programa terminal. Para facilitar esta tarea Virtec me brindó el programa XVM Terminal, con prestaciones orientadas al funcionamiento del protocolo XVM.

Además, Virtec me informó que para probar la configuración de los equipos tenía dos alternativas (ambas a través del puerto serie del VL):

Con XVM Terminal



Con Hyperterminal

Si utilizaba Hyperterminal para conectarme al puerto serie del VL lo debía configurar de la siguiente manera:

Configuración de puerto:

- Bits por Segundo: 115200
- Bits de Datos: 8
- Paridad: Ninguno
- Bits de parada: 1

Configuración de Sesión de Hyperterminal:

- Emulación: ANSI
- Id. Terminal de Telnet: ANSI
- Líneas de Buffer: 500

Botón de configuración ASCII en configuración de Sesión de Hyperterminal:

- Enviar fin de línea con los avances de línea = SÍ
- Eco de los caracteres escritos localmente = SÍ
- Retardo de línea: 300 milisegundos
- Retardo de caracter: 0 milisegundos

El problema fundamental de realizar las pruebas utilizando XVM Terminal era que el mismo software ya parseaba y envolvía la información de forma tal que le resulte útil al usuario final, y por lo tanto no tenía forma de ver cuál era el mensaje completo que se enviaba.

Entonces lo que hice fue desarrollar un prototipo software que interactúe vía Serial con el VL tomando como referencia los parámetros proporcionados por Virtec para la interacción mediante Hyperterminal, de forma tal que pueda probar la emisión de distintos comandos y visualizar la respuesta que recibía.

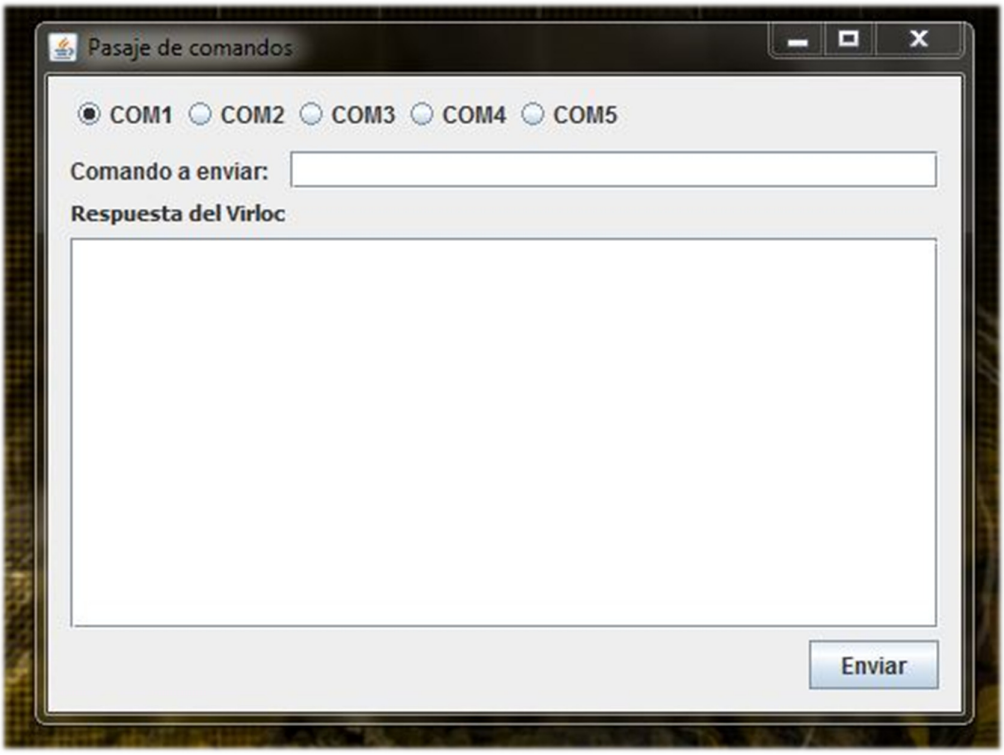
La intención de desarrollar un prototipo que interactúe a través del puerto serie de la PC era aislar el problema para que siempre que enviáramos un comando al VL éste nos

devuelva una respuesta, ya sea indicando algún tipo de error en el formato o indicando que el comando se recibió correctamente. Si hubiera emprendido la solución directamente utilizando GPRS, existía la posibilidad de que ya en el primer comando enviado con los parámetros de conexión el formato sea incorrecto y el VL nunca se pueda conectar, por lo cual nunca hubiera recibiría respuesta y las teorías sobre la causa del problema hubieran abarcado también a las inherentes al protocolo GPRS (que de hecho como se verá más adelante hubiera sido exactamente lo que hubiese pasado si afrontaba el tema utilizando GPRS desde un principio).

Nuevamente, luego de analizar las distintas alternativas que tenía para realizar la comunicación, opté por la librería RXTX mediante la cual se puede "simular" (*dentro del código*) un puerto serie, indicándole el COM utilizado y los parámetros que me indicó Virtec para la configuración de Hyperterminal. Luego, creando un objeto SerialPort en el código del programa, es posible administrarlo de forma bastante sencilla, permitiendo así por ejemplo la lectura y envío de datos como una de las funciones más simples de esta librería.

Este programa fue incluido en el prototipo ya existente que había desarrollado para interactuar mediante paquetes UDP, y por lo tanto ahora ya contaba con la posibilidad de realizar pruebas tanto mediante GPRS (a través del envío de paquetes UDP) como mediante protocolo Serial.

La parte del desarrollo que involucra a la interfaz Serial tiene como campos de entrada: el puerto COM que se va a utilizar y el comando a enviar. Y como campos de salida (o de solo lectura) el mensaje recibido en el puerto COM especificado.



Gracias a las pruebas realizadas con este prototipo se determinó el riesgo como mitigado. *El aporte más importante rescatado de este análisis fue que dentro del comando enviado al VL, siempre hablando de formato de texto plano, se debe concatenar la cadena* \r\n *al final, ya que de otra forma el VL no reconoce que se ha enviado un mensaje completo y queda esperando, sin emitir la respuesta correspondiente.*

Un riesgo menor pero relacionado fue el problema de que la mayoría computadoras actuales (sobre todo las notebooks) no cuentan con puerto serie, por lo cual no se podría utilizar ni este prototipo software ni XVM Terminal (el cual cuenta con un debugger fundamental para la configuración de los equipos) en ciertas ocasiones, como por ejemplo en la presentación de este proyecto en el aula del IUA. La solución encontrada para este inconveniente fue la utilización de un adaptador de USB a serie lo cual me independizó de forma completa de dicho puerto, comprobando también que el programa XVM Terminal se acoplaba perfectamente y no reconocía que la información estaba viajando a través del puerto USB.

Por último, paso a detallar un resumen del documento que elaboré como resultado de este estudio sobre el formato de los mensajes o protocolo para comunicarse con los Virlocs.

