Maestría en Bases de Datos

Proyecto de Investigación Aplicada

Propuesta NoSQL de Implementación de Modelos Universales de Datos

Luis Alberto Ruiz Arauz

Marzo, 2024

TRIBUNAL EXAMINADOR

Este proyecto fue aprobado por el Tribunal Examinador de la carrera: **Maestría en Tecnología de Bases de Datos,** requisito para optar por el título de grado de **Maestría,** para el estudiante: **Ruiz Arauz Luis Alberto.**

JOSE ALBERTO CABEZAS JAIKEL (FIRMA)
PERSONA FISICA, CPF-01-1089-0511.
Fecha declarada: 04/03/2024 10:00:00 PM

MBD José Cabezas Jaikel
Tutor

Firmado digitalmente por RAFAEL ERNESTO VILLEGAS VILLEGAS (FIRMA)
Fecha: 2024.03.05 10:54:48 -06'00'

M.Eng. Ernesto Villegas Villegas, PhD. Candidate
Lector 1

JASON ULLOA HERNANDEZ (FIRMA)
Firmado digitalmente por JASON ULLOA HERNANDEZ (FIRMA)
Fecha: 2024.03.05 13:45:34 -06'00'

M.Sc. Jason Ulloa Hernández
Lector 2

San José, Costa Rica, 4 de marzo de 2024

Firmada digitalmente, de conformidad con la Ley de Certificados, Firmas Digitales y Documentos Electrónicos N° 8454, destacando el artículo 9°-

Propuesta NoSQL de Implementación de Modelos Universales de Datos

Luis A Ruiz Arauz, y José A. Cabezas Jaikel. Universidad Cenfotec

[1] ***Resumen*— En este artículo se analizó la importancia de los Modelos Universales de Datos en el desarrollo de software y los factores que inciden en su baja popularidad. Se identificó como uno de los factores principales la complejidad para su desarrollo e implementación y como su causa raíz el desajuste por impedancia objeto-relacional. Ante dichos factores se propone cambiar el modelo relacional utilizado por uno NoSQL.**

Se desarrolló una aplicación WebAPI con MongoDB como base de datos, C# como lenguaje de programación, además de MongoDB C# Driver como componente de conexión para demostrar que es posible realizar este tipo de implementación.

Durante el proceso de desarrollo se identificaron oportunidades de mejora, como lo es separar los modelos y funcionalidad de persistencia creados en uno o más componentes reutilizables, así como la posibilidad de crear un repositorio único, en la nube, a través del cual se puedan conectar las aplicaciones.

Por otra parte, durante la implementación se logró dejar a un lado la idea de modelado de datos per se que caracteriza los Modelos Universales de Datos y en su lugar, incluir únicamente un modelado de objetos, creando así el concepto de Objetos o Componentes Universales de Negocio.

***Índice de Términos*—Modelos Universales de Datos, UDM, NoSQL, Procesos de Negocio, Desajuste por Impedancia Objeto-Relación, Persistencia Políglota.**

I. INTRODUCCIÓN

No reinvente la rueda, es una frase cuyo significado está arraigada en el mundo de la ingeniería en cualquiera de sus ramas, ya que en general, lo que se busca desde la ingeniería es resolver las necesidades de la sociedad a través de diversos conocimientos que permitan el aprovechamiento de los recursos.

En otras palabras esta frase refiere a que no se arriesgue ni pierda el tiempo en idear cómo resolver un problema del cual ya se tiene una solución estudiada, probada y perfeccionada.

En Ingeniería de Software a ese tipo de soluciones se les conoce como Patrones de Diseño y su uso constituye una parte muy importante de las buenas prácticas de desarrollo de sistemas de información. Un patrón de diseño es un modelo a seguir, no porque su uso sea obligatorio, sino porque representa la solución recomendada a un problema o funcionalidad específica.

Seguir un patrón de diseño permite ahorrar tiempo, no solo en el diseño, sino también en la corrección de problemas y errores potenciales, incluso tener la necesidad de volver a iniciar el proceso. El tiempo es un factor que en cualquier actividad implica dinero y en el desarrollo de un proyecto puede determinar su éxito o fracaso.

Ese beneficio intrínseco de los Patrones de Diseño hace que se hagan populares al punto de establecerse como norma o estándares de diseño. Su uso es tan amplio y recomendado que cualquier analista de sistemas los ha tenido que estudiar, al menos, para prepararse para una entrevista laboral.

Los Modelos Universales de Datos son la versión de los Patrones de Diseño para los procesos de negocio a nivel de base de datos. Son una propuesta de diseño que se ha desarrollado y promete ser el punto de partida para cualquier sistema empresarial.

Sin embargo, cuando se consulta entre colegas de Ingeniería de Software por Modelos Universales de Datos, las reacciones son diferentes. La mayoría, frunce el ceño porque no está familiarizado con el tema, los más atrevidos se lanzan a tratar de armar una explicación con base en lo que entienden o conocen de cada palabra por separado.

Por lo que es entendible que su uso en la práctica es casi nulo, ya que su estudio en la universidad se limita al interés del profesor o profesora que imparte los cursos, la cercanía que tenga con el tema y que por alguna razón considere pertinente incluir esta temática en el desarrollo del curso.

En este artículo se pretende exponer algunas razones por las que los Modelos Universales de Datos no tienen la popularidad que merecen y proponer a la vez una estrategia de implementación que refute dichas razones, para que estas logren cumplir la meta de facilitar la vida de los desarrolladores y construir mejores sistemas de información.

