Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 12, No.7, Mes Julio, 2019
ISSN: 2306-2495 | RNPS: 2343
Pág. 22-36
http://publicaciones.uci.cu

Tipo de artículo: Artículo original
Temática: Soluciones informáticas
Recibido: 20/06/2019 | Aceptado: 18/07/2019 | Publicado: 20/07/2019

API de Servicios REST para el seguimiento de indicadores en SIGE v3.0

REST Services API for tracking indicators in SIGE v3.0

Rosemary Morales Jiménez [1*], Danisleydis de la Caridad Hernández Rodríguez[2], Mariannys Santana Montero[3]

[1] Facultad de Ciencias y Tecnologías Computacionales, Universidad de las Ciencias Informáticas. rmorales@estudiantes.uci.cu
[2] Facultad de Ciencias y Tecnologías Computacionales, Universidad de las Ciencias Informáticas. ddhernandez@estudiantes.uci.cu
[3] Facultad de Ciencias y Tecnologías Computacionales, Universidad de las Ciencias Informáticas. msantanam@uci.cu

[*] Autor para correspondencia: rmorales@estudiantes.uci.cu

Resumen

El manejo de la información constituye una herramienta de vital importancia para las empresas en la actualidad, pues al analizar los datos, la gestión de la información ocurre de forma más fluida y sencilla. El Sistema Integrado de Gestión Estadística (SIGE), desarrollado en la Universidad de las Ciencias Informáticas (UCI) tiene como objetivo principal la informatización de los procesos y captura de datos estadísticos de una institución. Uno de sus procesos es el seguimiento de indicadores, teniendo en cuenta que inciden directamente en la toma de decisiones. SIGE fue desarrollado en el marco de trabajo Symfony v1.0 y ExtJS v3.4; presenta dificultad en el acceso a los datos generados por el sistema desde otras aplicaciones o dispositivos móviles, además de las dificultades para la realización de un seguimiento personalizado de sus indicadores. Resulta de interés para el Centro de Tecnologías de Gestión de Datos que la nueva versión sea implementada con una arquitectura orientada a servicios para solucionar las limitaciones antes mencionadas. En el presente trabajo se trazó como objetivo desarrollar una API de servicios REST para el SIGE v3.0 que permita dar seguimiento a los indicadores, además de garantizar el acceso a los recursos brindados por el sistema a través de aplicaciones móviles, de escritorio u otras aplicaciones web.

Palabras clave: seguimiento, indicadores, interfaz de aplicación, estadística, reporte

Abstract

Information management is a vital tool for companies today, because when analyzing data, information management occurs more smoothly and easily. The Integrated Statistical Management System (SIGE), developed at the University of Computer Science (UCI) has as its main objective the computerization of the processes and capture of statistical data of an institution. One of its processes is the monitoring of indicators, taking into account that they directly

influence decision making. SIGE was developed in the framework of Symfony v1.0 and ExtJS v3.4; It presents difficulty in accessing the data generated by the system from other applications or mobile devices, in addition to the difficulties in performing a personalized monitoring of its indicators. It is of interest to the Center for Data Management Technologies that the new version be implemented with a service-oriented architecture to solve the aforementioned limitations. This work aimed to develop a REST services API for SIGE v3.0 that allows monitoring of the indicators, as well as guaranteeing access to the resources provided by the system through mobile, desktop or other applications Web applications.

Keywords: *monitoring, indicators, application interface, statistics, report*

Introducción

Las empresas, durante los últimos años han venido utilizando las Tecnologías de la Información y las Comunicaciones (TIC) como apoyo en el proceso de toma de decisiones. Muchas de estas organizaciones utilizan reportes para analizar los datos, medir su progreso, contribuyendo así a la toma de decisiones, por lo que surge la necesidad de crear sistemas informáticos con el fin de gestionar y organizar los reportes. En correspondencia con lo anterior, existe una demanda creciente de aplicaciones informáticas que permitan la portabilidad de su interfaz a otro tipo de plataformas, lo que ha impulsado el desarrollo de soluciones escalables y que permitan que sus distintos componentes puedan evolucionar de forma independiente. La mayoría de las soluciones informáticas, en la actualidad, requieren de Interfaces de Programación de Aplicaciones (API, por sus siglas en inglés) que, a partir del uso de diferentes peticiones y servicios, logren la correcta separación entre el cliente y el servidor, brindando así facilidades a la hora de cambiar o probar nuevos entornos dentro del desarrollo (MUÑOZ-OSUNA *et al.* 2016)

En Cuba existen diversas entidades que desarrollan soluciones informáticas predestinadas a mejorar la calidad del trabajo en las empresas, siendo la Universidad de las Ciencias Informáticas (UCI) un centro productor de software que desarrolla soluciones informáticas orientadas a diversos sectores de la economía y los centros dentro y fuera de Cuba. Dentro de los centros que la integran está el Centro de Tecnologías de Gestión de Datos (DATEC) que tiene como objetivo principal la creación de bienes y servicios relacionados con la gestión de datos (COLMENARES *et al.* 2016), (SILVA *et al.* 2013).

