Modelo de objetos para la implementación de un entorno ubicuo

Mariana Brachetta, Julio Monetti, Oscar León

Resumen: En el artículo se presenta el modelo de objetos diseñado para una aplicación móvil, que implementa un entorno de enseñanza ubicuo mediante la utilización de servicios de la nube. En la introducción se describe la concepción general del entorno ubicuo; posteriormente se describe el esquema del diagrama de clases, el modelo de persistencia de objetos, las características de funcionamiento de la aplicación y finalmente se comenta la estructura del proyecto.

Palabras claves: modelo de clases, orientación a objetos, persistencia de objetos

El entorno ubicuo es implementado mediante una aplicación móvil que permite a un estudiante acceder a una secuencia de aprendizaje, en la que debe resolver en forma autónoma actividades, que están vinculadas a puntos geográficos donde las mismas son habilitadas para su realización. La información del recorrido se almacena en tiempo real en una base de datos (BD). También se incorporan evaluaciones implementadas en la plataforma Moodle, y se dispone de un conjunto de orientaciones de ayuda, construidas con tecnología de Realidad Aumentada (RA).

Las ubicaciones geográficas son personalizadas por el estudiante al iniciar su trayectoria. A continuación se muestra la organización del entorno ubicuo.

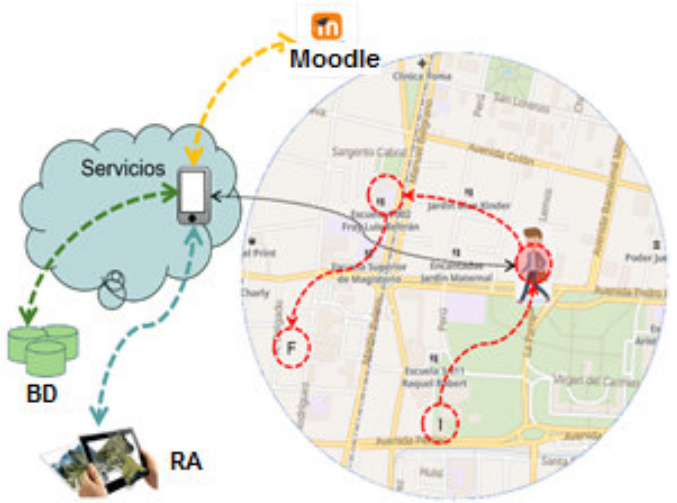

Fig. 1: Esquema del entorno ubicuo

El entorno implementa un esquema donde al transitar en las proximidades de puntos geográficos, se habilitan

actividades a resolver, luego al finalizarlas, se abre una evaluación en un navegador embebido en la App, y al superar la instancia se habilita la activación del próximo punto geográfico.

Arquitectura de la aplicación

La aplicación que presta servicios del lado del servidor (SAS), se despliega sobre un servidor de aplicaciones Java implementado en la capa de PAS (Plataforma como servicios) en nuestro contexto de desarrollo Cloud. En particular se ha usado Wildfly 10.0, no obstante, el desarrollo es portable y puede desplegarse en otros applications servers tales como JBoss, Tomcat, GlassFish, WebLogic, etc.

La aplicación en el back-end respeta una arquitectura por capas con la siguiente estructura.

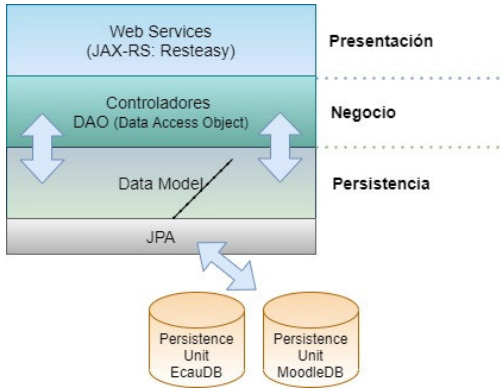

Fig. 2: Arquitectura de la aplicación

Modelo de clases implementado

Se presenta el modelo de clases diseñado con que representa la vista estática de objetos y sus relaciones,

para implementar el entorno descripto en la sección anterior. Se muestran solo los aspectos principales, habiendo suprimido clases utilitarias y no esenciales en ambos diagramas.

