Universidad de Buenos Aires
Facultad de
Ciencias Exactas y Naturales
Departamento de Computación

Ingeniería del Software 2

Segundo Cuatrimestre de 2013

Trabajo práctico

Corre por tu vida

Fecha de entrega: 7 de Octubre

Integrante	LU	Correo electrónico
Cammi, Martín	676/02	martincammi@gmail.com
De Sousa Bispo, Mariano	389/08	marian_sabianaa@hotmail.com
Felisatti, Ana	335/10	anafelisatti@gmail.com
Raffo, Diego	423/08	enanodr@hotmail.com

Índice

1.Introducción

El desarrollo de la aplicación “Corre por tu vida” fue realizado inspirándose en la idea de un entrenador ofreciendo sus servicios en 1920. En ese entonces, quien contratase al entrenador le daría sus datos personales y le presentaría un objetivo que quisiera cumplir y por el cual deseara entrenarse. El entrenador entonces armaría un plan para llevar a cabo los entrenamientos y luego se pondría a disposición del corredor para comenzarlos.

Al realizar un entrenamiento, el entrenador iría tomando nota del rendimiento del corredor y dándole información sobre el mismo, por ejemplo, indicándole su velocidad, la distancia que ha recorrido y la etapa de entrenamiento en la que se encuentra, y según esta si debe o no intensificar el ritmo. También le indicaría dónde se está realizando el entrenamiento.

Cuando el entrenamiento hubiera sido completado o el corredor decidiera cancelarlo, el entrenador utilizaría todos esas notas para armarle al corredor un informe y mostrarle cómo fue su rendimiento; más aún, el entrenador acumularía la información de cada entrenamiento para generar estadísticas de dicho rendimiento a lo largo del tiempo.

Nuestra aplicación modelará entonces un entrenador que permite al corredor contar con varios planes, cada uno con varios entrenamientos y que al realizarlos nos dará el información en tiempo real, una devolución al final y permitirá ver las estadísticas de las corridas.

2.Diseño

Considerando la analogía de la introducción, un entrenador es la persona que contratamos para llevar a cabo los objetivos que nos propusimos. Se pueden distinguir dos roles muy distintos dentro de este entrenador que se decidieron modelar separadamente:

- El Deportólogo (**Sports Doctor**).
- El Entrenador personal (**Trainer**).

El **SportsDoctor** tiene como única responsabilidad la de crear entrenamientos personalizados en base a las características del corredor (**RunnerProfile**), el objetivo propuesto (**RunnerObjective**), su disponibilidad (**RunnerAvailability**) y su estado físico (**RunnerState**).

El **Trainer** tiene como responsabilidad la de llevar a cabo el entrenamiento brindando información del mismo al corredor en todo momento. Posee como colaborador interno a un **SportsDoctor**, en el cual delega la creación de entrenamientos (**Training**).

Como sólo se quiere que haya un Trainer en la aplicación y que sea conocido, usamos el patrón Singleton para modelar este comportamiento.

El **Trainer** también mantiene un listado de planes (**Plan**) y un navegador (**Navigator**) que le brinda la posición, velocidad y distancia recorrida, y que permite actualizar la ubicación para poder informarla durante el **Training**. El **Trainer** también se encarga de generar una devolución al finalizar el **Training**. Como el navegador tendrá sentido sólo cuando el entrenador esté brindando información sobre el entrenamiento, utilizamos para el navegador el patrón State, que nos permite modelar cuando se está entrenando y se está usando el **Navigator** de cuando no, (**NavigatorState, ActiveNavigator, StoppedNavigator**)

Un **Plan** tiene un nombre y los entrenamientos que posee. Un **Training** tiene su nombre, el plan al que pertenecen y las fases (**Phase**) que lo componen, Las **Phase** a su vez consisten en un periodo de tiempo (**TimeLapse**) y una velocidad máxima y mínima (**Velocity**) para establecer dentro de qué márgenes se debe correr.

Continuando el razonamiento con la analogía, un entrenador debería de poseer dos temporizadores para saber cuando cambia la fase de un entrenamiento y cuando actualizar los datos de velocidad, posición, etc. Esto en nuestro modelo se traduce en que Trainer tiene un colaborador **TimeKeeper** a quien se puede suscribir para recibir alertas de cambio de fase en un tiempo establecido. El **Navigator** además envía notificaciones de cambio de velocidad y posición al entrenador. Para lograr este comportamiento se aplicó el patrón Observer y se necesitó distinguir cuál era cada aviso, la implementación convergió en un Adapter para cada tipo de mensaje.