[1]L. Ruiz es Lead consultantsr en Gorilla Logic, empresa de desarollo de software, San José, Costa Rica (email: luis.ruiz@gorillalogic.com)

J. Cabeza es Enterprise Technology Architect y Data Scientist para Oracle, San José, Costa Rica, además de profesor de maestría de la Universidad Cenfotec (email: jcabezas@ucenfotec.ac.cr)

El codigo fuente de la aplicacion y los modelos de datos utilizados en este articulo se pueden accesar en el repositorio, Cenfotec-Paper-UDM, a travex del siguiente link: https://github.com/machocr/Cenfotec-Paper-UDM\

II. MODELOS UNIVERSALES DE DATOS

La Realidad

Las empresas son, en esencia, un negocio, que se quiere, sostenible en el tiempo. No importa su giro, qué produce, su tamaño, incluso las que se hacen llamar "sin fines de lucro", todas tienen la misma meta; aumentar su ganancia de dinero y mantenerlo en el futuro.

Para eso han sido creadas las empresas y esa es su razón de ser, una empresa que no genere dinero tiene sus días contados. Por lo tanto, para sus propietarios (los inversionistas), las empresas no son otra cosa más que máquinas que producen dinero.

El deseo de estar mejor de los empresarios, es la fuerza que permite reunir a los trabajadores, clientes, proveedores, incluso el estado y la sociedad en un esfuerzo común para beneficiarse mutuamente con la empresa. Ese lucro, si se quiere, hasta egoísta de los empresarios, en un contexto de libre mercado, ha impulsado el progreso de la humanidad, ha creado riqueza donde no había, ha dado empleo a las personas, ha permitido producir mejores productos, de mejor calidad y menor precio, ha impulsado el desarrollo científico y tecnológico, entre otras muchas cosas.

Lamentablemente, las máquinas de hacer dinero, las legales, tampoco las que están en los bancos centrales, solo existen en los sueños de los accionistas. Por lo tanto, las empresas, para ganar dinero, deben producir algo que alguien considere de valor y que quiera comparar de manera consistente. Ahí es donde ya se empiezan a ver diferencias que existen en las empresas.

Unas empresas producen y venden teléfonos móviles, otras pantalones de mezclilla, otros zapatos, otras televisores. Y existen empresas que venden servicios, servicios hospitalarios, servicios bancarios, servicios alimenticios entre muchos otros.

Sin embargo, con todas esas diferencias circunstanciales, no se puede negar que las empresas poseen similitudes entre sí; todas necesitan mantener datos de sus clientes, sobre la planilla, sobre las ventas. Por lo que necesitan sistemas para controlar y automatizar los procesos de facturación, órdenes de venta, de compra, de trabajo, contabilidad, producción, recursos humanos, presupuesto, administración de proyectos, manejo de contactos, por mencionar algunos.

Se resalta el caso de las empresas públicas, las cuales a pesar de que también deben generar ganancia económica para sostenerse, esta no es su prioridad. Ya que su existencia no depende del éxito económico que pueda generar y porque poseen otros mecanismos de financiamiento. Sin embargo, siguen siendo empresas y cómo se analiza más adelante, su administración sigue los mismos patrones.

El Modelo

Los sistemas de información automatizados tienen su origen no hace muchos años atrás, se remontan a los años 50s con el desarrollo de las computadoras. Estos sistemas empezaron a almacenar los datos en dispositivos, utilizando un sistema físico de tablas. A la representación lógica y física de sistema de tablas se conocería como modelo entidad relación y junto al lenguaje SQL que permite almacenar y extraer datos, se convertirían en el estándar de facto para el desarrollo de bases de datos.

No es de extrañar que los nuevos ingenieros o analistas de sistemas que surgieron en ese entonces, utilicen este modelo de desarrollo, que, además permitió solucionar todos los problemas de información a los que se enfrentaban en ese entonces.

La enseñanza del modelado de datos, definido como el proceso de abstraer la realidad y representarla en una estructura de base de datos, se limitaba en una serie de convenciones de diagramación, algunos principios de buen diseño, de buenas prácticas y reglas de normalización para atacar de alguna manera las debilidades o limitaciones de este modelo. Lo que pretendía garantizar la integridad de la información y enfrentar la reducida escalabilidad y complejidad de los sistemas de ese momento.

El estudio de procesos de negocio que se repetían de una empresa a otra y de modelos o patrones existentes para modelarlos, también debería ser parte importante de esa enseñanza. Sin embargo, ir adquiriendo ese conocimientos ha sido, prácticamente, responsabilidad individual de cada analista, para indagar, aprender y hacer uso esas prácticas, durante el desarrollo de su carrera profesional.

El auge del internet cambió al mundo y además el concepto que se tenía de bases de datos. Las empresas se enfrentaron a otros niveles de demanda de información y competencia, lo que derivó en cuestionar el modelo entidad relación mismo y desarrollar otro tipo de bases de datos para satisfacer esa nuevas necesidades de forma eficiente.

A estas nuevas bases de datos emergentes con modelos de datos diversos, se les agrupó con el nombre de bases de datos NoSQL (Not Only SQL), ya que se caracterizan por no seguir el modelo relacional tradicional y solo algunas utilizan algunas implementaciones similares lenguaje SQL. Algunos ejemplos de bases de datos NoSQL son: MongoDB, CouchDB, Cassandra, Redis, Neoj, OrientDB, ArangoDB, entre otras.

Actualmente muchas empresas optan por utilizar un mix de bases de datos tanto SQL, como de tipo NoSQL, según los requerimientos del sistema o servicio. A esto se le conoce como Persistencia Políglota. De esa manera se puede obtener los mejor de ambos mundos, además se puede asegurar la integridad de los datos en aquellas áreas críticas que requieran un control de transacciones e integridad, referencia del modelo relacional.