DATEC cuenta con el Sistema Integrado de Gestión Estadística (SIGEv3.0), el mismo tiene como objetivo la informatización de los procesos de gestión estadística y su ventaja principal es la rápida confección de los resúmenes estadísticos a través de los cuales se dan a conocer los principales resultados en las diferentes esferas económicas y sociales del país.

Actualmente uno de los procesos utilizados en SIGE es el análisis de información mediante indicadores, los cuales inciden directamente en la toma de decisiones operativa y estratégica por la dirección de cada cliente. La ONEI es la

entidad que lleva el control de toda la estadística del país, por lo que son muchos los reportes que diariamente se analizan y con ellos sus indicadores de los cuales es necesario saber el comportamiento que toman en el tiempo y en determinada esfera.

Este análisis permite identificar potencialidades, conocer tendencias o comportamientos de fenómenos específicos, así como también otros datos estadísticos que reflejan aspectos esenciales de la realidad nacional; este proceso es denominado por la ONEI como seguimiento de indicadores.

Aun cuando la diversidad y el cúmulo de indicadores permiten tener un mayor entendimiento de la información, existen dificultades a la hora de realizar un reporte o seguimiento personalizado de determinado indicador en SIGE:

- Para obtener del sistema, una información solicitada, es necesario haber realizado previamente un diseño del reporte y la implementación de las funciones necesarias para visualizar dicho comportamiento.
- La posibilidad de informar con premura el comportamiento de un indicador específico se dificulta al no contar con reportes genéricos que brinden información de cualquiera de los indicadores existentes en el sistema.
- Para cambiar alguno de los indicadores establecidos por la empresa, necesariamente habría que modificar todo el reporte.

Materiales y métodos o Metodología computacional

El modelo conceptual puede utilizarse para capturar y expresar el entendimiento ganado en un área bajo análisis como paso previo al diseño de un sistema, ya sea de software o de otro tipo. Similares a los mapas mentales utilizados en el aprendizaje, el modelo de dominio es utilizado como un medio para comprender el sector de negocio al cual el sistema va a servir (SILVA *et al.* 2013). El presente modelo conceptual tiene como objetivo contribuir a la comprensión de la API REST para el seguimiento de indicadores en el SIGE v3.0 (Ver Figura 1).

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 12, No.7, Mes Julio, 2019
ISSN: 2306-2495 | RNPS: 2343
Pág. 22-36
http://publicaciones.uci.cu