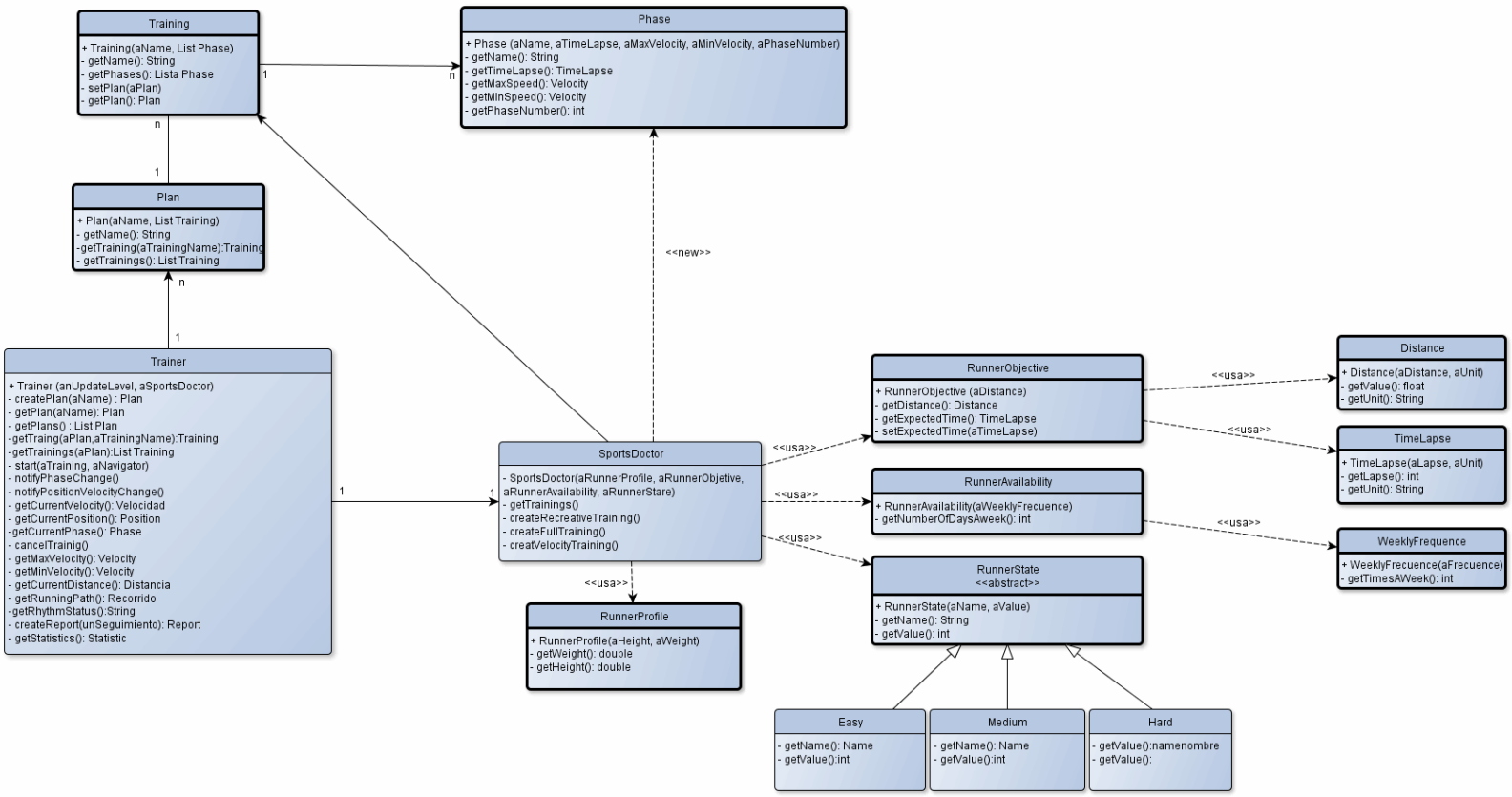
El valor que determina cada cuanto actualizar los datos y el modo de aviso que se debe utilizar en el cambio de fase se obtienen del nivel de actualización (**UpdateLevel**), colaborador interno del Trainer. Esto sería el nivel de consumo de bateria, considerando que un nivel de actualización bajo involucra un consumo bajo.

Las devoluciones (**Report**) representan el informe que nos daría un entrenador en el momento que terminamos de correr con los datos del entrenamiento del día, que son la velocidad máxima, distancia recorrida y el recorrido realizado. La devolución puede ocurrir cuando finaliza el entrenamiento, o cuando la persona que esta entrenando y decide cancelar el entrenamiento, en cuyo caso la devolución se hace con los datos parciales del entrenamiento.

Las estadísticas (**Statistics**) recopilan todas las devoluciones de entrenamientos finalizados, es decir, no contempla devoluciones parciales.

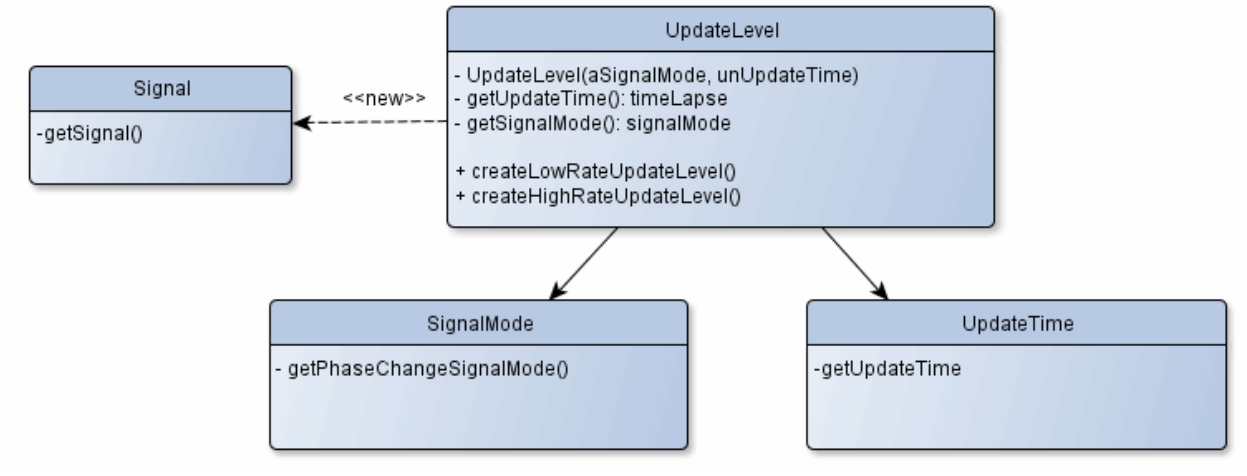
2.1 Diagramas

A continuación presentamos el diagrama de clases por secciones y algunos diagramas de secuencia de casos interesantes.