Además se puede asegurar la escalabilidad horizontal, el manejo de enormes cantidades de datos y así evitar cuellos de botella y la flexibilidad de almacenamiento, entre otros beneficios del modelo NoSQL

Como se mencionó, las bases de datos NoSql por lo general no poseen una estructura de datos definida, por lo que su modelado se facilita. Incluso, en algunos casos, ni siquiera es necesario modelar la base de datos por separado; existen librerías para la desarrollo de aplicaciones que permiten

conectarse y generar los objetos de la base de datos a demanda, al ser utilizados por los objetos de estas aplicaciones. Es decir, que únicamente es necesario diseñar y desarrollar las aplicaciones siguiendo su modelo propio, orientado a objetos y agregar una serie de atributos a las clases para indicar cómo persistirán los objetos.

El Estándar

En general, un patrón o modelo es algo que intenta ser una guía de algo más. La palabra universal se define en el diccionario como algo que es aplicable a una gran variedad de usos. Por lo tanto, los Modelos Universales de Datos UDM (por sus siglas en inglés - Universal Data Models) son guías reusables que proveen un machote para crear modelos de datos de temas muy frecuentes o universales.

Un estandard es algo que se ha decidido utilizar de forma generalizada, ya que representa una solución estudiada y probada para facilitar y hacer más barato aquello en lo se utiliza. En la manufactura, por ejemplo, la estandarización cumple un papel muy importante en la reducción de costos, partes y componentes intercambiables. También en las formas de producir, en las características de productos para su transporte, entre otros.

Los UDM, son estándares de Modelos de Datos que permiten reducir el tiempo dedicado al modelado de datos y a la vez al desarrollo de software en general. Ya que permite reutilizar modelos, usarlos como referencia o como componente de otros, al evitar volver atrás a corregir algún problema de diseño, en otras palabras, permiten evitar reinventar la rueda.

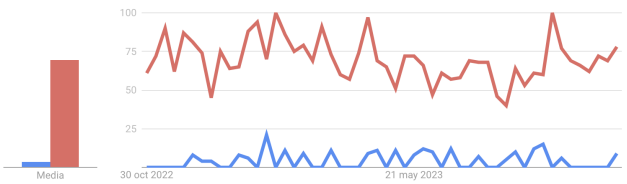
Lo importante es que existen esos UDM, personas con amplio conocimiento y experiencia, tanto en modelado de datos como en procesos de negocio, que se han tomado el tiempo de desarrollarlos y/o recopilarlos, organizarlos y probarlos.

En la serie de libros “The Data Model Resource Book” del Len Silverston, el autor explica los fundamentos de los UDM y describe varios de estos modelos que van desde el uso genérico como los modelos “Party”, “Role” y “Status” (por mencionar algunos que se utilizan en otros modelos), hasta modelos específicos para diferentes procesos e industrias como Manufactura, Telecomunicaciones, Salud, etc.

III. PROBLEMA

Los Modelos Universales de Datos parecieran ser, a todas luces, el camino a seguir en el diseño de bases de datos. Sin embargo surge el cuestionamiento de ¿por qué la mayoría de personas encargadas de desarrollar sistemas de información no los utilizan o del todo los desconocen? ¿Por qué razón no son tan conocidos y recomendados como los Patrones de Diseño de Software? Solo hace falta consultar en un buscador de internet acerca de ambos temas para identificar las diferencias de popularidad.

El siguiente gráfico muestra la comparación de la cantidad de búsquedas para los Modelos Universales de Datos y los Patrones de Diseño de Software en todo el mundo, en los últimos 12 meses.



Fuente: Google Trends

Ante este análisis surgen una serie de interrogantes: ¿Por qué no disfrutar de las ventajas que implica seguir estándares en la producción de sistemas de información y utilizar machotes preestablecidos al menos para utilizar de base en el modelado de datos?

¿Por qué se prefiere “reinventar la rueda” constantemente, si ya se analizó que las similitudes de las empresas y sus procesos son mayores que sus diferencias?

¿Por qué no se cree que sean tan útiles después de todo? Si se concibiera así, sería un tema por desarrollar en la formación académica de las universidades y además se incluiría en las entrevistas de trabajo.

Para poder responder a estas interrogantes es necesario referirse a la experiencia de quien desarrolla este análisis, ya que fue lo que en primera instancia, motivó la creación de este artículo, tomando en cuenta que no se cuenta con muchos documentos sobre este tema en específico.

A pesar de que las conclusiones que se obtienen son generales y de sentido común, este análisis se complementará con la ayuda de un concepto conocido como desajuste de impedancia, que permite explicar la dificultad de traducir o mapear el modelo relacional de la base de datos, con el modelo orientado objetos de la aplicación.

Complejidad

En el mundo del desarrollo de software existe un mantra que se transfiere de generación en generación y que es lo que ha permitido asegurarle a los desarrolladores noches tranquilas y el disfrute de su tiempo libre. Este mantra en realidad es un acrónimo muy sencillo de recordar, no solo por lo que representa para los desarrolladores de sistemas, sino, también, porque es el nombre de uno de los grupos de rock más famosos que ha existido, KISS (beso en español).

A diferencia de la banda, que cuando les han preguntado el significado de su nombre, lo han dejado a interpretación de cada uno. Pero en el contexto de tecnologías de la Información, KISS sí posee un significado definido, ya que son las siglas de una reconocida buena práctica de programación que en inglés representan el estatuto, Keep It Simple Stupid (Mantenlo las cosas simple, estupido, en espanol) que es bastante autoexplicativo.