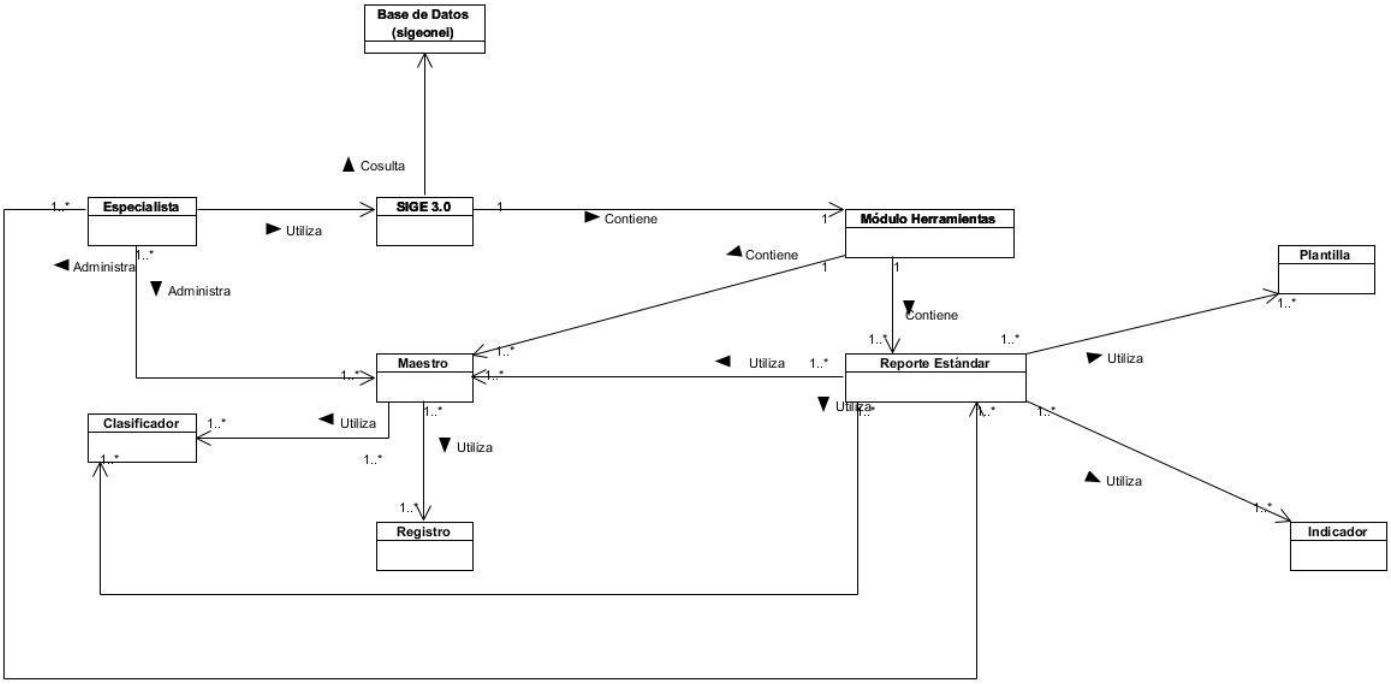

Fig. 1 Modelo Conceptual

A continuación se realiza una descripción de las clases del modelo de conceptual

Especialista: usuario encargado de generar los maestros y los reportes estándares.

SIGE 3.0: Sistema Integrado de Gestión Estadística, dentro del cual se encuentra el Módulo de Herramientas, el cual posee a su vez la generación de Maestros y Reportes Estándares.

Módulo Herramientas: permite mediante su concepción genérica llevar a cabo el seguimiento de indicadores, mediante las opciones de generación de Maestros y Reportes Estándares.

Maestro: permite depurar el universo de información estadística obtenida durante el proceso de captación, para ello hace uso de los registros y los clasificadores.

Clasificador: son aquellas agrupaciones y niveles jerárquicos que definen el alcance de cada elemento.

Registro: permite agrupar los centros informantes en diversas categorías.

Reporte Estándar: es la sumatoria de valores y datos estadísticos solicitados por la ONEI a la fuente de datos, a partir de una plantilla y un maestro.

Plantilla: es el elemento que sirve como guía para el diseño de un modelo estadístico.

Indicador: es el componente principal que conforma una plantilla y por el cual realizando un seguimiento del mismo se puede obtener un comportamiento de determinado elemento.

Base de Datos: conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso, en ella se encuentra toda la información centralizada del SIGE

Es de gran importancia lograr un entendimiento entre clientes y el equipo de proyecto a lo largo del proceso de desarrollo del sistema, por lo que se deben establecer los objetivos del producto a desarrollar. Con el fin de llegar a un consenso entre ambas partes, surge la especificación de requisitos, la cual es el proceso de desarrollo de software donde se analizan las necesidades exactas de los usuarios. Luego son traducidas a precisas funciones y acciones que serán usadas en el desarrollo del sistema. Un correcto levantamiento de requisitos permite posteriormente la construcción de un software de calidad. Estos pueden clasificar por categorías en: funcionales y no funcionales (PRESSMAN 2010), (BAQUERO *et al.* 2016).

Requisitos Funcionales: Un RF es una capacidad o condición que el sistema debe cumplir (PRESSMAN 2010).
A continuación, se muestra el listado de requisitos funcionales que debe cumplir el sistema:

RF01-Crear Maestro (caso de uso Gestionar Maestro)
RF02-Listar Registros (caso de uso Gestionar Maestro)
RF03-Listar Clasificadores comunes a partir de los Registros (caso de uso Gestionar Maestro)
RF04-Eliminar Maestro (caso de uso Gestionar Maestro)
RF05-Listar Maestros de forma paginada (caso de uso Gestionar Maestro)
RF06-Recargar Maestros (caso de uso Gestionar Maestro)
RF07-Actualizar datos del Maestro (caso de uso Gestionar Maestro)
RF08-Visualizar definición del Maestro (caso de uso Gestionar Maestro)
RF09-Visualizar datos del Maestro (caso de uso Gestionar Maestro)
RF10-Buscar Maestros (caso de uso Gestionar Maestro)
RF11-Crear Reporte Estándar (caso de uso Gestionar Reporte Estándar)
RF12-Seleccionar Rango de Fecha (caso de uso Gestionar Reporte Estándar)
RF13-Listar Maestros (caso de uso Gestionar Reporte Estándar)
RF14-Listar Clasificadores comunes a partir de los Maestros (caso de uso Gestionar Reporte Estándar)