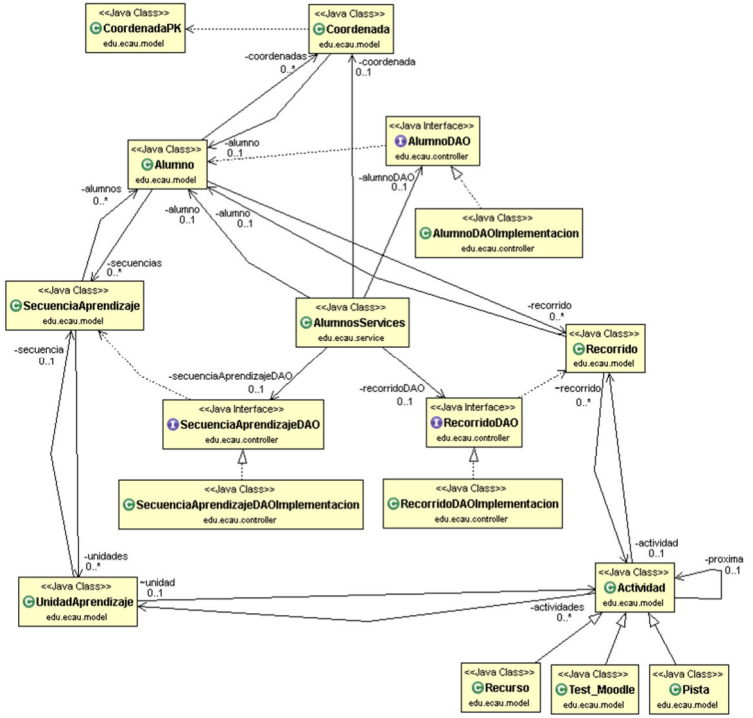

Fig. 3: diagrama de clases que modelan el entorno.

Class AlumnosServices

Es la principal clase en la capa de presentación. La clase agrega atributos de las interfaces de acceso a objetos de datos (DAO) y de los *beans* de entidad Alumno y Coordenada, donde cada uno de sus métodos implementa un servicio web.

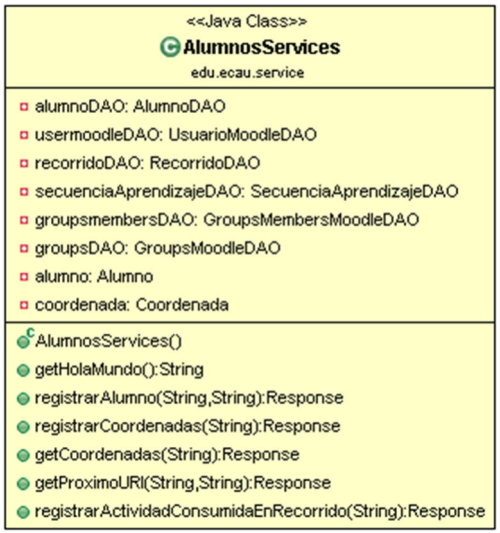

Fig. 4: clase AlumnosServices

Contiene métodos auxiliares como por ej. getHolaMundo(), para testear si el servicio está operativo.

Una de las tareas necesarias es registrar a un alumno en el entorno, lo cual se realiza con el método registrarAlumno(), al que se le pasa el nombre del usuario Moodle y su clave. Para esto se implementa un servicio que es utilizado por la App Móvil cuando un alumno se "loguea" en el entorno por primera vez, al "re-loguearse" en el mismo dispositivo, o si lo hace en otro dispositivo.

Otro aspecto que se debe atender es registrar la lista de coordenadas GPS (latitud, longitud) utilizadas para definir los geo-vallados de un alumno, lo cual se hace mediante el método registrarCoordenadas(). Las coordenadas GPS a registrar para el alumno se serializan en un objeto JSON.

Complementariamente a lo anterior, es necesario poder recuperar las coordenadas registradas por un alumno, lo que se hace el método getCoordenadas(), que devuelve la lista de coordenadas almacenadas para el alumno, con el detalle del orden de procesamiento y su estado, todo esto implementado con una cadena JSON.