Protocolo de comunicación XVM:

Ejemplo de mensaje:
>QGP**;ID=**1234**;#**8000**;***5C**<**[CR][LF]

Todos los paquetes enviados al y por el dispositivo comienzan con el carácter **>** (Mayor) y terminan con la cadena **<**[CR]**[**LF**]** (Menor y Enter (Caracteres con valor ASCII 13 y 10))

Los mensajes del protocolo XVM se componen de cuatro partes las cuales se delimitan con un **;** (Punto y coma)

Las partes son las siguientes:

1. **Mensaje**
2. **ID= Nro de ID del dispositivo**
3. **# Nro de mensaje**
4. *** Check sum**

Mensaje

Empieza con las letras S, Q, R o C según sea la acción a tomar y continúa con las letras del comando. A continuación sigue una seria de caracteres variables que son los parámetros del comando.

La primera letra puede ser:

S = Set; es usado para escribir parámetros en el dispositivo.

Q = Query; es usado para preguntar el valor de parámetros al dispositivo.

C = Clear; es usado para escribir variables en cero.

R = Response; indica la respuesta del dispositivo a un comando S, Q o C.

ID del dispositivo

De fábrica viene seteado un ID que puede ser modificado por el usuario.

Este número de 4 cifras hexadecimal representa al equipo y permitirá reconocerlo dentro de la flota.

Para utilizar algunas características especiales del equipo se debe incluir alguna letra detrás del número de ID.

Número de Mensaje

Debido a que el diálogo del Virloc es siempre de ida y vuelta, los mensajes son numerados para identificar que la respuesta pertenece a un mensaje enviado determinado.

Cuando el Virloc envía un reporte generado por un evento, el mensaje llevará un número de 0001 a 7FFF y la base en respuesta a dicho mensaje deberá enviar un reconocimiento (ACK) con idéntico número de mensaje.

Si la base envía un comando al Virloc, el mensaje enviado deberá tener un número del 8000 al FFFF y el Virloc responderá dicho mensaje según corresponda pero tendrá idéntico número de mensaje que el que lo generó.

Check SUM

En formato hexadecimal que se calcula haciendo una XOR OR-exclusiva con todos los códigos ASCII de los caracteres que componen el mensaje comenzando con > y terminando en el último ; incluido pero no incluyendo el * indicador de Check SUM.

Si el check sum calculado coincide con el recibido y no se trata de un mensaje de respuesta, hay que confirmarle la correcta recepción respondiéndole el siguiente mensaje:

>ACK;ID=1017;#0093;*52<[CR][LF]

Donde:

>ACK Identificador de tipo de mensaje

; Separador

ID=1017 ID de equipo

; Separador

#0093 Número de mensaje (igual al recibido)

; Separador

*52 Check sum de este mensaje

< Fin de mensaje

[CR][LF] Fin de la cadena

Para determinar que no se trata de un mensaje de respuesta basta con comparar que el número de mensaje recibido sea inferior a 0x8000. Eso es debido a que los mensajes generados desde los Virlocs tienen un rango de 0x0000 a 0x7FFF y los generados desde la PC deben tener un rango de valores de 0x8000 a 0xFFFF.

Si no le confirma la recepción, el Virloc enviará el mensaje hasta XX veces por cada intento de comunicación telefónico sesión, es decir que todos los mensajes deben ser respondidos.

Este protocolo es el utilizado por cualquiera de los medios de comunicación, con lo cual todos los mensajes enviados al mismo van a ser respondidos con el número de mensaje correspondiente si los recibió bien.

El dispositivo analiza el mensaje recibido, si es un mensaje con respuesta responde con el mensaje correspondiente y el número de mensaje igual al recibido. Si el mensaje no tiene respuesta responde con >ACK;ID=0003;#ABC0;*2C<[CR][LF].

Si el mensaje no es respondido debe enviárselo nuevamente hasta diez veces con un intervalo de 1 segundo entre cada intento. Ej. de mensaje enviado:

>QGP;ID=6167;#FFE8;*2B<[CR][LF]

Donde:

> Comienzo del mensaje.

QGP Pedir mensaje GP (va a ser respondido con RGP) Se utiliza para saber la posición actual.

; Separador.

ID=6167 ID de dispositivo.

; Separador.

#FFE8 Número de mensaje enviado en hexadecimal (0x8000 .. 0xFFFF).

; Separador.

*2B Chksum en hexadecimal.

< Fin del mensaje.

[CR][LF] Fin de la cadena.

Múltiples comandos

Permite enviar múltiples comandos en un mismo mensaje del protocolo de transporte y especificar cuáles de ellos deben ser respondidos o no. El formato general para este tipo de mensajes es:

>comando1<>comando2;#8A1B<>comando3<>comandoN;ID=1234;#8A1C;*XX<

Cada comando puede ser cualquiera de los comandos definidos en el protocolo de transporte del VIRLOC encerrado entre “>” y “<”. Para que un comando sea respondido el mismo debe incluir un número de mensaje que se especifica con el carácter “#”. Los comandos que no poseen número de mensaje no serán respondidos.

El último comando del mensaje debe incluir el ID del equipo (ID=1234) y el checksum total del mensaje (XX) que se calcula haciendo OR exclusiva con todos los caracteres del mensaje total comenzando por el primer caracter “>” del primer comando y finalizando con el último caracter “;” que va precediendo al carácter “*” indicador del chksum (ver Cálculo del check sum).

Este tipo de comandos múltiples puede ser enviado por cualquier medio de comunicación soportado por el dispositivo. El tamaño máximo del mensaje depende del medio de comunicación a emplear pero como máximo se pueden recibir 160 caracteres.

Por ejemplo.

>SSXP01<>SSXP11<>STD090025000010000600<>QTF;ID=1234;#FAEC;*XX<

En este ejemplo el dispositivo va a responder solo el comando QTF. Esto sirve a la plataforma de software que envió el mensaje saber que el mismo llegó sin errores.

PASSWORD

Si el dispositivo posee habilitado algún password será necesario enviar al comienzo de cada comunicación el mensaje >QPWpppppppp;ID=&ldots;<. Donde pppppppp es un password válido.

VIRLOC va a responder >RPWxpppppppp;ID=&ldots;<, donde x es el número de password (0..1) que coincide con el enviado. Si el password enviado es erróneo VIRLOC responde >RPW_WRONG;ID=....< .

Si envía el mensaje >QPW;ID=... < y VIRLOC no tiene habilitado ningún password el mismo va a responder >RPW_FREE;ID=...<.

Si envía cualquier comando sin iniciar una sesión con password VIRLOC va a responder con >PASSWORD?;ID=...< a cada comando enviado.