La parte de “estupido” puede sonar un poco fuerte, pero es solo una forma de hacer énfasis en la idea de querer “complicarse uno mismo y por ende afectar a los demás” sin que haya una necesidad de peso, lo cual no es hacer algo muy astuto.

Queda claro que la complejidad es algo que se quiere evitar a toda costa, se considera que es como una bola de nieve que

tiende a crecer y puede aplastar, lo cual se considera que es la mayor debilidad de los UDM. Ya que se ha intentado en varias ocasiones poner en práctica los UDM y no se ha tenido un resultado exitoso, dejando como principal responsable de este fracaso a la anteriormente citada: la complejidad.

La principal motivación para adentrarse en el tema de los UDM ha sido realizarlo a través de la puesta en práctica.. Era un esfuerzo que prometía ahorro de tiempo y la certeza de diseñar bases de datos correctamente en el futuro. La idea no era simplemente construir la base de datos, sino también la aplicación o librerías que utilizarían esos modelos, por lo que hay considerar no solo el modelado de datos, sino también la implementación tanto en la base de datos como en la aplicación que la utiliza. Para esto, se propone desarrollar una aplicación web en .Net y SQL server como base de datos.

Curva de Aprendizaje

A pesar que los UDM no constituyen algo que se considere difícil de entender o aprender, si es necesario dedicarle tiempo para investigar y comprender los conceptos y lógica que están detrás. Se introduce el concepto de Party, el cual toma en cuenta a las personas y las organizaciones como partes activas de los sistemas y que pueden ser tratadas de forma indistinta en la mayoría de los casos. Además que estas asumen diferentes Roles según sea requerido e interactuando por medio Relaciones temporales, estos son algunos ejemplos de la lógica de abstracción propia de los UDM.

Por lo tanto es necesario lograr que todas las personas involucradas en el desarrollo del sistema, entiendan los UDM. Que se comprenda que la utilización de los UDM conlleva una curva de aprendizaje adicional en comparación a si no se siguiera ningún patrón específico de modelado de datos.

Implementation

Para esta investigación, se inicia investigando el tema a través de internet, lo que permite determinar la escasez de contenidos acerca de dicha temática. La idea inicial era buscar un ejemplo práctico que se pudiera utilizar como base, sin embargo no se logra encontrar, por lo que se decide desarrollar un proyecto para implementarlos desde cero.

Para esto, se propone desarrollar una aplicación Web API en .Net C# y SQL Server como base de datos. Básicamente, una aplicación de Backend que permitiera realizar las operaciones básicas de manejo de datos, lo que motivó a realizar una investigación exhaustiva acerca del tema.

Se recopiló en dicha investigación una serie de libros que serían utilizados más adelante para el desarrollo del proyecto. Se trata de la serie de libros “The Data Model Resource Book” de Len Silverston, volúmenes 1, 2 y 3, los cuales se convirtieron en la referencia final.

Durante la implementación de UDM en la base de datos, se percibe como poco a poco empieza a aumentar la cantidad de tablas, relaciones, restricciones, campos en llaves entre otros, esto en comparación con la cantidad con la que se trabaja normalmente. Lo que provocó que el tiempo de desarrollo de diseño se fuera incrementando, lo que dificultó avanzar con el proyecto en el tiempo establecido para este fin.

A pesar de que no se logra terminar la implementación que buscaba, en su totalidad, se logra experimentar algunos de los problemas de complejidad asociados a su puesta en práctica. Durante el proceso de desarrollo es necesario agregar y consultar datos de prueba. Ese aumento de la cantidad y la complejidad de los objetos de base de datos mencionados previamente, aumenta la dificultad para insertar, actualizar, eliminar y consultar datos. Esto significa sentencias SQL más complejas que derivan en un mayor tiempo de elaboración y riesgo de error. Esto también puede afectar negativamente el desempeño de la aplicación al manejar mayores cantidades de datos.

Desajuste por Impedancia objeto-relacional

Como se mencionaba anteriormente se han dado muchos cambios importantes en el desarrollo de aplicaciones; cambios de paradigma que llevan desde la programación procedural hasta la orientada a objetos, muchos lenguajes de programación han nacido y otros han sido reemplazados, además diferentes modelos de arquitectura se han posicionado como tendencia de desarrollo.

Sin embargo, la mayoría de estos cambios han estado del lado de la programación, del lado de las bases de datos, el cambio ha sido menor. Hasta hace relativamente poco, el modelo relacional/SQL ha sido considerado estándar de facto.

El desajuste por impedancia o como es más conocido a nivel mundial Impedance Mismatch, consiste en el problema o falta de concordancia objeto-relacional, lo que consiste en un grupo de dificultades y problemas técnico-conceptuales a los que se enfrentan los diseñadores de bases de datos y los programadores. Estos problemas son generalmente la incompatibilidad de las estructuras y tipos de datos entre los tipos de datos de las bases de datos y los tipos de datos del lenguaje de programación que se utilice en determinado momento.

En el pasado, en la década de los 90s, se consideraba que era posible eliminar esas diferencias si se reemplaza las base de datos relaciones con bases de datos que repliquen el modelo en memoria en disco. Esa década se destacó por un auge de lenguajes de programación orientados a objetos y con ellos las base de datos orientadas a objetos, ambos proponiendose como los nuevos estándares del desarrollo.

Sin embargo, la historia se encargaría de mostrar que sólo los lenguajes de programación lograrían ese objetivo y las bases de datos orientadas a objetos se irán desvaneciendo con el tiempo.