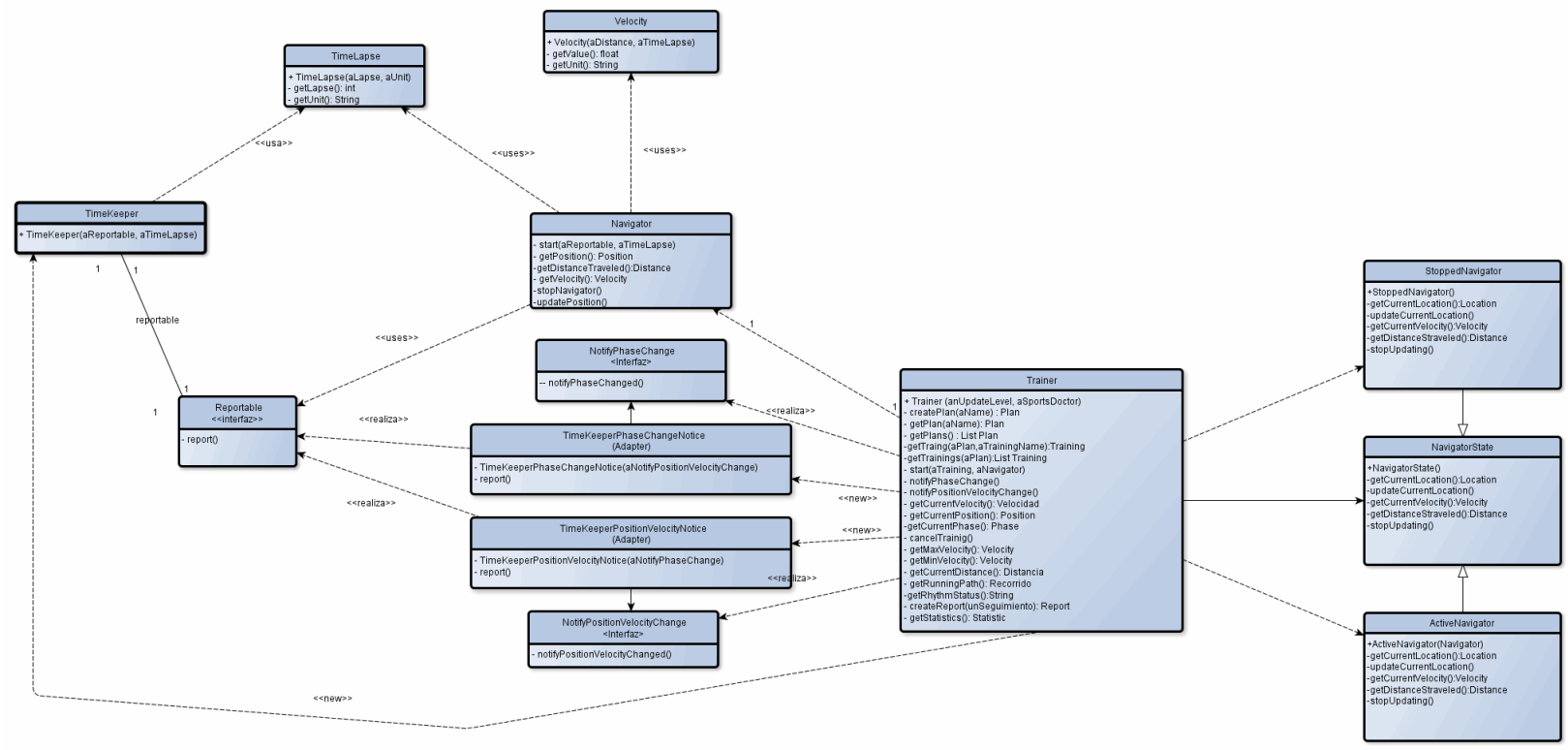


La clase Trainer se instancia en un único objeto Trainer. Como ya se explicó anteriormente, este objeto es responsable de llevar a cabo los entrenamientos, pero delega en el SportsDoctor la creación de los planes con sus entrenamientos y sus fases.

En el diagrama se pueden ver las relaciones y dependencias de la clase SportsDoctor.

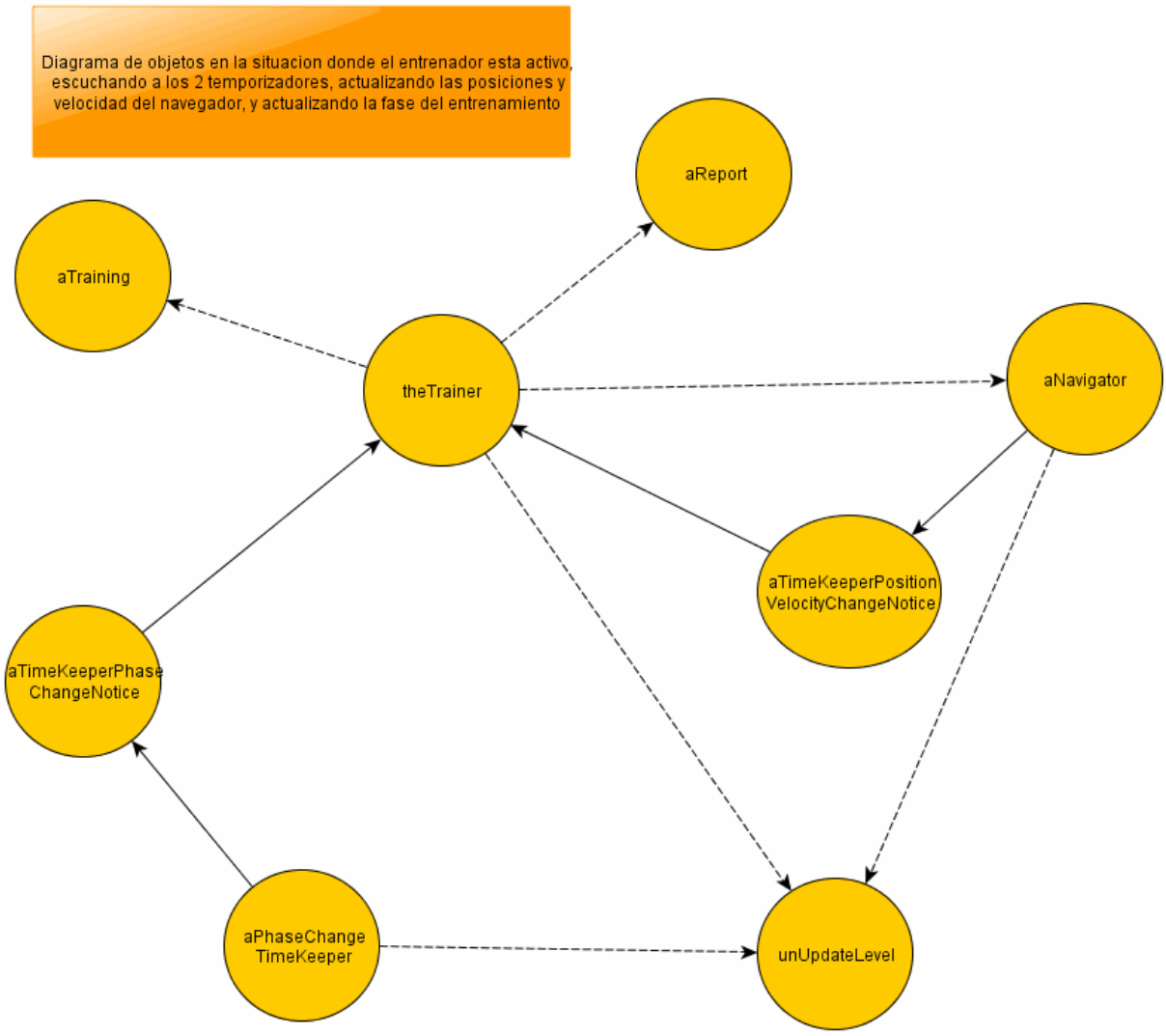


En este diagrama se muestra la clase UpdateLevel y sus colaboradores internos. El objeto Trainer usa (<<uses>>) la clase UpdateLevel