Otro requerimiento es seguir la traza del trayecto que realiza un alumno, función que cumple el método getProximoURI(), que obtiene la próxima actividad a realizar por el alumno. Esta actividad puede ser del tipo Recurso, Test o Pista. Complementariamente, se actualiza el estado de la coordenada asociada a la actividad que está en curso, regresando una cadena JSON que permite identificar el tipo de actividad (Recurso, Test o Pista), la URI de acceso al recurso web (típicamente una URL), y el estado de procesamiento de la coordenada asociada. Es el servicio de mayor complejidad y el que más requerimientos realiza a la capa de negocios.

También se necesita conocer cuáles actividades han sido completadas por un alumno, para lo cual se usa el método registrarActividadConsumidaEnRecorrido().Este servicio es requerido cuando el alumno indica que completó una actividad. Esto es necesario para luego determinar si un alumno está en condiciones de pasar a la siguiente actividad de su recorrido.

Modelo de integración con Moodle

En el siguiente esquema se ilustra lo que constituye la capa de presentación que implementa los servicios web y los controladores DAO (Data Access Object) de la capa de negocios, que permiten realizar las operaciones requeridas por los web services sobre el modelo de datos.

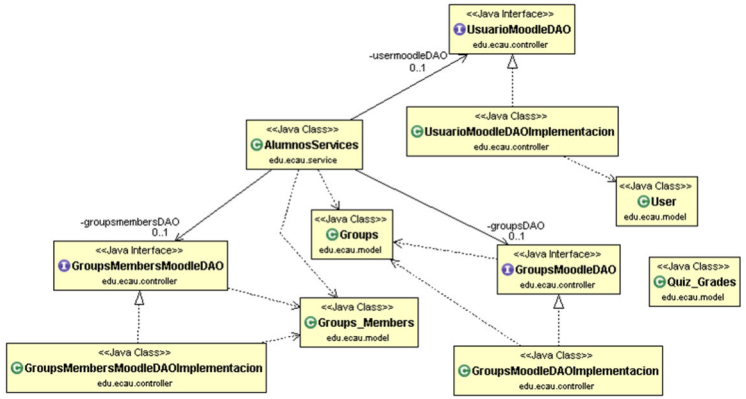

Fig. 5: Diagrama de clases integración con Moodle.

Se describen las principales clases del modelo, que constituye la capa de presentación que implementa los servicios web y los controladores DAO (Data Access Object) de la capa de negocios, que permiten realizar las operaciones requeridas por los web services sobre el modelo de datos.

Class AlumnoDAO

Controlador de sesión que permite acceder y gestionar los datos de un alumno y sus coordenadas. Trabaja en un contexto creado a partir de una unidad de persistencia. El contexto delimita el alcance de las transacciones y solo se tendrá acceso a las entidades definidas por la unidad de persistencia con que fue creado. A partir del contexto de persistencia, se define un EntityManager (Administrador de entidades en JPA) con el que se gestionan las entidades alcanzadas por el contexto.

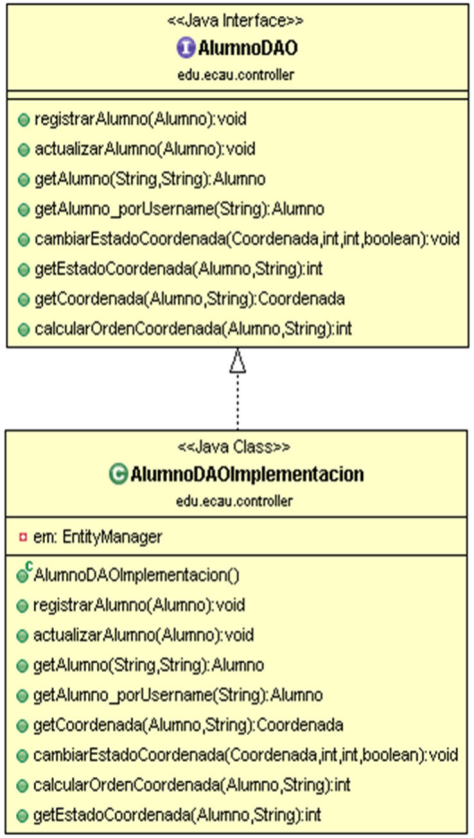

Fig. 6: Clase AlumnoDAO

Sus métodos, son accedidos desde los servicios web en la capa de presentación.