Actualmente los programadores tratan de disminuir esas diferencias por medio de la utilización de librerías de acceso a datos que permiten automatizar esa tarea y mapear los modelos objeto-relacional como Hybermate, iBATIS y Entity Framework-ADO.Net.

El problema es que los UDM están creados en el modelo de datos relacional, constan de tablas y relaciones y al introducir una mayor complejidad de ese lado, el problema de desajuste por impedancia se hace mayor.

No se puede separar los datos de los objetos, la base de

datos de los programas que las utilizan, en el desarrollo de sistemas de información. En el desarrollo del proyecto fue complejo mapear el modelo de datos con las estructuras de herencia de la clases orientadas a objetos, se tuvo que tratar de simplificar ciertos aspectos a fin de hacer la tarea un poco más llevadera. Un ejemplo de esto, fue la de mapear entidades de tipos que en UDM son tablas, a tipos de datos enumerables en C#.

Desconocimiento del negocio.

Un factor que puede influir en el hecho que los UDM no sean tan populares, es el desconocimiento de las similitudes entre las empresas que se mencionaron al inicio. Se tiende a pensar que cada sistema es único y por ende no se busca una referencia a seguir o reutilizar. Ese conocimiento no es trivial y por lo general no se aborda en las carreras de sistemas.

Durante el desarrollo de la investigación, influye de forma positiva el conocimiento en Ingeniería Industrial e Ingeniería de Software de quien investiga y el hecho de concebir que ese conocimiento no es trivial, fue lo que motivó a investigar sobre el tema.

Microservicios

Los microservicios son un enfoque o estilo de arquitectura para el desarrollo de software que está en tendencia actualmente. La aplicación se compone de pequeños servicios independientes que se comunican a través de API bien definidas. Los propietarios de estos servicios son equipos pequeños independientes, incluso con sus propias bases de datos. Este enfoque está en contraposición con el modelo monolítico, en el que todos los procesos están estrechamente asociados y se ejecutan como un solo servicio, y normalmente en una sola base de datos.

Los UDM están pensados para el enfoque monolítico, en la integración y normalización de los datos a través de todos los modelos o sistemas de la empresa. Eso no quiere decir que no se puedan aplicar parcialmente o con un grado de redundancia en cada uno de los microservicios o que los microservicios puedan compartir una base de datos. Se reconoce que la tendencia es otra, lo que afecta sin lugar a duda a que se popularicen los UDM entre los desarrolladores.

Los Microservicios están pensados principalmente para los sistemas que requieren una interacción con el cliente, por lo tanto requieren de una alta disponibilidad y escalabilidad.

Por otro lado, los sistemas ERP utilizados a lo interno de las organizaciones, todavía requieren la integración que ofrecen los UDM.

IV. SOLUCIÓN NoSQL

Como se mencionó anteriormente, a pesar de que los microservicios son una tendencia de arquitectura muy fuerte en la actualidad y que están en contraposición al modelo monolítico propio de los UDM. Esto no quiere decir que no se requieren sistemas integrados a lo interno de las organizaciones como los CRMs y ERPs. o incluso que se puedan adaptar o usar parcialmente en los microservicios. Por lo que no representan una amenaza o limitación para los UDM.

La curva de aprendizaje no es algo de lo que se pueda escapar fácilmente al tratar de implementar o replicar cualquier nuevo tipo de técnica o patrón. Los UDM son solo un ejemplo más de eso. Los patrones de diseño de software, con los cuales se están comparando los UDM, también requieren de un estudio exhaustivo y de práctica para su adecuada utilización. Por lo tanto, podríamos decir que no son un problema en sí mismo.

Tal vez la diferencia radica, en que por lo general, los profesionales de desarrollo de software ya traen un conocimiento previo de patrones de diseño de software, estudiado en las universidades, cosa que no ocurre con los UDM. Esto podría estar asociado a otro problema mencionado, el poco conocimiento, por parte los profesionales de software, sobre aquello que se desea modelar o abstraer; en este caso, los procesos del negocio, de la empresa. Como se va a creer que algo es importante enseñar, o por lo menos recomendar, si se desconoce.

Esta falta de conocimiento, hace que no se vea la gran similitud que existe entre las empresas y la necesidad de seguir patrones en lugar de “reinventar la rueda” cada vez. Sin embargo, es responsabilidad última del desarrollador, asegurarse de entender bien aquello que va a modelar.

Por otro lado, no se puede culpar a la academia, las universidades o el profesorado que forma a los y las estudiantes por no desarrollar un tema como este. No se puede esperar que dediquen recursos y enseñen un elemento que no ha demostrado ser lo que promete y que por el contrario resulta complicado, por lo que nadie lo utiliza.

Parece ser que ahí está la clave, lo que se citó al inicio de este análisis. Y de nuevo surgen las interrogantes:

¿El problema principal es la complejidad que trae la implementación de UDM? ¿Estos problemas se solucionarán el resto de problemas o limitaciones si se corrige el problema principal? ¿Existirá la oportunidad de que los UDM logren llegar a ser una solución aceptada, recomendada y utilizada de forma habitual? ¿Si la complejidad es el problema a solucionar, cuál será la causa raíz que se debe atacar?

Como análisis del problema, desde esta investigación la única causa que se ha podido pasar la prueba de un segundo análisis, es la del modelo de datos, el uso del modelo de datos relacional SQL y su incompatibilidad con los lenguajes de programación orientados a objetos de las aplicaciones, hacen posible la interacción con los usuarios de una forma u otra.