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 12, No.7, Mes Julio, 2019
ISSN: 2306-2495 | RNPS: 2343
Pág. 22-36
http://publicaciones.uci.cu

RF15-Listar Indicadores (caso de uso Gestionar Reporte Estándar)

RF16-Listar Plantillas (caso de uso Gestionar Reporte Estándar)

RF17-Autenticar Usuario (caso de uso Autenticar Usuario)

Diagrama de Casos de Uso del Sistema: Muestran el funcionamiento de un sistema desde el punto de vista del actor. Estos son la representación de los requisitos funcionales del sistema agrupados por casos de uso. A continuación, se muestra el diagrama de casos de uso del sistema. (Ver Figura 2).

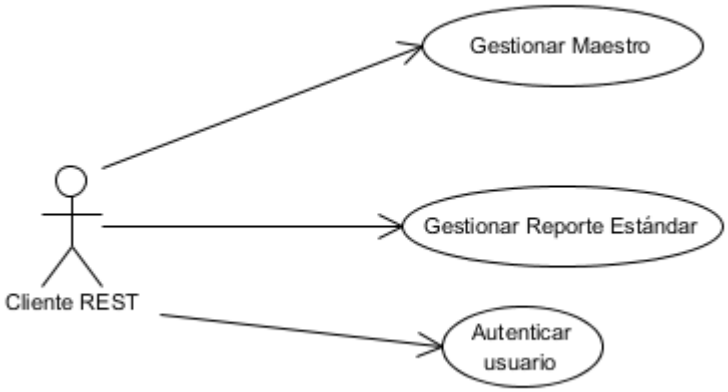

Fig. 2 Diagrama de Casos de Uso del Sistema

Los patrones de software son un mecanismo para capturar conocimiento del dominio, en forma que permita que vuelva a aplicarse cuando se encuentre un problema nuevo. En ciertos casos, el conocimiento del dominio se aplica a un nuevo problema dentro del mismo dominio de la aplicación. En otros, el conocimiento del dominio capturado por un patrón puede aplicarse por analogía a otro dominio de una aplicación diferente por completo.

CRUD (*Create, Read, Update, Delete*)

- CRUD Parcial: este patrón es preferible cuando uno de los flujos alternativos del caso de uso es más significativo, muy largo o mucho más complejo que el patrón completo
- CRUD Completo: permite modelar las diferentes operaciones para administrar una entidad de información, tales como crear, modificar, leer, y eliminar.

La Tabla 1, muestra una descripción del Caso de Uso ¨ Gestionar Maestro¨

Tabla 1:Descripción del Caso de Uso ¨ Gestionar Maestro¨

Objetivo		Gestionar Maestro
Actores		Cliente REST: (Inicia), Adiciona, Elimina, Recarga, Visualiza y Edita maestros.
Resumen		El CU Gestionar Maestro se inicia cuando el Cliente REST realiza una petición al controlador Maestro. En este caso se puede: Generar, Eliminar, Listar, Recargar datos de Maestro, Visualizar datos, ver la Definición de Maestro y Editar Maestro. Cada una de estas secciones se describe a continuación, finalizando así el caso de uso.
Complejidad		Media
Prioridad		Media
Precondiciones		Es autenticado el administrador en el sistema. Existe conexión con la Base de Datos (BD). Existen registros con clasificadores asociados. Existen clasificadores.
Postcondiciones		Se gestionan los maestros en el sistema.
Flujo de eventos		
Flujo básico: Gestionar Maestro		
	Actor	**Sistema**
1.	Realiza una petición POST, PUT, GET, DELETE enviando un objeto de tipo Maestro a la URL ”maestros”.	