Modificador de Reportes

Ésta es una posibilidad para interrogar al dispositivo para generar un nuevo reporte en cada mensaje que se le envía. Para efectuarlo debe enviar la siguiente cadena de caracteres:

;NR=aaabb

Donde:

aaa = Destino

bb = Tipo de mensaje

Por ejemplo:

>QGP;NR=TR2AD<

El mensaje enviado a través del COM1 interroga al dispositivo por un mensaje GP y generará un mensaje AD hacia el COM2.

Modificador de Prioridad de Respuesta

Una aplicación remota puede especificar la prioridad de la respuesta

;_HI secuencia que especifica que la respuesta tiene prioridad alta.

;_LO secuencia que especifica que la respuesta tiene prioridad baja.

Por ejemplo:

>QGP;_LO;ID=1234;#ABCD...<

Otra forma para efectuar lo mismo es incluir la siguiente letra al final del número de mensaje:

H especifica que la respuesta debe tener prioridad alta.

L especifica que la respuesta debe tener prioridad baja.

K especifica que la respuesta debe tener prioridad normal.

N especifica que no debe haber respuesta.

Por ejemplo: >QGP;ID=1234;#ABCDL;...<

Apertura de puerto UDP 4097

Cuando se analizó el primer riesgo identificado, se mencionó que las pruebas realizadas se llevaron a cabo dentro de la misma LAN, por lo que bastaba con informarle la

IP privada a nuestro Cliente UDP para que el paquete llegue a destino exitosamente. La discrepancia con la situación en la realidad es que la información debe viajar a través de internet y por lo tanto no nos serviría utilizar la *IP privada* del equipo destino (en mi caso el servidor de la aplicación).

Por este motivo, tuve que realizar nuevamente las pruebas de comunicación UDP pero utilizando la *IP pública* de salida de nuestra LAN. Cuando llevé a cabo la primera prueba noté que el paquete enviado desde el cliente no llegó nunca al servidor, y esto sucedía debido a que dicho paquete llegaba únicamente hasta el router frontera y no seguía su trayectoria hasta el servidor. Para solucionar este inconveniente tuve que llevar a cabo dos tareas.

En primer lugar asignar una *IP estática* al servidor de pruebas utilizado dentro de la LAN de la empresa para así poder identificarlo desde la consola del router y no tener que cambiar la configuración cada vez que se reiniciaba el equipo, tal como sucedería si manteníamos una configuración de *IP dinámica* en el mismo.

Y en segundo lugar "enrutar", o como la mayoría de las personas relacionadas a Sistemas llaman incorrectamente "abrir", el puerto UDP 4097. Es decir, que la información enviada al puerto 4097 sea redirigida al servidor, ya ahora con IP estática. Esto se logró sencillamente ingresando a la consola del router (marca Linksys) y aplicando esta regla de enrutamiento. Ahora ya se estaba escuchando desde el servidor de pruebas o Desarrollo los reportes enviados desde el VL, y respondiendo con los correspondientes mensajes ACK. De todas formas, cuando la aplicación se subió al servidor de Producción (servidor virtual alojado en amazon), el método para habilitar dicho puerto fue diferente, ya que allí dicho servidor contaba con IP estática y pública, y no existía ningún router o switch intermedio.

La tarea que debió llevarse a cabo fue agregar un grupo de seguridad (que es como son llamadas las reglas de seguridad aplicables desde la consola de EC2 – Amazon Web Services) donde se especificaba la habilitación del puerto UDP 4097.

Filtrado por puerto de recepción

A medida que seguí avanzando con el desarrollo del sistema noté, mediante el debugger del VL en XVM Terminal, que el dispositivo no estaba recibiendo los mensajes

ACK que se enviaban desde el servidor, por lo que se repetía la emisión de los reportes 4 o 5 veces, y luego se reiniciaba el módem interno.

Luego de varios días realizando pruebas e investigando, llegué a la conclusión de que los equipos Virloc cuentan con un filtrado por puerto de emisión/recepción. Más claramente, si el VL envía un paquete a determinada IP especificando por ejemplo el puerto 4097, no aceptará ningún paquete que provenga de un puerto distinto a ése. La causa del problema fue que en el diseño pensado inicialmente se cargaban como parámetros generales del servidor tanto el puerto de escucha como el puerto de emisión, entonces cuando utilizaba el puerto 4097 para escuchar y el 9876 para enviar la respuesta, el VL filtraba el mensaje dejándolo afuera ya que éste no provenía del puerto 4097, y emitía nuevamente el mismo reporte.

Identificado este conflicto, realicé una reestructuración del código de la aplicación Java y dejé un solo parámetro como puerto de comunicación. Desde ese momento el sistema comenzó a funcionar perfectamente y mediante los debuggers podía visualizarse cómo se recibían los paquetes UDP de forma correcta, tanto en el VL como en el servidor.

Transformación de la generación del Check SUM desde C hacia Java

Como ayuda por parte de la empresa Virtec, se nos otorgó un código de ejemplo en el lenguaje de programación C, mediante el cual se explicaba cómo generar el Check SUM (o suma de verificación) para los mensajes que utilizaban el protocolo XVM. Como se detalló anteriormente, se utiliza la operación XOR para generar el cálculo:

Cálculo del check sum

Ejemplo para calcular el check sum (fuente en C)

```
/* get chksum */

int r;

unsigned char chksum;

chksum = 0;
```

```
for (r = 0; r < strlen(modem.rxbuff); r++)

{

if (modem.rxbuff[r] == '*')

   break;

else

chksum = chksum ^ modem.rxbuff[r];

}
```