Por lo anterior, en este artículo se va a enfocar en presentar una propuesta de solución al problema de la complejidad en la implementación de los UDM; y la manera en que se va a abordar ese problema, será atacando lo que se considera, aquí, hasta ahora, la causa principal de esa complejidad, Desajuste por Impedancia objeto-relacional.

Propuesta

Lo que se propone, básicamente, es reemplazar el modelo de datos utilizado hasta ahora por los UDM, el modelo relacional. En su lugar se sugiere seguir un modelo de datos

no estructurado, algo que no existía, o por lo menos no se concibe, en el momento en que fueron desarrollados.

Este cambio de modelo se haría utilizando una base de datos NoSQL, ampliamente utilizadas en la actualidad. Específicamente, se propone una que utilice un modelo de datos de documento, ya que esto, además de disminuir la cantidad de entidades y relaciones necesarias al combinarlas varias en un solo documento, también podrá ajustarse mejor al modelo orientado a objetos de los lenguajes de programación por seguir un patrón jerárquico de propiedades, sin mencionar que es el modelo de datos más utilizado por los API de las aplicaciones cliente y servidor.

Queda claro que esto reduciría la complejidad, sin embargo es necesario validar o probar que es posible, que no se va a encontrar con un obstáculo no considerado en el camino. De nada sirve hacer una propuesta, si esta es inviable o imposible de implementar. Así que no queda otra que embarcarse nuevamente en un proyecto de implementación de UDM por motivos didácticos.

¿Qué es lo peor que puede pasar? ¿qué abonado a los avances de la AI, permite crear una plataforma global que termine ayudando a el dominio absoluto de las máquinas sobre la humanidad?

Implementción

El objetivo de esta implementación no es crear una aplicación completa y que sea un ejemplo de utilización de las mejores prácticas, sino, desarrollar algo básico, pero funcional, que permita demostrar la factibilidad de utilizar una base de datos NoSQL para implementar UDM.

Para lograr esto se decidió desarrollar una aplicación WebAPI, un web service, pero que para nuestros propósitos, no es más que una aplicación "back end" que nos va permitir interactuar con la base de datos y exponer la funcionalidad sin a necesidad desarrollar una interfaz de usuario o "front end".

A la interfaz de usuario no le interesa que haya detrás de los métodos expuestos por el API, le da lo mismo si lla implementación sea SQL o NoSQL. Es por esta razón que no forma parte de los esfuerzos y análisis de esta propuesta.

No obstante, hay herramientas como Swagger[2], por ejemplo, que generan una documentación del API en el navegador que no sólo permite consultar todos los endpoints de la aplicación, sino que también, permite probarlos inmediatamente en acción enviando una petición y recibiendo una respuesta. De esta forma se tiene una interfaz de usuario básica pero suficiente para cumplir con el objetivo.

El código de la aplicación está alojado en la plataforma Github de manera pública y libre en el repositorio Cenfotec-Paper-UDM y se puede descargar en la siguiente dirección:
https://github.com/machocr/Cenfotec-Paper-UDM

A. Lenguaje de Programación

Para el desarrollo de esta aplicación, se determinó como lenguaje de programación C# de la plataforma Net de Microsoft, un lenguaje ampliamente utilizado y orientado a objetos. Como se mencionó anteriormente, este paradigma de programación es el estándar utilizado para el desarrollo de sistemas de información en la actualidad y presenta incompatibilidades con el modelo de datos relacional que añaden complejidad a la implementación de los UDM.

Documentación interactiva de la aplicación

El criterio para la escogencia de este lenguaje de programación, además de lo mencionado anteriormente, no es otro que la familiaridad con su uso de quien realiza la investigación. Esta experiencia previa no es algo menor, ya que ha permitido conocer las librerías externas que se utilizarán para la comunicación e interacción con la base de datos que se mencionará en el siguiente apartado.

[2] https://swagger.io/

B. Base de Datos

La base de datos constituye el elemento principal de esta implementación y el principal criterio a considerar es que sea de tipo NoSQL y orientada a documentos, ya que ese es el modelo escogido para la propuesta.

Se decidió utilizar MongoDB[3] ya que cumple a cabalidad con estos requerimientos y es una base de datos bastante representativa dentro del grupo de bases de datos NoSQL según el índice DB-Engines Ranking[4]. Cuenta con una popularidad creciente desde 2015, lo que la sitúa entre las bases de datos más utilizadas actualmente a nivel general, como se observa en el siguiente video que muestra esta evolución: https://www.youtube.com/watch?v=bRyCjIX2rXg

Además, se considera que de parte de quien realiza esta investigación, se posee experiencia con el lenguaje de programación, ya que se ha utilizado un par de veces en proyectos personales.

C. Comunicación

Cuando se habla de la comunicación, se hace referencia a los mecanismos de conexión e interacción entre las bases de datos y la aplicación. Existen diferentes opciones de librerías o componentes desarrollados por tercera partes para cumplir ese cometido.

No se va a hacer una comparación de las opciones disponibles, ya que como se mencionó, el objetivo no es desarrollar la aplicación perfecta, por lo tanto el criterio utilizado es nuevamente, la experiencia de uso y popularidad, y en este caso en particular, la conveniencia..

A estas librerías de conexión de bases de datos se les conoce como drivers o controladores. La plataforma .Net posee componentes nativos para manejar la base de datos SQL Server, la cuál pertenece el ecosistema de Microsoft. Para el resto de bases de datos, sus drivers deben ser desarrollados por terceros, incluidos los mismos desarrolladores de las bases de datos.