2.		El API recibe el objeto de tipo Maestro y se pueden realizar una de las siguientes operaciones: Crear Maestro. Sección 1. Eliminar Maestro. Sección 2. Recargar Maestro. Sección 3. Visualizar Maestro. Sección 4. Editar Maestro. Sección 5.
Sección 1: "Adicionar Maestro"		
Flujo básico Adicionar Maestro		
	Actor	Sistema
1.	Realiza una petición POST, enviando un objeto de tipo Maestro a la URL "maestros".	
2.		El API recibe el objeto de tipo Maestro con los datos requeridos para la creación de un nuevo maestro. Los datos requeridos son: Nombre, Alias, Fecha de Cierre, Fecha de Generación, Registros asociados, Clasificadores comunes y Observaciones.
3.		Verifica la existencia del Maestro. De no existir se guarda con sus respectivos atributos y se notifica al Cliente REST mediante un mensaje HTTP 200 el éxito de la operación. Además de ello, muestra los datos del nuevo Maestro, terminando así el caso de uso.
Flujos alternos		
"Datos Incorrectos"		
	Actor	Sistema
		2a. Verifica que ya existe el Maestro y se retorna un mensaje HTTP 500 notificando que no se puede efectuar la operación. 2b. En caso de existir valores en blanco o valores incorrectos,

		retorna un mensaje HTTP 500 especificando que los datos son inválidos, terminando así el caso de uso.
Sección 2: "Eliminar Maestro"		
Flujo básico Eliminar Maestro		
	Actor	Sistema
1.	Realiza una petición DELETE, enviando el Alias y la Fecha de Cierre a la URL "maestros".	
2.		Retorna un mensaje HTTP 200 notificando que se eliminó el maestro y se retorna el maestro, terminando el caso de uso.
Sección 3: "Recargar Maestro"		
Flujo básico Recargar Maestro		
	Actor	Sistema
1.	Realiza una petición PUT, enviando un objeto de tipo Maestro a la URL "maestros".	
2.		Retorna un mensaje HTTP 200 notificando que se recargó el maestro y se retorna el maestro, terminando el caso de uso.
Sección 4: "Visualizar Maestro"		
Flujo básico Visualizar Maestro		
	Actor	Sistema
1.	Realiza una petición GET, enviando el Alias y la Fecha de Cierre a la URL "maestros/alias/fechadecierre".	
2.		Retorna un mensaje HTTP 200 notificando que se encontró el maestro y se retorna el maestro, terminando el caso de uso.

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 12, No.7, Mes Julio, 2019
ISSN: 2306-2495 | RNPS: 2343
Pág. 22-36
http://publicaciones.uci.cu

Sección 5: "Editar Maestro"		
Flujo básico Editar Maestro		
	Actor	Sistema
1.	Realiza una petición PUT, enviando un objeto de tipo Maestro a la URL "maestros".	
2.		Retorna un mensaje HTTP 200 notificando que se editó el maestro y se retorna el maestro, terminando el caso de uso.
Relaciones	CU incluidos	N/A
	CU extendidos	N/A
Requisitos no funcionales	N/A	
Asuntos pendientes	N/A	

Resultados y discusión

En el diseño se modela el sistema y se define su forma para determinar cómo funcionará el producto final de forma general, centrándose en como los requisitos funcionales y no funcionales tienen impacto en el sistema, considerándose fundamental para el proceso de implementación. Generalmente se desarrolla con diagramas que describan las relaciones entre las entidades y su secuenciado, imponiendo una estructura del sistema (Bruegge, y otros, 2008).

Los patrones arquitectónicos representan el nivel más alto dentro del sistema de patrones y expresan el esquema de la estructura fundamental de la organización para sistemas de software. Proveen un conjunto de subsistemas predefinidos, especifican sus responsabilidades e incluyen reglas y guías para organizar las relaciones entre ellos. Cada patrón ayuda a lograr una propiedad específica del sistema global como es la adaptabilidad de la interfaz de usuario (Almeira, y otros, 2007).

Patrón arquitectónico: El patrón arquitectónico a emplear es Modelo Vista Controlador (MVC), es un patrón de arquitectura de software que separa la lógica de negocio de la interfaz de usuario, facilita la evolución por separado de ambos aspectos e incrementa la reutilización y la flexibilidad del sistema. (Potencier, y otros, 2011)

- **Modelo:** Es el componente que se encarga de la lógica. Gestiona la información para que pueda ser utilizada por la vista y el controlador. Envía a la vista aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al modelo a través del controlador.
- **Controlador**: Responde a eventos e invoca peticiones al modelo cuando se hace alguna solicitud sobre la información. También modifica la vista si se solicita un cambio en la forma en que se presenta el modelo (por ejemplo, cuando se agrega o modifica algún elemento del diagrama es necesario actualizarlo en la vista).