En esta última sección del diagrama de clases podemos ver la parte del entrenamiento del sistema. Los TimeKeepers y sus Adapters, Los NavigatorState y el Navigator

A continuación se explicará más detalladamente el funcionamiento del sistema con diagramas de colaboraciones y de secuencia sobre algunas situaciones que se consideraron interesantes

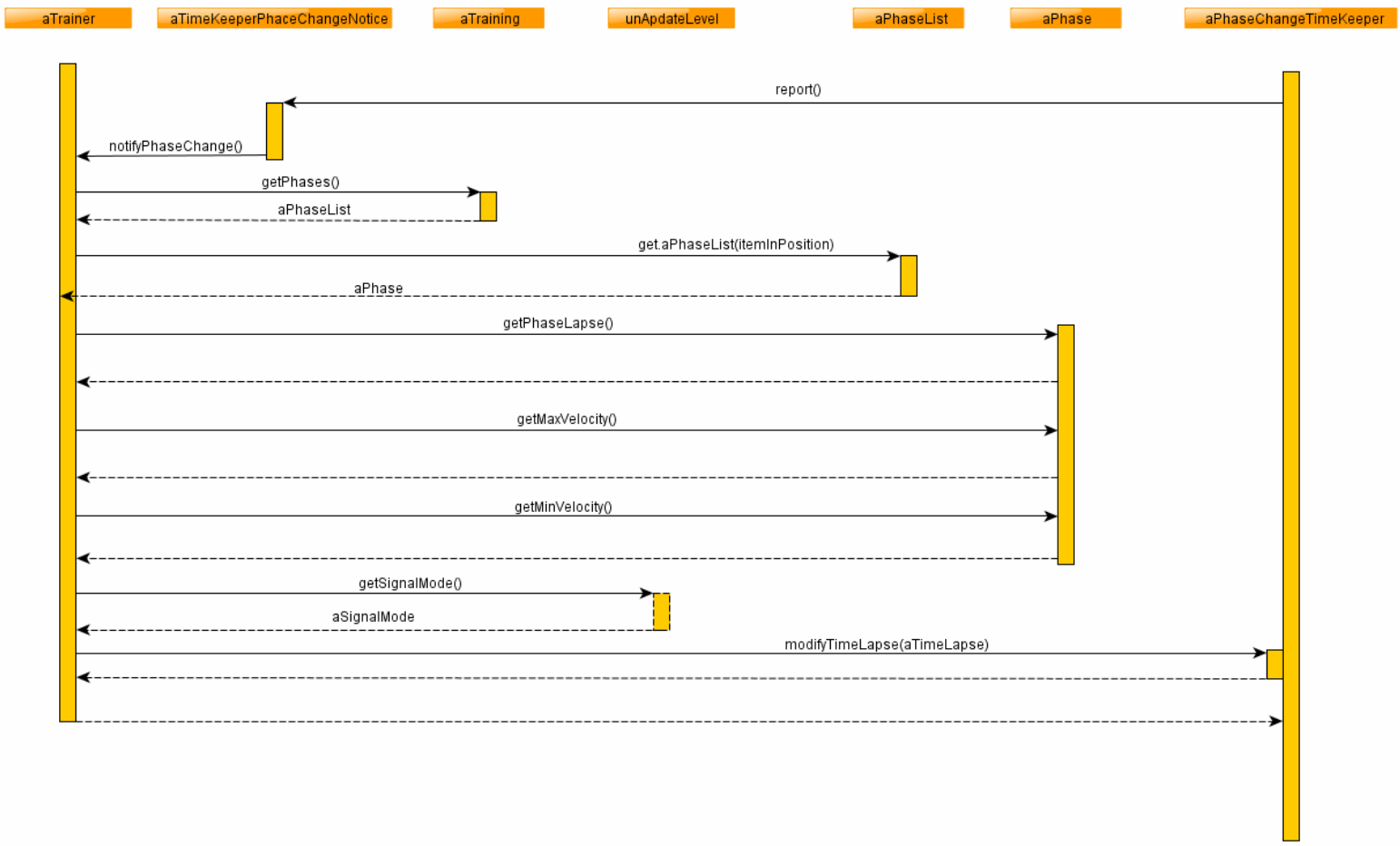


Como se ve en la imagen, este diagrama modela las colaboraciones del objeto theTrainer cuando se esta llevando a cabo un entrenamiento, y posteriormente arma un report.