Para efectos de esta investigación, se utilizó la librería oficial de MongoDB, MongoDB C# Driver[5]. Aunque se prefiere utilizar drivers oficiales, ya que están desarrollados por personas que están más familiarizadas con la base de datos y se asegura la continuidad de desarrollo y su mantenimiento. Esta librería se incorpora al proyecto como una dependencia externa descargada del repositorio Nuget[6].

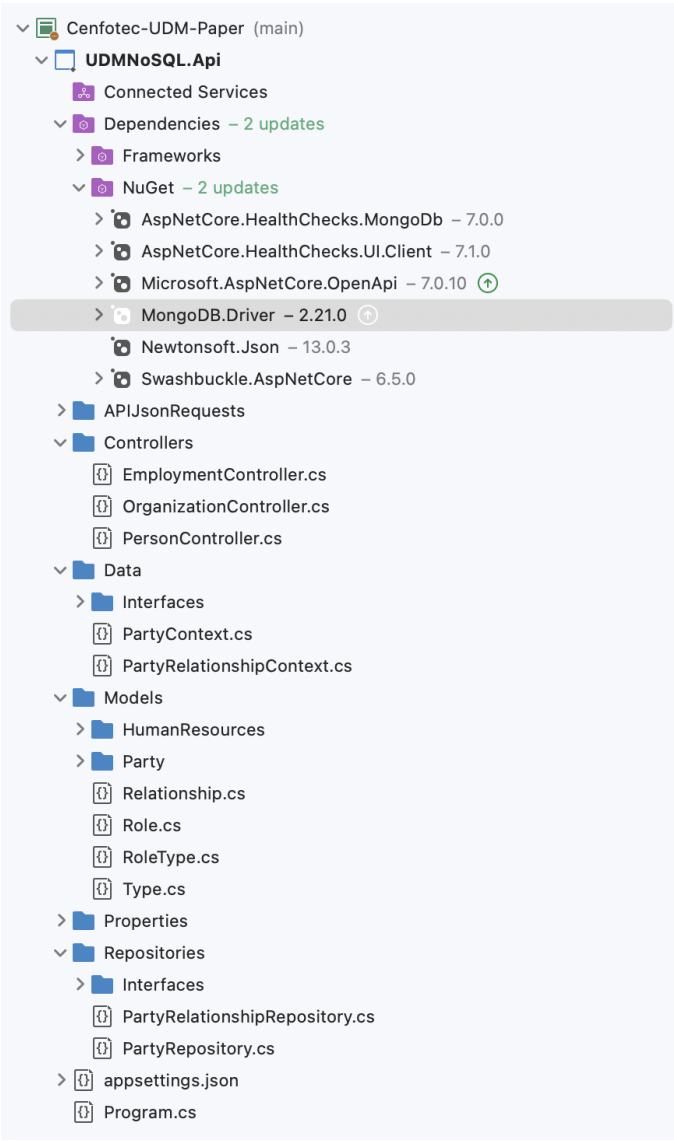
En esta propuesta, el papel de driver adquiere una relevancia significativa. No es exagerado decir que es el responsable, en mayor grado, de disminuir la complejidad de implementar UDM.

Esta librería, permite serializar y deserializar objetos o instancias de la aplicación, directamente en la base de datos. Esto significa que, si se decora una clase, la representación en código de un objeto de la aplicación con los atributos adecuados, al correr o ejecutar la aplicación, se generará la colección, en bases de datos, que almacenará los objetos en formato de documento.

Esto hace que las operaciones de consulta y almacenamiento en la base de datos sean transparentes para el desarrollador, permitiendo así, enfocarse principalmente en las las clases y funcionalidad de la aplicación, ahorrar tiempo de diseño y construcción de la base de datos y eliminar de cuajo los efectos del desajuste por impedancia objeto-relacional. Cuando se expliquen las clases desarrolladas se detalla más sobre la serialización.

C. Estructura del Proyecto

Como se mencionó anteriormente, se creó un microservicio. En la siguiente imagen se muestra el proyecto Web API en el IDE Microsoft Visual Studio.



Estructura de la Aplicación

Es un servicio sencillo que se limita a exponer la funcionalidad de consulta y almacenamiento a datos. Adicionalmente al código generado por parte de Visual Studio

[3] https://www.mongodb.com/
[4] https://db-engines.com/en/ranking
[5] https://www.mongodb.com/docs/drivers/csharp/current/
[6] https://www.nuget.org/

a partir del machote del tipo de proyecto, se crearon cuatro directorios de archivos que se describen a continuación.

1) **Controllers**

Como se puede intuir su nombre, este directorio y espacio de nombres, contiene los controladores del API, Api Controllers. Estos controladores son clases que se especializan en manejar las solicitudes de información HTTP entrantes del servicio y enviar respuestas a los clientes o emisores de esas llamadas. En otras palabras, los controladores permiten mapear los URLs de los llamados a los métodos públicos, de la clase llamados "Actions" que se encargan de devolver los datos serializados en el formato solicitado, en este caso documentos Json.

Como se puede intuir su el nombre, este directorio y espacio de nombres, contiene los controladores del API, Api Controllers. Estos controladores son clases que se especializan en manejar las solicitudes de información HTTP entrantes del servicio y enviar respuestas a los clientes o emisores de esas llamadas.

Básicamente, es la interfaz que expone la funcionalidad del servicio. Para este proyecto se crearon tres controladores:

- EmploymentController
- OrganizationController
- PersonController

Este microservicio, específicamente, es un servicio de Recursos Humanos, por lo tanto, expone funcionalidad relacionada a este tema. EmploymentController, expone funcionalidad relacionada a este tema. El controlador EmploymentController permite consultar, crear, actualizar, eliminar y terminar empleados, entre otras acciones.