- registrarAlumno(): persiste un objeto del tipo alumno.
- actualizarAlumno(): realiza un merge para actualizar datos de un alumno.
- getAlumno(): ejecuta un query para obtener un objeto Alumno.

- getAlumno_porUsername(): hace un query para obtener un objeto Alumno que se le indica.
- getCoordenada(): obtiene de la lista de coordenadas de un alumno.
- cambiarEstadoCoordenada(): cambia los atributos del objeto coordenada recibido como parámetro, produciendo cambios que se persisten en la BD.
- calcularOrdenCoordenada(): calcula el orden correspondiente a la coordenada recibida como parámetro, incrementando en uno el orden de la última coordenada procesada para un alumno.
- getEstadoCoordenada(): devuelve el código de estado de la coordenada (0 – No procesada, 1 – En proceso, 2 – procesada).

En todos los casos se opera sobre objetos y sus relaciones en el entorno, no sobre tablas de la BD. Luego JPA mediante anotaciones implementa las conversiones necesarias. Esto permite mantener la independencia entre capas; así, la capa de presentación no se ocupa de cómo se implementan las operaciones sobre los objetos persistentes para gestionar los datos, solo utiliza sus servicios. Del mismo modo, la capa de negocios ve el modelo de datos como un modelo OO sin ocuparse de cómo se resuelve -en la capa de persistencia- el almacenamiento de datos en la BD.

Class SecuenciaAprendizajeDAO

Controlador de sesión para acceder y gestionar los datos de una secuencia de aprendizaje. Opera en un contexto de persistencia creado a partir de la unidad de persistencia.

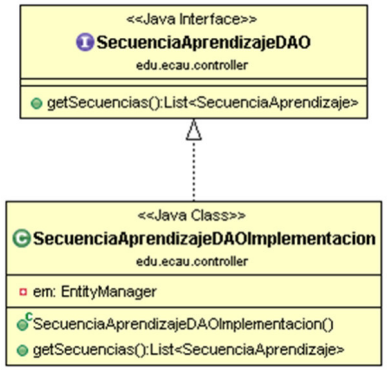

Fig. 7: Clase SecuenciaAprendizajeDAO

Sus métodos son accedidos desde la capa de presentación.

- getSecuencias(): devuelve un objeto del tipo con la lista de secuencias de aprendizaje existentes.

Class RecorridoDAO

Controlador de sesión que permite acceder y gestionar el recorrido de un alumno en la secuencia de aprendizaje.

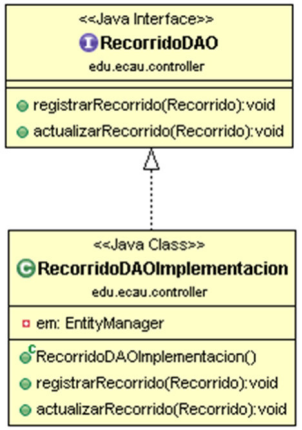

Fig. 8: Clase RecorridoDAO

Sus métodos, son accedidos desde los servicios web en la capa de presentación.

- registrarRecorrido(): persiste un objeto del tipo Recorrido, que vincula una actividad en particular a un alumno que la ha accedido y/o consumido.
- actualizarRecorrido(): realiza una operación del tipo merge para actualizar un objeto del tipo Recorrido. Este método es invocado por el servicio web para registrar en el recorrido de un alumno que completó o consumió la actividad en curso.