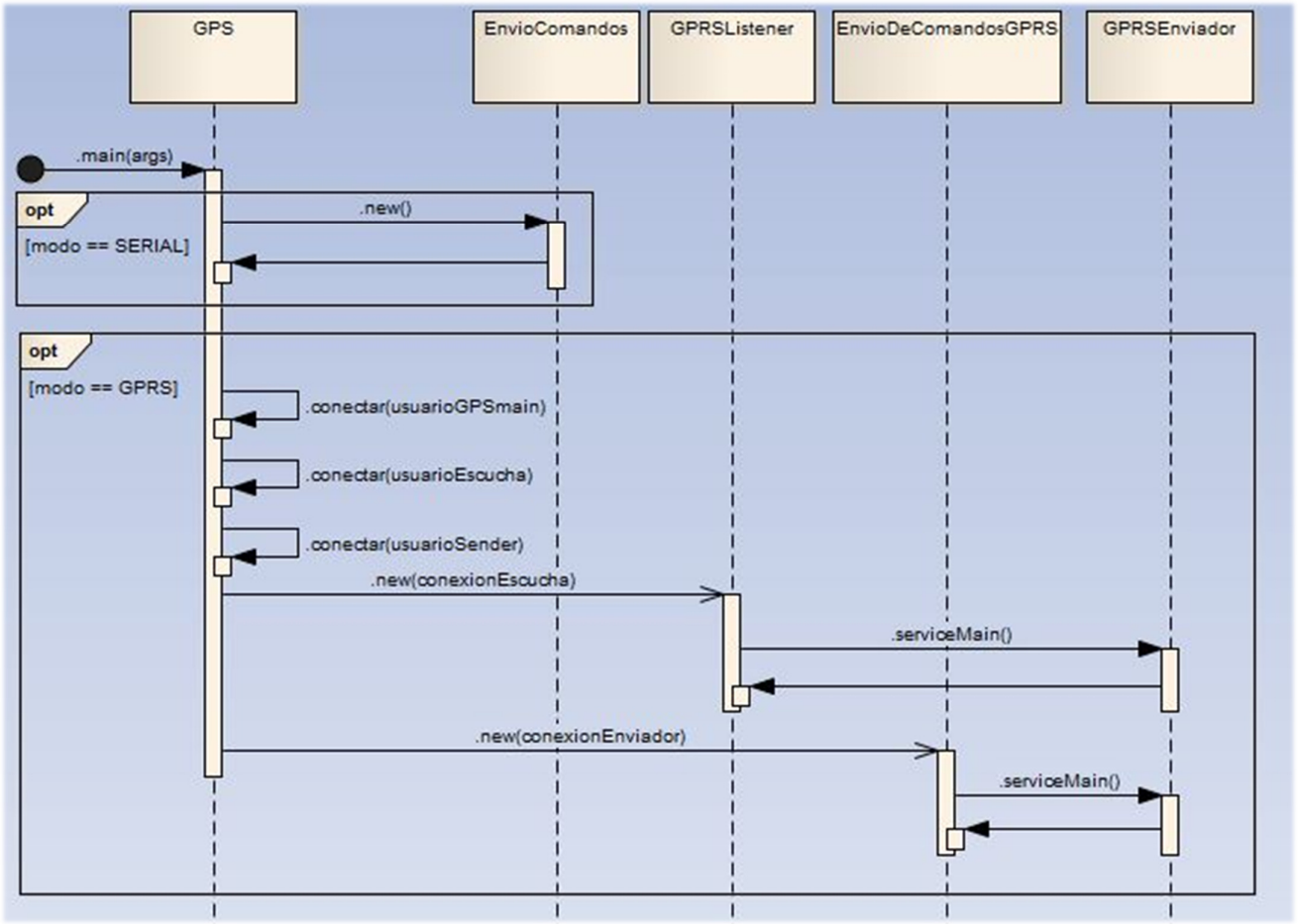
Afortunadamente, aunque la conversión desde C a Java (que es el lenguaje en el cual se había diseñado el software de comunicación) requirió su debido tiempo, Java "desciende" de C y por lo tanto muchas operaciones llevan una sintaxis similar (como por ejemplo el operador XOR - ^). Así quedó el código final mediante el cual se calcula el Check SUM de los mensajes:

```
int chksum = 0;

for (int r = 0; r < entrada.length(); r++) {

   char c = entrada.charAt(r);

   int decVal = (int) c;

   chksum = chksum ^ decVal;

}

String checksum = Integer.toHexString(chksum);

checkSUM = checkSUM.toUpperCase();
```

Cabe mencionar que en este último caso, la variable *entrada* corresponde al mensaje a enviar tomando en cuenta hasta el punto y coma que sigue al número de mensaje.

Gestión de usuarios de la BD

Con todos los riesgos anteriores eliminados, me encontré ya en situación de realizar la implementación completa de este primer prototipo del sistema. Como se declaró desde el comienzo, el software Java es multihebra y además existe la interacción con la aplicación GeneXus. Para un mejor entendimiento sobre la asignación de usuario de la BD, agrego el siguiente diagrama de secuencia que muestra la comunicación entre las clases del componente Java al inicio de su ejecución:


GPS
EnvioComandos
GPRSListener
EnvioDeComandosGPRS
GPRSEnviador
.main(args)
opt
[modo == SERIAL]
.new()
opt
[modo == GPRS]
.conectar(usuarioGPSmain)
.conectar(usuarioEscucha)
.conectar(usuarioSender)
.new(conexionEscucha)
.serviceMain()
.new(conexionEnviador)
.serviceMain()

Este diagrama muestra cómo quedó el diseño de la versión final del software, ya que en el diseño inicial, ambas hebras (la instancia perteneciente a la clase GPRSListener y la instancia perteneciente a la clase EnvioDeComandosGPRS) utilizaban el mismo usuario de BD para el acceso, lo que produjo que en un momento se bloqueara el DBMS. ***Con esto me refiero a que no era posible entrar mediante la interfaz gráfica (pg Admin III) ni tampoco ejecutar comandos desde la consola, lo cual también repercutió en la pérdida de la información que teníamos en la base de datos (datos almacenados en las tablas y relaciones entre las mismas).***

Afortunadamente, al encontrarnos en una etapa bastante temprana del proceso de desarrollo, el impacto no fue tan grave y pudimos alcanzar el estado anterior al bloqueo en dos días de trabajo intensivo, realizando back-up de la base de datos permanentemente luego de haber aprendido la lección.

Como solución provisoria, y hasta encontrar la causa de tal bloqueo, se optó por crear cuatro usuarios de BD. El primero, usuarioGPSmain, lo utiliza la hebra principal del componente Java, la cual inicia a los otros dos hilos que se ejecutan en paralelo. Los usuarios usuarioEscucha y usuarioSender se asignan a la instancia de GPRSListener y a la de EnvioDeComandosGPRS respectivamente. Y por último, el super usuario creado durante la instalación del DBMS fue asignado provisoriamente al componente GeneXus del sistema (cuestión que tendremos que modificar en el futuro asignándole un usuario estándar por norma básica de seguridad informática).

Con este despliegue hemos podido salir adelante y nos encontramos en búsqueda activa sobre la causa del conflicto sucedido, con el fin de determinar si efectivamente la caída del DBMS se produjo por los distintos procesos que utilizaban simultáneamente el mismo usuario de BD.

Generación del número de mensaje para comandos emitidos desde la base

Con respecto a los números de mensaje que se envían desde la base, como se explicó anteriormente, deben ir en formato hexadecimal y su valor varía desde el 8000 al FFFF. Esta manipulación la resolví de la siguiente forma:

- En primer lugar, si el último número de mensaje enviado desde la base (y recuperado como parámetro genérico) es el FFFF, seteo como último número de

mensaje enviado el valor 8000, de forma tal que el próximo valor a enviar se encuentre dentro del rango válido.

- En segundo lugar, se ejecuta el siguiente código para transformar el último número de mensaje enviado a formato decimal, sumar una unidad, y transformarlo nuevamente a hexadecimal:

```
Integer numeroMensaje = Integer.parseInt(ultimoNumeroDeMensaje, 16);
numeroMensaje++;
String numeroMensajeString = Integer.parseInt(ultimoNumeroDeMensaje, 16);
numeroMensajeString = numeroMensajeString.toUpperCase().trim();
```

Implementación de las funcionalidades requeridas en la aplicación GeneXus

Uno de los requerimientos no funcionales, declaraba que la aplicación principal debía ser desarrollada mediante el IDE GeneXus. Pero, ¿qué es GeneXus?