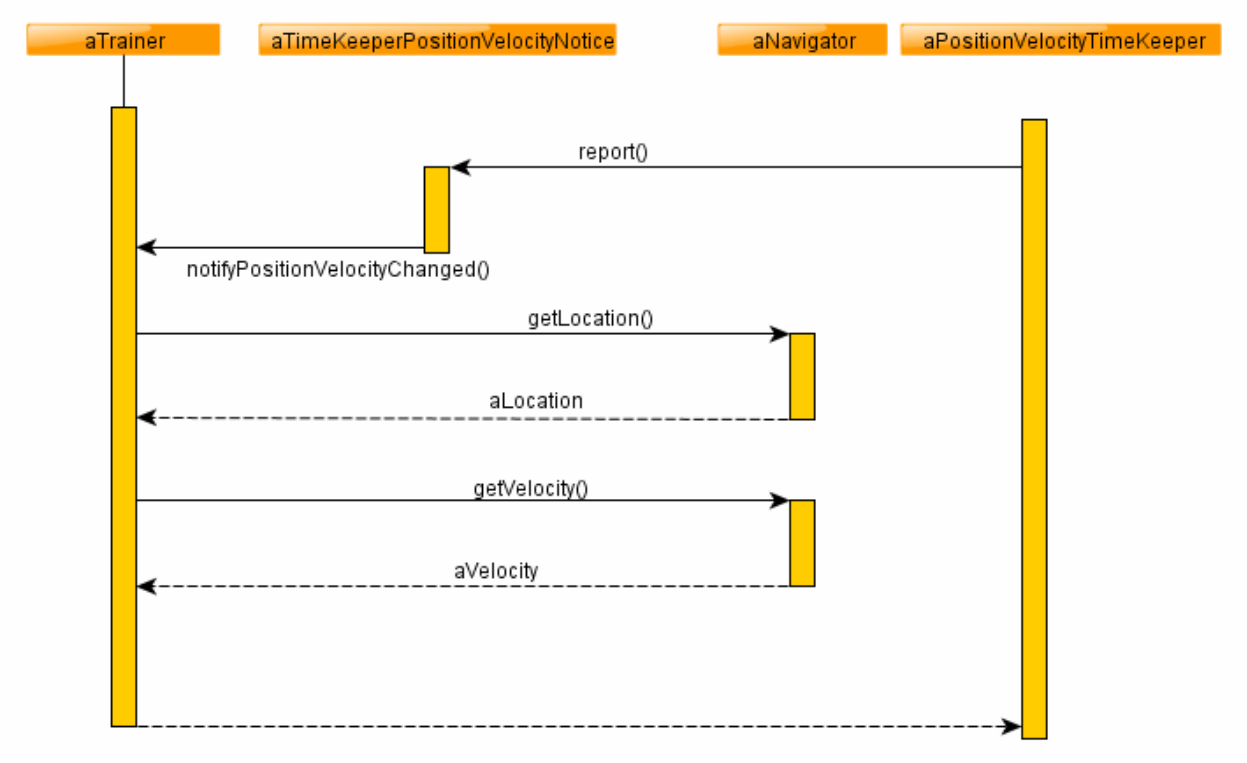
Conoce aNavigator ya que se lo envían junto con el los objetos unUpdateLevel y aTraining para comenzar el entrenamiento. Luego colabora como veremos en detalle en los diagramas de secuencia, con los objetos aTimeKeeperPhaseChangeNotice y aTimeKeeperPositionVelocityChangeNotice que son adaptadores por los cuales los timeKeepers pueden notificarle que cambió la fase del entrenamiento, o que debe actualizar la

velocidad y la posición actual. Luego de finalizarse el entrenamiento, theTrainer genera un report.

Los siguientes dos diagramas de secuencia muestran cómo interactúa theTrainer con los temporizadores para actualizar, en el primer diagrama la Fase del entrenamiento, en el segundo la posición y velocidad del navegador.