Class UsuarioMoodleDAO

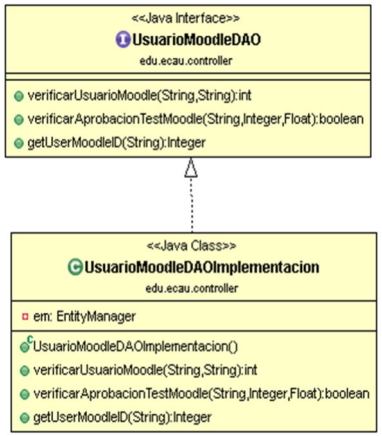

Fig. 9: Clase UsuarioMoodleDAO

Sus métodos, son accedidos desde los servicios web en la capa de presentación para realizar comprobaciones vinculadas al alumno contra la BD de Moodle.

- verificarUsuarioMoodle(): verifica que el alumno identificado por el username y password recibidos como parámetros exista como usuario en la BD de la

plataforma Moodle a la cual se integra el entorno, devolviendo un código para indicar el resultado de la operación.

- verificarAprobacionTestMoodle(): realiza un query a la BD de Moodle para verificar si el usuario recibido como parámetro, aprobó o no el test vinculado al quiz_moodle_id recibido. Para determinar la aprobación, la nota obtenida debe superar la nota de aprobación recibida como parámetro.
- getUserMoodleID(): devuelve el identificador unívoco de usuario en la BD Moodle para el usuario recibido como parámetro.

Class GroupsMoodleDAO

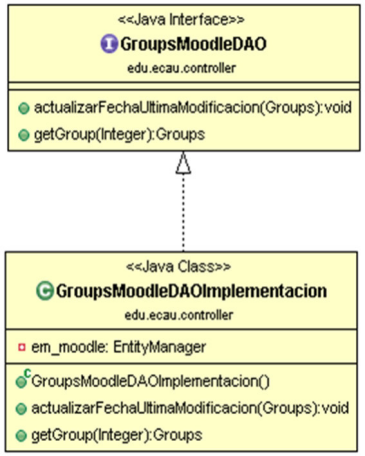

Fig. 10: Clase GroupsMoodleDAO

Sus métodos, son accedidos desde los servicios web en la capa de presentación para agregar usuarios a grupos de Moodle al momento de habilitar test.

- getGroup(): bbtiene un objeto del tipo Grupo (beans de entidad) que modela un grupo de usuarios Moodle cuyo identificador (id) está vinculado a un test.
- actualizarFechaUltimaModificacion(): actualiza la fecha de la última modificación del grupo Moodle que recibe como parámetro. Esto es un requisito para que Moodle advierta la modificación realizada.

GroupsMembersMoodleDAO

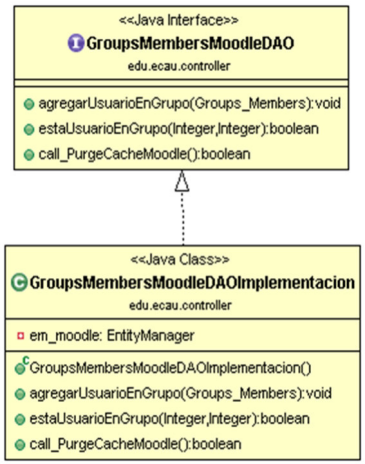

Fig. 11: Clase GropusMembersMoodleDAO

Sus métodos, son accedidos desde los servicios web en la capa de presentación para agregar usuarios a grupos de Moodle al momento de habilitar test.

- agregarUsuarioEnGrupo(): persiste un objeto del tipo GroupsMembers en el que se vincula un usuario Moodle a un grupo. Se usa cuando debe agregarse un usuario a un grupo para que le sea habilitado un test.

- estaUsuarioEnGrupo(): realiza un query a la BD de Moodle para verificar si un usuario pertenece a un grupo. El usuario y grupo se identifican de forma unívoca mediante los id recibidos como parámetros.
- call_PurgeCacheMoodle(): ejecuta un request http a la página helper2.php existente en el directorio de instalación de Moodle. Esta página, construida de forma especial para la integración del entorno con Moodle contiene una función que limpia la cache de Moodle. Esta acción es requerida cuando se agrega un usuario a un grupo para que el cambio sea inmediatamente tomado en cuenta y el usuario pueda acceder al test habilitado para ese grupo.

Conclusiones