Una definición rápida nos diría que es una herramienta de desarrollo de software ágil, multiplataforma, orientada principalmente a aplicaciones web empresariales, plataformas Windows y dispositivos móviles o inteligentes. El desarrollador describe sus aplicaciones en alto nivel, de manera mayormente declarativa, a partir de lo cual GeneXus genera código para múltiples plataformas (Windows, iSeries, Web, dispositivos móviles).

El problema fundamental de trabajar con este IDE o herramienta es que resulta muy poco flexible, es decir, para desarrollar con GeneXus tenemos que limitarnos a lo que GeneXus nos brinda y no podemos alejarnos demasiado del margen. Por lo tanto, si deseamos que nuestra aplicación implemente alguna funcionalidad fuera de lo común para lo que son sistemas de gestión administrativa, debemos recurrir a los denominados User Control. Otra vez, ¿qué son los User Control?

Los **User Control** son herramientas Web que pueden desarrollarse para extender la arquitectura de **GeneXus** y que se exhiben sobre la aplicación final, enfocadas al usuario. Su virtud principal consiste en expandir aplicaciones y generar una interfaz de usuario enriquecida; más rica y atractiva.

Para eliminar muchos de los riesgos relacionados al diseño del componente GeneXus de GeoManager he acudido a la utilización de este tipo de controles. Entre ellos:

- JSCook Menu: utilizado para realizar el CU Administrar usuarios. Consiste en una barra de menú muy intuitiva y amigable al usuario, mediante la cual se puede filtrar el acceso a determinadas vistas del sistema dependiendo del tipo de usuario que se encuentre logueado (en mi caso, Administrador o Usuario).
- Map Control: permite embeber un mapa dentro de la aplicación, utilizando Google, Baidu o Yahoo como proveedor de información. Con este mapa fue posible mostrar en tiempo real la ubicación de cada móvil, además de contar con la posibilidad de poder agregarle cualquier información relacionada que deseemos, como ser por ejemplo la velocidad actual, la patente, una foto, etc.
- Timer: este control sirve para disparar un evento cada cierto intervalo de tiempo el cual es definido por el usuario o el programador. De esta forma, pude permitir que el mapa se refresque periódicamente mostrando la información actualizada de los vehículos de la empresa, sin la necesidad de que el usuario sea el que deba presionar un botón o generar un evento en particular para la actualización o refresco de la pantalla.

Energización del VL para pruebas en ciudad

Uno de los objetivos planteados desde el comienzo fue la posibilidad de contar con algún medio que me permitiera en esta etapa inicial de pruebas independizarme de la batería del automóvil cuando se moviera al VL dentro de la ciudad.

Luego de consultas al soporte de Virtec, me aconsejaron utilizar una “ficha macho para el encendedor del auto”. Este dispositivo consiste en un adaptador que se conecta al encendedor del auto y mediante el cual obtenemos 12 V desde la batería del vehículo.

Este valor se encuentra dentro del rango de tensión de trabajo del VL, y por lo tanto me permitió realizar las pruebas necesarias en ciudad sin ningún tipo de problemas, las cuales requirieron entre uno y dos días de trabajo fuera de la empresa por cada prueba individual.

Continuidad en la comunicación por GPRS

Por último, este riesgo fue identificado cuando el prototipo se encontraba en marcha, y es hasta el momento un tema que está bajo revisión y puede poner en juicio el lanzamiento del producto GeoManager.

Se registró que en determinadas franjas horarias, aun probando con distintos APN, el VL no logra hacer llegar a destino (base) los reportes y respuestas a consultas o seteos.

El estudio se realizó utilizando el analizador de protocolos Wireshark, mediante el cual, y en conjunto con el debbuger incorporado en el software XVM Terminal, pude detectar que los paquetes enviados desde el servidor se emitían correctamente, pero nunca

llegaban al VL. Lo mismo pasaba con los reportes periódicos enviados desde el dispositivo: el VL los enviaba, pero éstos nunca llegaban a destino[4].

Notado este inconveniente, supuse que quizás el APN recomendado por Virtec (datos.personal.com) se encontraba sobrecargado en tales franjas horarias (normalmente de 10:00 am a 11:00 am, y de 12:00 am a 3:30 pm), y por lo cual decidí probar con otros:

- gprs.personal.com
- internet.personal.com
- ba.amx
- internet.unifon

Luego de esta acción, llegué a la conclusión de que el problema no era inherente al APN y decidí consultar con el soporte. Virtec me respondió que efectivamente, en ciertas ocasiones, el VL no logra conectarse a la red, y me propusieron dos soluciones alternativas:

1. Mediante programación XVM, configurar que el VL admita un mayor número de reintentos de envío de reportes antes de reiniciar el módem (solución menos aconsejable desde el soporte de Virtec). Es decir, el VL envía sus reportes periódicos a cierta frecuencia de tiempo, si no recibe un mensaje ACK por parte de la base vuelve a enviarlo "suponiendo" que no se ha recibido, y así sucesivamente una determinada cantidad de veces (3 por defecto), hasta que luego reinicia su módem lo cual conlleva una considerable cantidad de tiempo. Me propusieron esta alternativa como una de las dos opciones viables, pero la menos recomendable debido a que, si bien me permite un mayor intento de envíos de mensajes ACK desde la base con el fin de una posible comunicación, el VL puede quedar en un estado bloqueado, en espera de una respuesta y sin reiniciar su modem, y la única forma de volvernos a comunicar con él sería extrayéndolo del móvil y desbloqueándolo mediante conexión Serial, lo cual significaría un impacto tremendo para el negocio y Legado IT no puede arriesgarse de esa forma.
2. La segunda alternativa, es dejar el sistema como se encuentra actualmente, ya que una vez reiniciado el modem, en la mayoría de las ocasiones consigue comunicarse, aunque luego se pierda nuevamente la señal. Esta opción no me asegura que haya

[4] Para el mencionado estudio se aplicó el filtro udp.port == 4097 en Wireshark con el fin de evitar cualquier información irrelevante que pudiera entorpecer el aislamiento del problema.

una comunicación fluida entre el VL y la base pero, debido a que este problema se da solo en ciertas ocasiones, es una tolerancia que sí o sí debemos estar dispuestos a aceptar frente a la otra alternativa, que en el peor de los casos resultaría caótica.

Como conclusión, no hay solución inmediata por parte de la empresa proveedora del hardware. Aseguran que todos sus clientes actuales (incluyendo a grande compañías como Nuevo Central Argentino, Fenabus Chile, Pan American Energy, entre otras) tienen la misma problemática y luego del primer o segundo reinicio del modem, el VL sincroniza nuevamente, lo cual impacta en una ventana de dos minutos aproximadamente en la cual no se tiene información actualizada sobre el móvil.