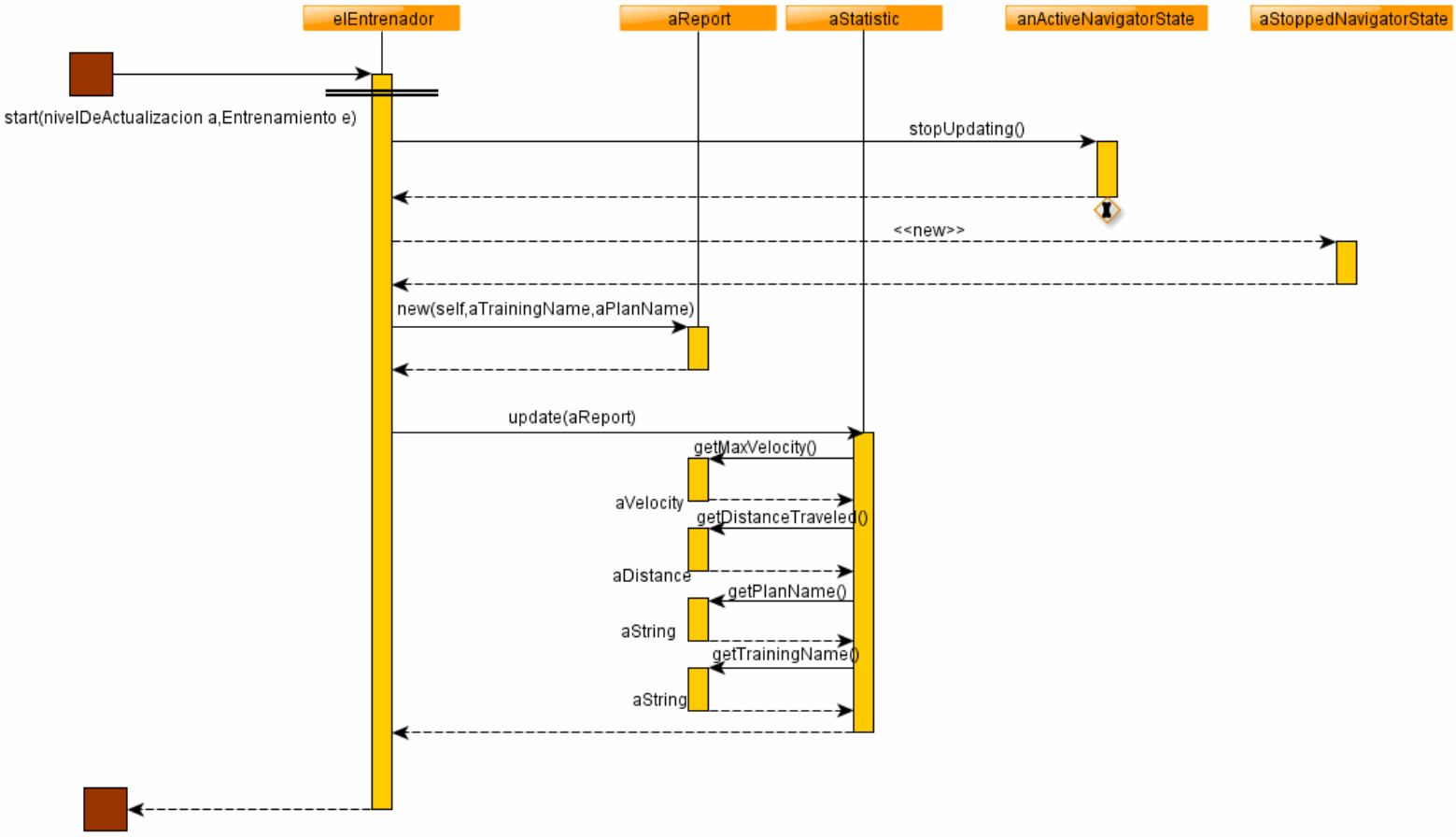


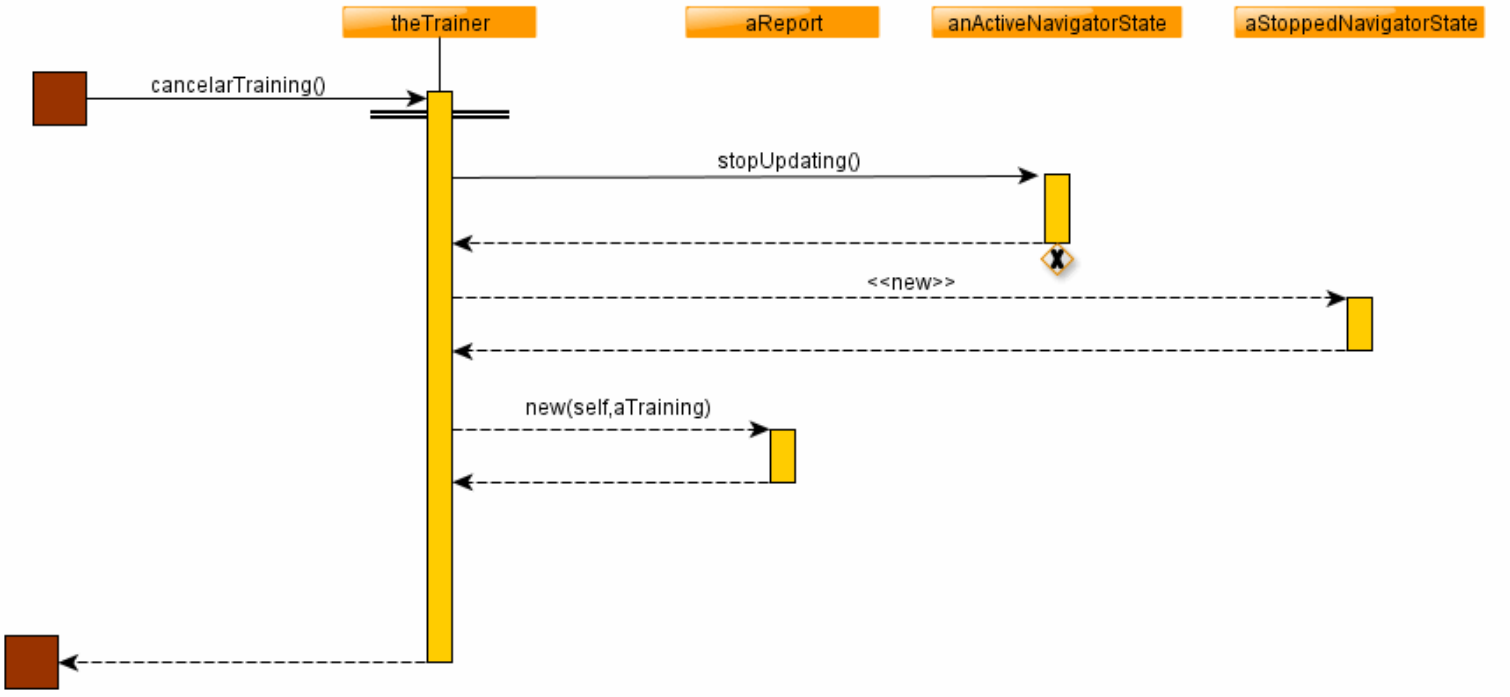
Podemos observar la secuencia de colaboraciónes, y los objetos participantes en el proceso que inicia aPhaseChangeTimeKeeper cuando envía la notificación al adapter, que luego enviará el mensaje de notifyPhaseChange a theTrainer, quien luego se encargará de conseguir todo lo necesario para cambiar de fase



En esta secuencia podemos observar con detalle como aTrainer interactúa con el navegador para actualizar la posición y velocidad. El timeKeeper funciona de manera análoga en el diseño.

Los siguientes dos diagramas muestran el comportamiento de lo que hace theTrainer cuando finaliza un entrenamiento, teniendo el segundo la particularidad de que termina debido a que el usuario decide interrumpir el entrenamiento antes de que este finalice.


elEntrenador
aReport
aStatistic
anActiveNavigatorState
aStoppedNavigatorState
start(nivelDeActualizacion a,Entrenamiento e)
stopUpdating()
<<new>>
new(self,aTrainingName,aPlanName)
update(aReport)
getMaxVelocity()
aVelocity
getDistanceTraveled()
aDistance
getPlanName()
aString
getTrainingName()
aString


theTrainer
aReport
anActiveNavigatorState
aStoppedNavigatorState
cancelarTraining()
stopUpdating()
<<new>>
new(self,aTraining)

Se puede apreciar que los comportamientos son muy similares, con la salvedad de que cuando el usuario decide interrumpir el entrenamiento no se utiliza la devolución generada para actualizar las estadísticas generales.

3. Análisis

Del diseño presentado podemos analizar varios aspectos interesantes. Por ejemplo, la extensibilidad. El proyecto fue presentado con algunos requerimientos de este tipo, siendo estos los tipos de entrenamiento y nivel de consumo de batería.

Se consideró que la esencia de cualquier entrenamiento son las fases que contiene. La posibilidad de generar nuevos entrenamientos estará dada en la forma que se crean las fases que en su conjunto formarán un entrenamiento, responsabilidad que recae únicamente en el objeto SportsDoctor.

Para lograr extensibilidad en cuanto a los niveles de consumo de batería, se consideró que la esencia de estos niveles es el “UpdateLevel”, que incluye la frecuencia con la que actualizará los datos del navegador, y el modo en que lo notificará. Agregar nuevos niveles de consumo se traduce en generar otro(s) UpdateLevel con distinto colaborar Tiempo (pues este marca cada cuanto se actualizará), y de ser necesario extender la clase “Modo Aviso”.

Estas dos características hacen que nuestro diseño sea, a nuestro criterio, resistente a los cambios que prevemos para las siguientes versiones de la aplicación y cambios que puedan querer realizarse para su optimización.

En cuanto a la declaratividad, al contar con un razonamiento análogo de cómo resolver el mismo problema de una persona que quiere entrenar para alcanzar un determinado objetivo en una época donde no existían los dispositivos electrónicos, el diseño del sistema resulta muy sencillo de comprender, ya que los nombre de los objetos y los mensajes que responden son de un dominio conocido.