El caso de uso se inicializa cuando un usuario envía un objeto de tipo maestro en formato JSON, a través de un cliente REST, a la URL o recurso "maestro" [http://127.0.0.1:8085/api/v1/maestros], en el cuerpo de la petición haciendo uso del método POST. El controlador recibe la petición HTTP y utiliza la instancia del servicio *MaestroService*, el cual invoca a su método *createMaestro* que recibe como parámetro el objeto de tipo Maestro.

El servicio emplea el método *save* del repositorio *MaestroRepository* que implementa la interfaz *JPARepository*, quien recibe una instancia de la clase Maestro y se encarga de persistir la misma en la base de datos. Si la operación es exitosa el método *createMaestro* de la clase MaestroService retorna al controlador MaestroController el objeto maestro recién creado, y a su vez el MaestroController lo retorna al Cliente REST en formato JSON junto con el código HTTP 201 *Created*. En caso de fallar, la clase *MaestroService* retorna al controlador *MaestroController* y este a su vez al Cliente REST un mensaje de error con la excepción correspondiente y un código de estado HTTP 500. Como precondición para poder ejecutar esta operación el usuario debe estar autenticado, en caso de que este no este autenticado se envía un código de error HTTP 401 *Unauthorized*, además el usuario debe tener un rol con permisos para realizar la petición, en caso de no tener este permiso retorna un error con código HTTP 403 *Forbidden*.

Los patrones GRASP están diseñados para asignar responsabilidades, se utilizan en la realización de diagramas de interacción.

Experto: asignar una responsabilidad al experto en información o a la clase que tiene la información requerida. Se garantiza que cada clase del sistema asuma las responsabilidades que le conciernen, según las funcionalidades que se quieren implementar y a partir de la información que posee; por lo que cada clase contendrá la información necesaria para cumplir su responsabilidad, el mismo se evidencia en la clase Maestro (Figura 3), esta es experta en representar

los datos de la entidad maestro en la Base Datos, se comporta como una colección en memoria de los datos de la tabla maestro o de una fila de esa tabla (como un objeto simple) y permite operar sobre ellos.

Fig. 3 Patrón Experto

Controlador: este objeto no pertenece a la interfaz de usuario, es el encargado de controlar el flujo de acciones y peticiones realizadas en la web. Este define el método para la operación del sistema. En la (Figura 4) la clase *MaestroController* es la que atiende cada una de las peticiones realizadas, manejando luego la acción requerida para satisfacer la petición. Sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado. Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación para aumentar la reutilización de código y a la vez tener un mayor control.

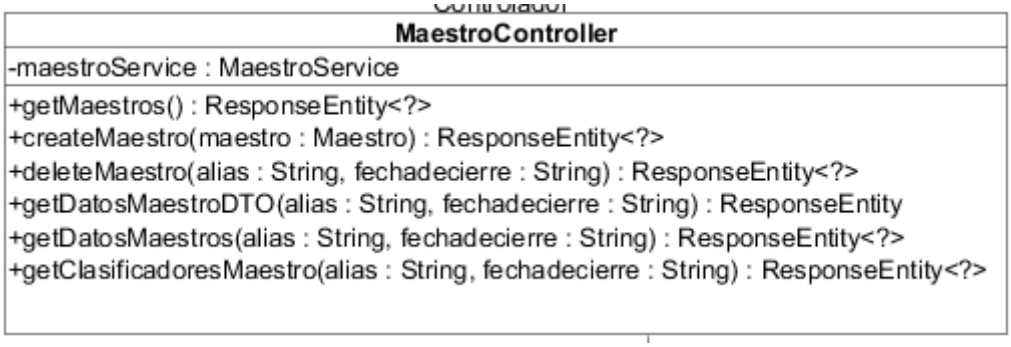

Fig. 4 Patrón Controlador

Creador: es el responsable de crear e instanciar otras clases. Ejemplo: en la clase *MaestroService* (Figura 5) se encuentra entre sus funcionalidades la opción de crear instancias de la clase Maestro.