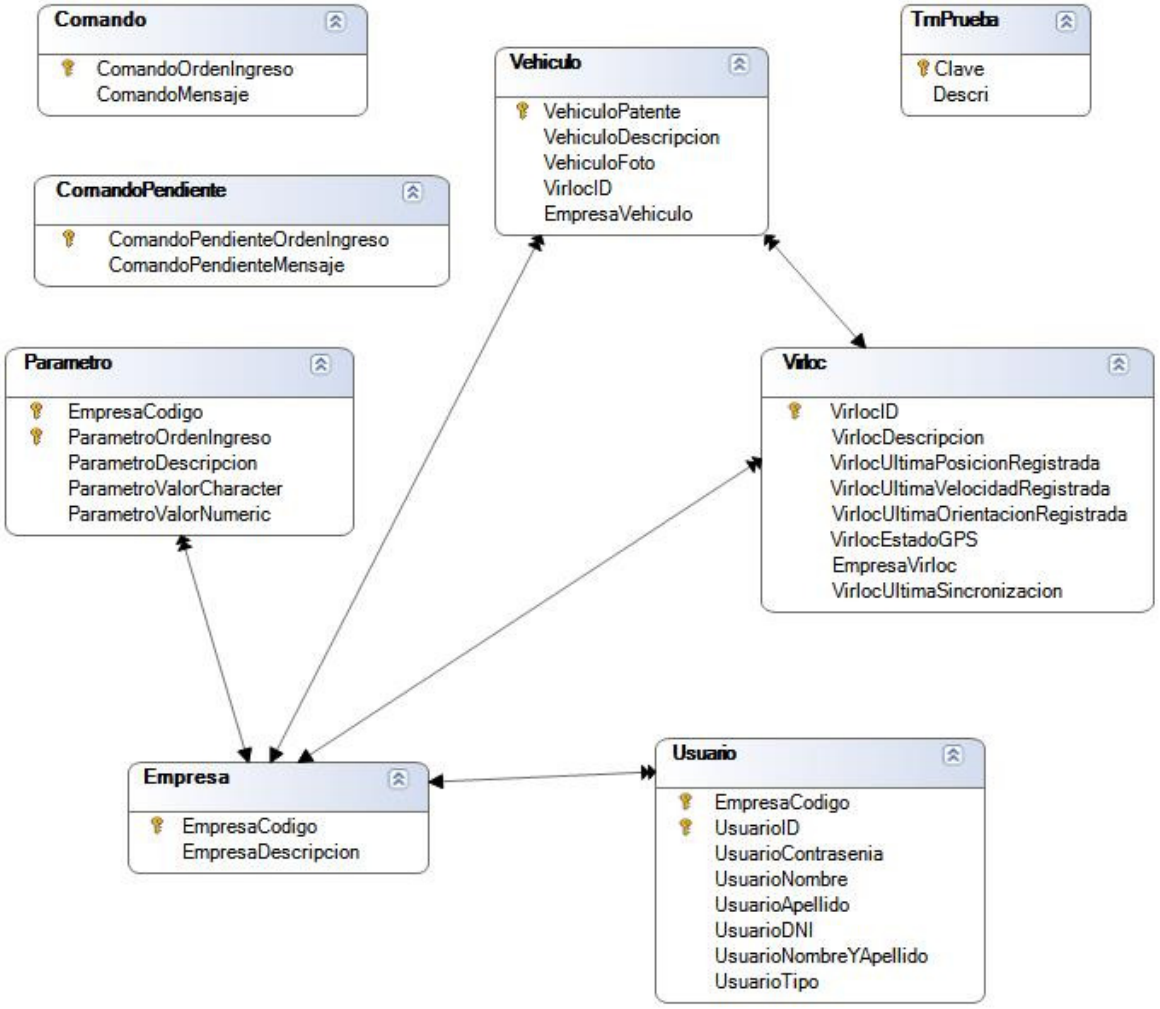
Informada esta problemática al dueño de Legado IT y actual gerente de Federada SALUD, éste determinó exponer el caso ante el directorio de la mutual y que allí se evalúe el destino del proyecto.

Diagramas globales

Diagrama GeneXus

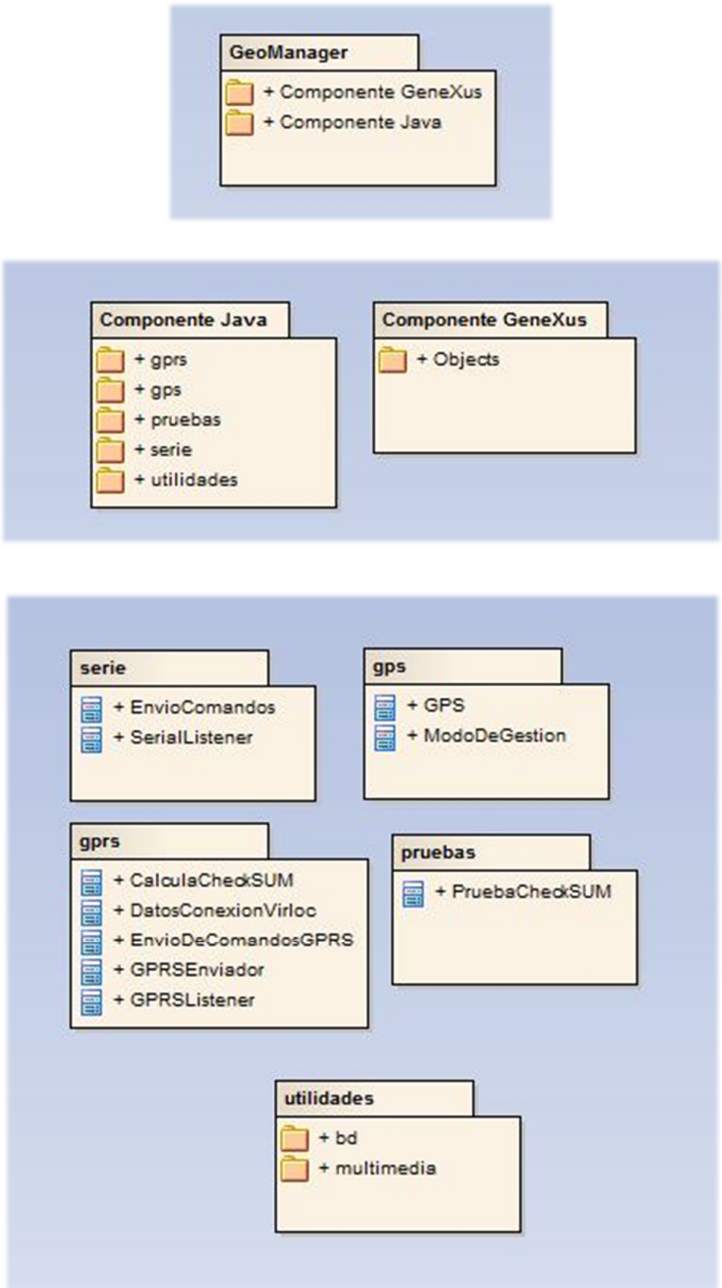
Dentro del IDE GeneXus existen objetos de tipo Diagrama, mediante los cuales podemos observar las relaciones entre las distintas tablas (de la BD de la aplicación) o Transacciones del sistema. En vocabulario GeneXus llamamos objetos Transacción a algo muy parecido a lo que son la clases dentro del paradigma de orientación a objetos, por lo cual, mediante el objeto Diagrama tendríamos algo similar a un diagrama de clases del sistema o DER.