4. Implementación

4.1 Diferencias entre el modelo y la implementación

Al momento de implementar el modelo se encontraron algunos detalles de implementación que tuvieron que ser adaptados:

- Si bien el modelo propuesto utiliza un **TimeKeeper** para contar el tiempo, éste tuvo que basarse en el **Timer** de Java que realmente es el que controla el tiempo.
- Con el **Navigator** sucede lo mismo, pues utiliza los servicios de Google que son los que realmente obtienen la posición real. Además, el navegador sólo se utiliza cuando el Trainer está realizando el s seguimiento de una corrida, en otro caso el Navegador no será utlizado. Para modelar esto utilizamos un **NavigatorState** que contiene un **Navigator** que a su vez engloba a los servicios de Google Maps (véase **GoogleMapsServices**).
- Para utilizar un modelo cercano a MVC se agregaron clases **Controller** para interactuar con las **Activity** y permitir un claro manejo del modelo y la vista .

4.2 Patrones que emergieron del modelo

Diferentes patrones de diseño fueron surgiendo del diseño a medida que avanzaba el modelado y posteriormente el desarrollo. Los patrones identificados fueron los siguientes:

- Observer (Aviso del Timer y Navigator al Trainer) : Utilizado para que el Trainer observe a los observables TimeKeeper y Navigator. Como para las notificaciones de cada uno el Trainer debía realizar acciones diferentes, el trainer tuvo que implementar diferentes interfaces para poder identificar de donde llegaba cada notificación. Es por ello que se requirió "adaptar" esto.

- Adapter (TimeKeeperPhaseChangeNotice, TimeKeeperPositionVelocityNotice): Se utilizaron como adaptadores para los Observables TimeKeeper y Navigator para poder diferenciar sus respectivas notificaciones.

- State (NavigatorState): Utilizado para manejar el estado del Navegador y cambiarlo según el Entrenador se encuentra creando planes o realizando un seguimiento de entrenamiento.
- Singleton (Trainer): Usado para modelar al Entrenador que debe ser único y bien conocido.

4.3 Alternativas planteadas y finalmente rechazadas

Consideramos tener un objeto Runner que representará al corredor pero esto fue eventualmente descartado pues sólo era un contenedor de las características personales y estado físico y no representaba un ente concreto o razonable de la realidad.

Por otro lado, pensamos en separar del Trainer la administración de planes y el seguimiento de un entrenamiento. Sin embargo, esto sólo hubiera resultado en un handler intermedio que mantuviera los planes, el cual no agregaba valor ni funcionalidad. De hecho hacía más complejo el diseño.

Sobre los entrenamientos, en un principio pensamos en hacer una jerarquía de tipos para ellos. Se descartó esto porque los entrenamientos no tienen diferencias esenciales, simplemente varían en sus valores. Es decir, lo que cambia es cómo se utilizan los datos del corredor para crear las fases. Esto a nuestro parecer sólo significa que el SportsDoctor tenga distintos métodos de creación de entrenamiento en los cuales utiliza esos datos de manera distinta, obteniendo entrenamientos diferentes.

Tuvimos una idea similar para los niveles de consumo (o “actualización” como los nombramos) pensando en tenerlos definidos como clases hermanas. Pero nuevamente, los niveles de consumo no tienen un comportamiento esencialmente distinto, sólo sucede que sus colaboradores que establecen tiempo de actualización y modo de aviso varían. Por ello decidimos tener al objeto UpdateLevel con colaboradores UpdateTime y SignalMode, que sí pueden ser modificados para obtener distintos niveles.

4.4 Aclaraciones

- RunnerState tiene un método getValue, suponemos que se va a usar como parte del algoritmo de creación de entrenamientos.

- El TimeKeeper**,** una vez que se ejecuta el método de generación de instancia, ya empieza a contar. No se agregó un stop ya que a este momento no es necesario. Un TimeKeeper posiblemente necesitaría tener un stop, pero no lo vamos a implementar si no lo estamos utilizando.

- Plan es un mero agrupador de entrenamientos.

- En el diagrama de clases, las clases que se encuentran en negro son las implementadas en el primer sprint del proyecto de Android.

- Location es una clase de Android que representa una locación geográfica.

5. Planificación

La planificación del proyecto fue manejada íntegramente en RallyDev, en la sección 7. Apéndice se pueden encontrar tanto el Product Backlog como el Sprint Backlog.

5.1 Detalles del Sprint

Las siguientes fueron las User Stories comprometidas y finalmente implementadas en el sprint:

- Como equipo de desarrollo quiero configurar el ambiente de desarrollo para comenzar la etapa de programación.

- Como desarrollador quiero capacitarme en la tecnología a utilizar para comenzar el desarrollo.

- Como desarrollador quiero diseñar un modelo inicial de objetos de la aplicación para comenzar con el desarrollo.

- Como corredor quiero poder elegir un plan y uno de sus tipos de entrenamiento sugerido por la aplicación para poder hacer su seguimiento.

- Como corredor quiero poder ver la velocidad actual, la fase, la distancia recorrida y el tiempo transcurrido para alcanzar el objetivo impuesto para el entrenamiento.

- Como corredor, quiero que la aplicación me avise si debo aumentar, disminuir o mantener el ritmo mostrando la velocidad que a alcanzar para poder cumplir con el entrenamiento.

- Como corredor, quiero poder visualizar mi localización actual para tomar decisiones sobre mi rumbo.

En base a las User Stories anteriormente definidas podemos obtener los siguientes datos del sprint:

Tiempo del Sprint: **22 días.**
Cantidad de Story Points: **55 story points.**
Cantidad de Horas de tareas: **65 horas.**
Velocity inicial del equipo: **55 Story Points per Sprint**

5.2 Asunciones

- La selección de consumo de bateria solo afecta el intervalo de tiempo con el cual se realizan actualizaciones en los datos y la forma de notificación sonora sobre cambio de fase.

- Los datos personales del corredor se utilizarán globalmente para todos los planes, mientras que los de disponibilidad y objetivos son específicos para cada plan, por ello se ingresan y almacenan por separado.