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 12, No.7, Mes Julio, 2019
ISSN: 2306-2495 | RNPS: 2343
Pág. 22-36
http://publicaciones.uci.cu

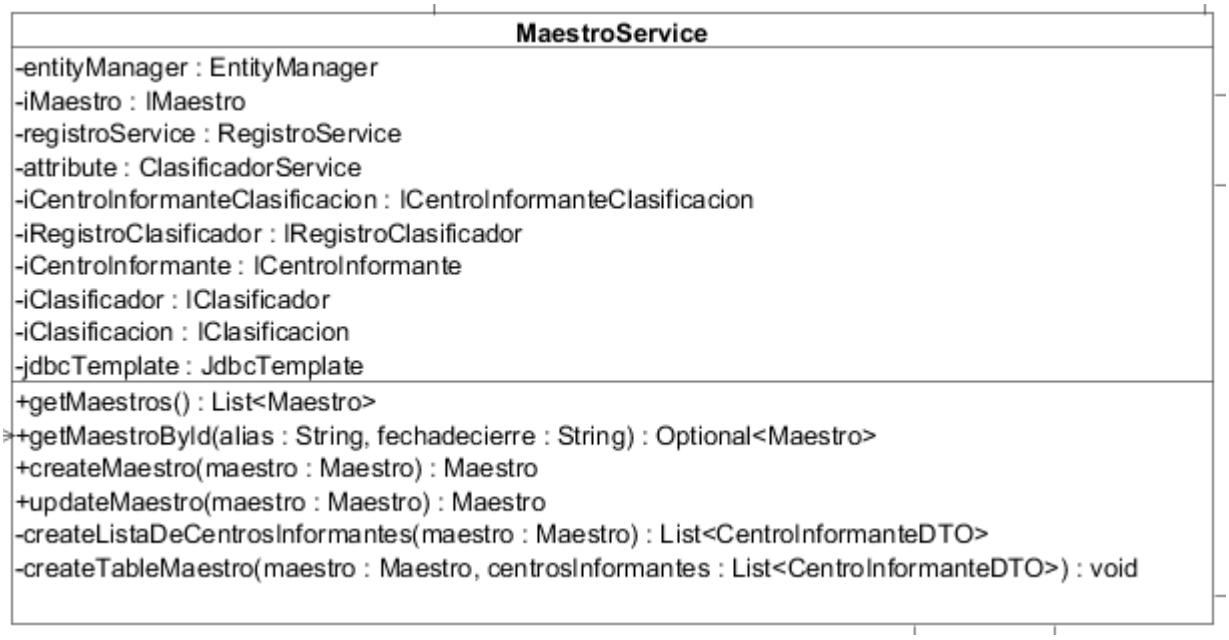

Fig. 5 Patrón Creador

Bajo Acoplamiento: es la medida de cuanto una clase está conectada con otras clases. El bajo acoplamiento permite que el diseño sea más independiente. Las clases que implementan el acceso a los datos se encuentran en el modelo, proporcionando que la independencia en este caso sea baja. En la (Figura 6) se puede observar que cada clase posee la mínima dependencia, proporcionando la mínima cantidad de relaciones entre las clases de la vista.

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 12, No.7, Mes Julio, 2019
ISSN: 2306-2495 | RNPS: 2343
Pág. 22-36
http://publicaciones.uci.cu