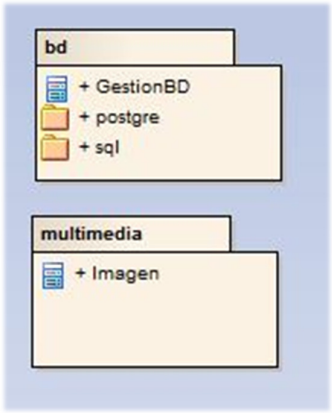
La siguiente figura es un objeto Diagrama de la KB (así llamamos a la aplicación en vocabulario GeneXus) que muestra las tablas que componen la BD, ya que si mostrara los objetos Transacción podría llevar a confusión sobre las relaciones existentes.

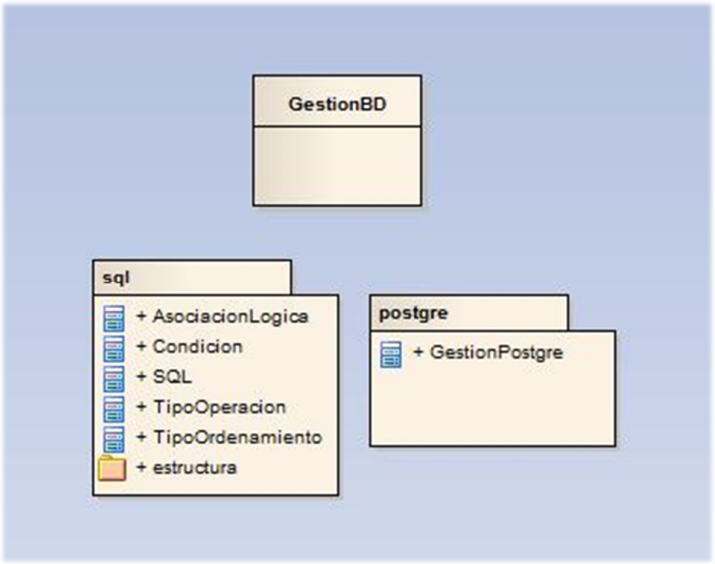

Comando
ComandoOrdenIngreso
ComandoMensaje
Vehiculo
VehiculoPatente
VehiculoDescripcion
VehiculoFoto
VirlocID
EmpresaVehiculo
TmPrueba
Clave
Descri
ComandoPendiente
ComandoPendienteOrdenIngreso
ComandoPendienteMensaje
Parametro
EmpresaCodigo
ParametroOrdenIngreso
ParametroDescripcion
ParametroValorCharacter
ParametroValorNumeric
Virloc
VirlocID
VirlocDescripcion
VirlocUltimaPosicionRegistrada
VirlocUltimaVelocidadRegistrada
VirlocUltimaOrientacionRegistrada
VirlocEstadoGPS
EmpresaVirloc
VirlocUltimaSincronizacion
Empresa
EmpresaCodigo
EmpresaDescripcion
Usuario
EmpresaCodigo
UsuarioID
UsuarioContrasenia
UsuarioNombre
UsuarioApellido
UsuarioDNI
UsuarioNombreYApellido
UsuarioTipo

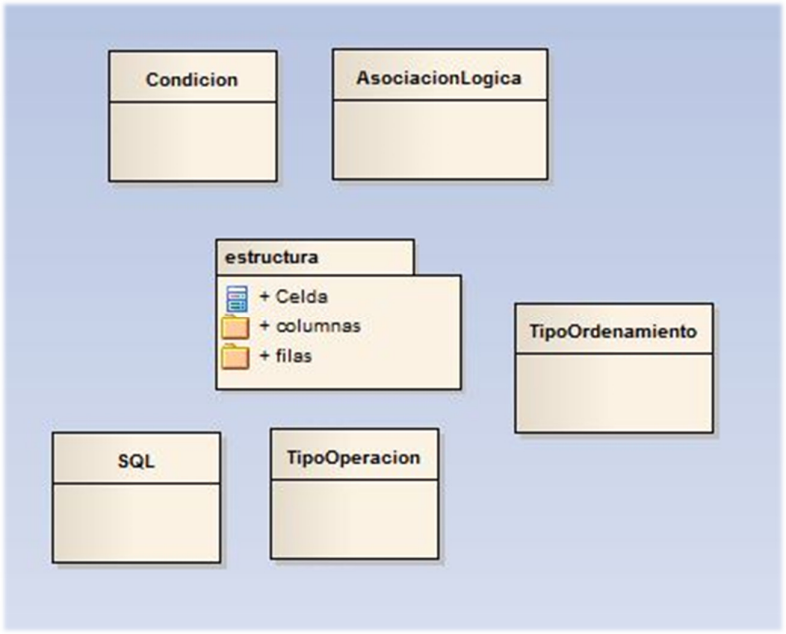
Como se puede observar, el sistema es multiempresas, y en la tabla *Empresa* se guardarían los datos de los potenciales clientes que utilicen GeoManager. Mediante el atributo EmpresaCodigo (numérico de 4 dígitos) se filtran todas las vistas (o pantallas) para mostrar únicamente los objetos correspondientes.