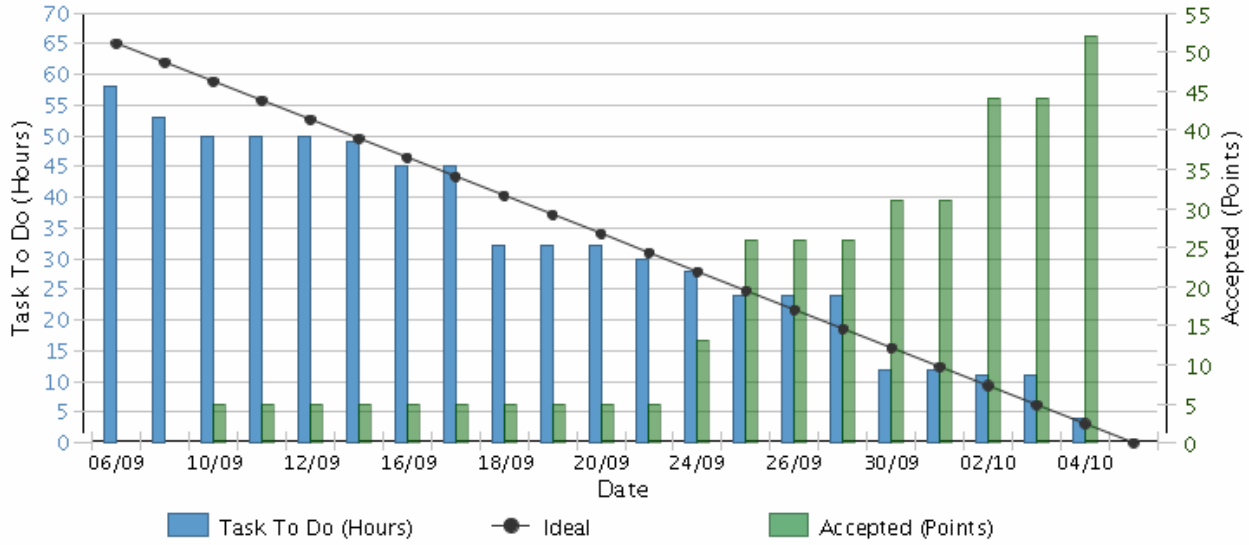
- Optamos por escribir como User Stories algunas tareas de desarrollo incorporando a “desarrollador” como usuario para poder identificar e incluir en las estimaciones más claramente estas necesidades.

- La User Story **US47** utiliza el rol “Como equipo de desarrollo quiero…” ya que consideramos que es una story que afectará a todos los desarrolladores siendo que las tareas derivadas los afectarán indefectiblemente.

- Como la localización, velocidad, tiempo, distancia, fase y velocidad objetivo se visualizarán en la misma pantalla pero los datos pertenecen a User Stories distintas, optamos por tener una tarea en una de ellas para crear tal pantalla, mientras para las demás la tarea será modificar la misma.

5.3 Burndown Chart

El siguiente es el burndown del sprint completo.



6. Retrospectiva

A continuación nombraremos algunos de los problemas que surgieron, las cosas que tendríamos que mejorar para futuras iteraciones, como también aquellas cosas en las que estuvimos correctos.

Tendríamos que haber compartido más el conocimiento sobre la configuración del ambiente de desarrollo. Esto nos hubiera ahorrado mucho tiempo en las primeras semanas de desarrollo.

Por otro lado, las estimaciones de tiempo de desarrollo quedaron cortas, pues menospreciamos el tiempo que tomaría adaptar nuestro modelo.

No definimos claramente las dependencias entre las tareas a realizar, lo cual hizo que algunas se bloquearán esperando la completitud de otras.

En una luz más positiva, las consultas desde etapas tempranas al Product Owner hicieron que el modelado fuera por buen camino y las dudas se evacuaran rápidamente.

Para concluir, creemos que la iteración fue exitosa y nos marcó los puntos en los que debemos mejorar: transferencia de conocimiento y estimaciones.

7. Apéndice

En esta sección se adjuntan el Product Backlog y el Sprint Backlog.

Rank	Id	Name	Iteration	Plan Est	TaskEst
1,0	US2	Como corredor quiero poder elegir un plan y uno de sus tipos de entrenamiento sugerido por la aplicación para poder hacer su seguimiento.	Sprint1	5,0	7,0
1,0	US3	Como corredor quiero poder ver la velocidad actual, la fase, la distancia recorrida y el tiempo transcurrido para alcanzar el objetivo impuesto para el entrenamiento.	Sprint1	8,0	8,0
1,0	US5	Como corredor, quiero que la aplicación me avise si debo aumentar, disminuir o mantener el ritmo mostrando la velocidad que a alcanzar para poder cumplir con el entrenamiento.	Sprint1	3,0	4,0
1,0	US8	Como corredor, quiero poder visualizar mi localización actual para tomar decisiones sobre mi rumbo.	Sprint1	13,0	9,0
1,0	US52	Como corredor quiero poder crear un plan de entrenamiento ingresando mi disponibilidad y mis objetivos para tenerlo definido.	-	3,0	7,0
2,0	US4	Como corredor, quiero que la aplicación me avise de los cambios de fase para poder cumplir con el entrenamiento.	-	3,0	0,0
2,0	US6	Como corredor quiero poder elegir el modo de consumo de batería para administrar el consumo de mi dispositivo.	-	8,0	0,0
2,0	US7	Como corredor, quiero poder ver las estadísticas de entrenamiento para poder seguir el progreso.	-	8,0	0,0
3,0	US1	Como corredor quiero poder ingresar mi información personal para que sea utilizada en mis planes de entrenamiento.	-	1,0	0,0
3,0	US45	Como corredor, quiero poder acceder a recorridos pasados para obtener información de los mismos.	-	5,0	0,0
3,0	US51	Como corredor quiero que se publique en redes sociales mi recorrido para compartirlo con mis contactos.	-	13,0	0,0

-	US46	Como desarrollador quiero diseñar un modelo inicial de objetos de la aplicación para comenzar con el desarrollo.	Sprint1	13,0	12,0
-	US47	Como equipo de desarrollo quiero configurar el ambiente de desarrollo para comenzar la etapa de programación.	Sprint1	5,0	13,0
-	US50	Como desarrollador quiero capacitarme en la tecnología a utilizar para comenzar el desarrollo.	Sprint1	8,0	12,0