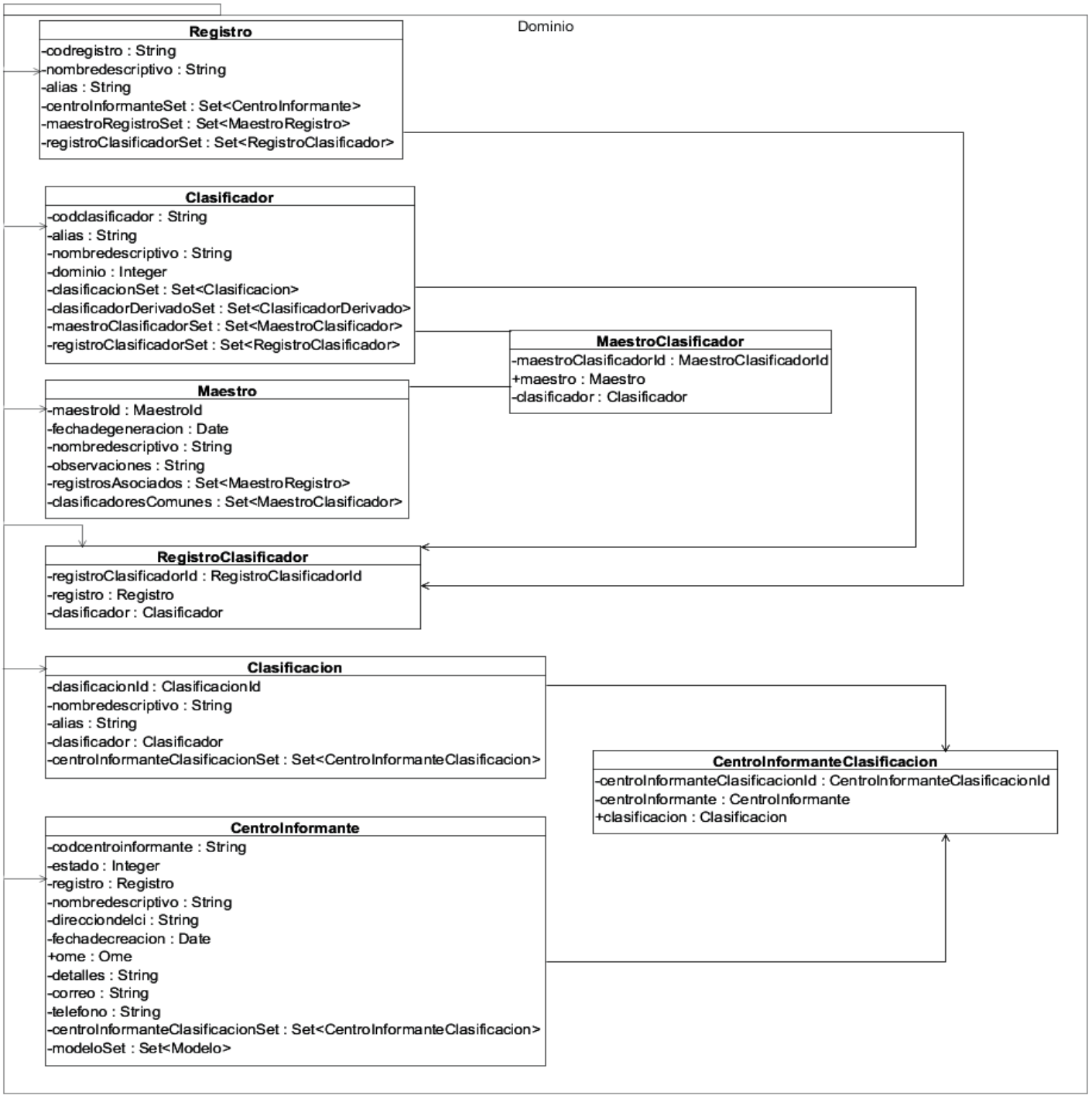

Fig. 6 Patrones Bajo Acoplamiento y Alta Cohesión

Alta Cohesión: una clase de diseño cohesiva tiene un conjunto pequeño y centrado de responsabilidades; para implementarlas emplea atributos y métodos de objetivos únicos. En la (Figura 6) se ve evidenciado este patrón.

Conclusiones

Con la realización de artefactos como el modelo conceptual se logró esclarecer a todos los involucrados, cómo se realiza el proceso que propone mejorar la solución. Al especificar los requisitos funcionales y no funcionales del sistema se establecieron los límites del mismo.

Se especificaron aquellos considerados como críticos por su relevancia; brindándole al equipo de desarrollo un soporte más específico y concreto de lo que se debe hacer. Conjuntamente se realizó el diagrama de clases del diseño para un mayor nivel de detalle, dejando el punto de partida marcado para la etapa de implementación.

Referencias

BAQUERO, L.; D. MENDOZA, *et al*. Extensión de la herramienta Visual Paradigm for UML para la evaluación y corrección de Diagramas de Casos de Uso *Publicaciones*, 2016, Vol.9(N0.4): 1-14.

COLMENARES, J. E.; N. R. HENDEZ, *et al.* Laboratorios virtuales desde la perspectiva de resolución de problemas: Caso de la asignatura de mecánica de suelos

016 *Rev. Educación en Ingeniería*, 2016, Vol.11(No.22): 97-103.

MUÑOZ-OSUNA, F. O.; A. MEDINA-RIVILLA, *et al.* Jerarquización de competencias genéricas basadas en las percepciones de docentes universitarios *Educación química*, 2016, 27: 126-132.

PRESSMAN, R. S. Ingeniería de Software: Un enfoque práctico 7ma edición. , 2010.

SILVA, C.; E. CHAPIS, *et al*. La gestión del conocimiento en función del control interno en el municipio Lajas, 2013, Vol.5(No.3).

Bruegge, Bernd y Dutoit, Allen H. 2008. Ingeniería de software orientado a objetos. [ed.] Pearson Educación. 2008. pág. 553.

Almeira, A y Pérea, Cavenago Pérez, V. 2007. Arquitectura de Software: Estilis y Patrones. 2007.

Potencier, Fabien y Weaver, Ryan. 2011. LIBROSWEB. LIBROSWEB. [En línea] 2011. [Citado el: 10 de diciembre de 2013.] http://librosweb.es/symfony_2_3/.