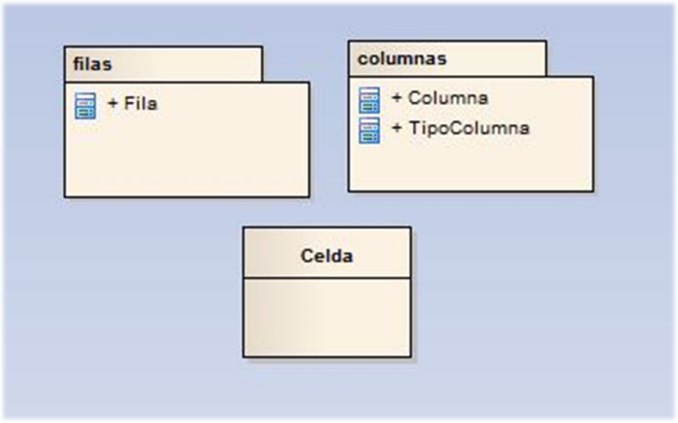
Diagrama de paquetes



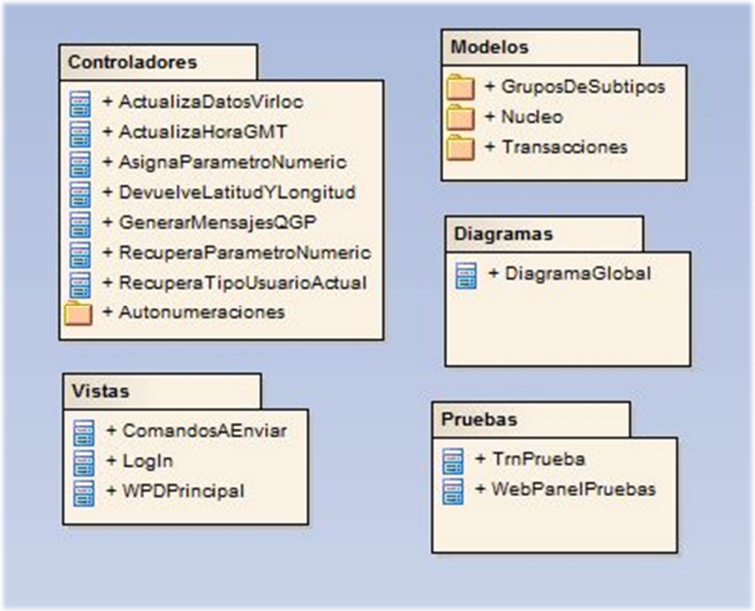

bd
+ GestionBD
+ postgre
+ sql
multimedia
+ Imagen

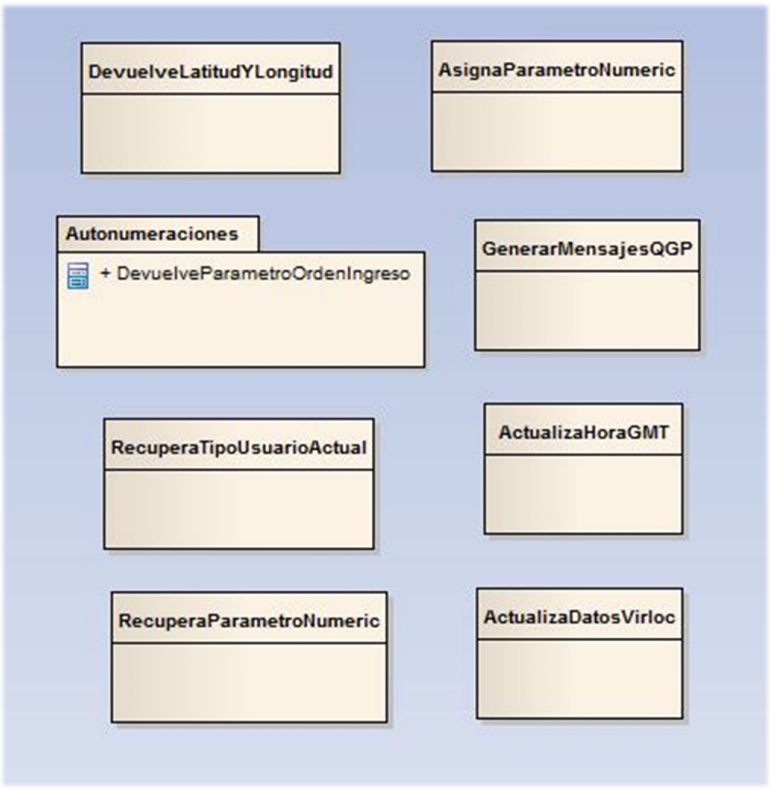

GestionBD
sql
+ AsociacionLogica
+ Condicion
+ SQL
+ TipoOperacion
+ TipoOrdenamiento
+ estructura
postgre
+ GestionPostgre

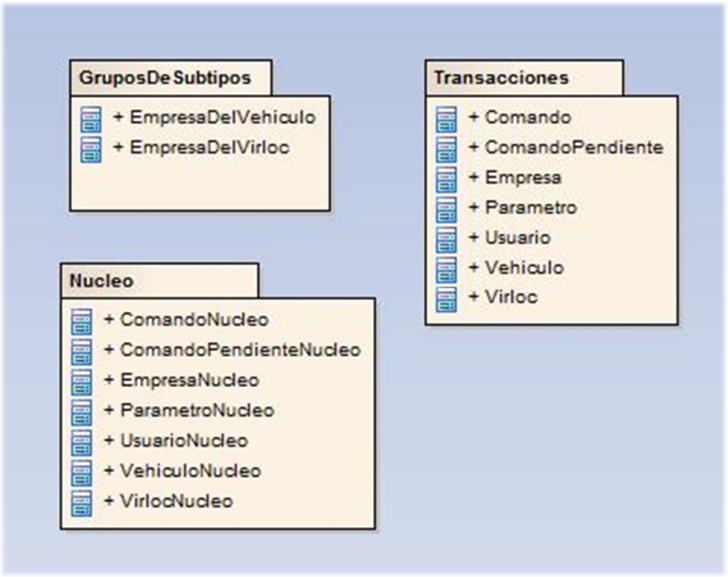

Condicion
AsociacionLogica
estructura
+ Celda
+ columnas
+ filas
TipoOrdenamiento
SQL
TipoOperacion


filas
+ Fila
columnas
+ Columna
+ TipoColumna
Celda


Objects
+ Controladores
+ Diagramas
+ Modelos
+ Pruebas
+ Vistas


Controladores
+ ActualizaDatosVirloc
+ ActualizaHoraGMT
+ AsignaParametroNumeric
+ DevuelveLatitudYLongitud
+ GenerarMensajesQGP
+ RecuperaParametroNumeric
+ RecuperaTipoUsuarioActual
+ Autonumeraciones
Modelos
+ GruposDeSubtipos
+ Nucleo
+ Transacciones
Diagramas
+ DiagramaGlobal
Vistas
+ ComandosAEnviar
+ LogIn
+ WPDPrincipal
Pruebas
+ TrnPrueba
+ WebPanelPruebas





Como se puede observar, dentro del paquete *Modelos* se ha creado un paquete de orden inferior llamado *Nucleo*. Allí coloqué una copia exacta de todas las transacciones que conforman la parte GeneXus de la aplicación. Esto se realizó con el fin de que si en el futuro se desea migrar la aplicación desde el cliente actual (Federada SALUD) hacia uno

nuevo, basta con exportar dicho paquete mediante un proceso propio del IDE GeneXus e importarlo en la KB de nuestro nuevo cliente. Con solo realizar esta operación de exportación/importación, en el nuevo cliente ya contaríamos con todas las relaciones y atributos de las distintas tablas de la BD, lo cual representa la mayor parte de la estructura de GeoManager.

Diagrama de actividades