Como se verá más adelante, las organizaciones y personas son entidades abstractas, esto quiere decir que no existen por sí solas, existen solo en la medida que se asocian a un empleador y a un empleado, respectivamente.

Por lo que surge el cuestionamiento ¿ Por qué se crearon controladores para estas entidades (OrganizationController y PersonController), entonces?

Como se describió previamente, el objetivo último de esta implementación es probar la factibilidad de adecuar los UDM a NoSQL, por lo tanto, únicamente, se agregan estos controladores con el fin de probar la funcionalidad de esos modelos, aunque, en un servicio real, esos controladores no existirían.

2) **Data**

Data contiene las clases e interfaces que definen y configuran las colecciones (especie de tablas en NoSQL), de la base de datos. A estas clases se le llama contextos y se le agrega el sufijo "Context".

- PartyContext
- PartyRelationshipContext

Como se puede ver, únicamente, hay dos clases en este directorio, eso quiere decir que en la base de datos solo hay dos colecciones Party y PartyRelationship.

A diferencia del modelo relacional, en el que es necesario crear una gran cantidad de tablas para implementar los UDM, en este caso, gracias a que la base de datos carece de una estructura de datos definida, y que la orientación a objetos de la aplicación permite representar mejor la herencia inherente de los UDM. Además que es posible serializar y deserializar los objetos de forma directa, se hace necesario una menor cantidad de objetos de bases de datos.

La entidad "Party" o parte, por ejemplo, es la clase padre, de la que heredan las organizaciones y personas, por lo que es posible almacenar todas esos objetos hijos junto con sus componentes en la misma colección de bases de datos. En el modelo relacional se requiere una tabla y relaciones para cada tipo de objeto y sus componentes.

3) **Models**

Esta carpeta contiene los nuevos UDM. Gracias a que el driver de conexión de MongoDB permite generar las colecciones de base de datos automáticamente, los modelos se diseñan directamente en la aplicación. Únicamente es necesario crear los modelos en la aplicación usando la orientación a objetos que provee el lenguaje de programación, en este caso C#.Net. Prácticamente, no se puede olvidar de un diseño en la base de datos.

En esta propuesta se ha decidido implementar una parte representativa de los UDM, ya que tratar de hacerlo para su totalidad requiere escribir un libro en varios volúmenes.

Los modelos seleccionados fueron los del capítulo 1, "Personas y organizaciones" (Party) y "Recursos Humanos" del libro xxx de xxx. Party es una entidad que se utiliza en la mayoría de los UDM, y RRHH va a permitir observar una aplicación real de los UDM.

Estos modelos se agruparon en subcarpetas "Party" y "PartyRelationship" únicamente por motivos de orden. Más adelante se van a detallar más sobre estos modelos.

4) **Repositories**

El patrón repositorio se encarga de encapsular y centralizar la lógica relacionada con la persistencia de los datos.

No se va a dedicar mucho a explicar este apartado, ya que que no constituye un elemento relevante sobre el estudio de los UDM y su implementación NOSQL, únicamente se menciona que estas clases contienen los métodos que ejecutan los métodos CRUD por medio de los contextos de las colecciones mencionadas anteriormente en el apartado "Data".

- PartyRepository
- PartyRelationshipRepository

C. UDM - Clases

La carpeta Models contiene, lo que se puede considerar, como los Modelos Universales de Datos que se implementan. Como se mencionó, se enfoca únicamente en la estructura de clases de los modelos de la aplicación y dejarle la generación de objetos de bases de datos al driver de conexión.

Así que, en realidad, se puede dejar incluso dejar de hablar de UDM y empezar a hacer referencia a una especie de UBC, Componentes Universales de Negocio, ya que no son modelos de datos.

Se trata de una implementación general que se puede usar directamente o ajustar si se quiere, ya sea cambiando las clases o utilizando mecanismos de orientación de objetos para escribir lo que se desee cambiar, agregar nueva funcionalidad, o esconder propiedades u otros miembros que no formen parte de los requerimientos.

Para efectos de la investigación fue posible pasar de más de 40 tablas del modelo relacional a dos colecciones de modelo NoSQL de documento. Una reducción bastante considerable, sin mencionar que la generación de la base de datos y la interacción con ella, es prácticamente transparente para el desarrollador, lo que permite reducir el tiempo y esfuerzo en tareas de diseño, implementación y mantenimiento de las bases de datos. También se reduce la complejidad de acceso y almacenamiento de datos, incluso la posibilidad de errores o pulgas de programación se reducen, al reducir la interacción humana. Pero, ¿cómo se logra eso? Eso es lo que se va a responder aquí.

Como se dijo, en esta implementación se desarrollaron modelos para los UDM mostrados en los capítulos 2: “People and Organizations” y 9: “Human Resource” del libro “The Data Model Resource Book”, Volume 1, “A library of Universal Data Models for all Enterprises” de Len Silverston.

En el repositorio se puede ver cada una de las clases de los modelos desarrollados. También incluye una página web donde se puede ver los UDM que se van a mencionar.

A continuación se muestran los modelos de la propuesta:

- PartyRelationship
- Role
- RoleType
- Type

HumanResources /
- BenefitType
- BudgetItem
- Deduction
- DeductionType
- Employment
- EmploymentApplication
- EmploymentApplicationSourceType
- EmploymentApplicationStatusType
- PartyBenefit
- Paycheck
- PayGrade
- PayHistory
- Payment
- PaymentMethodType
- PayrollReference
- PerformanceNote
- PerformanceNoteType
- PerformanceReview
- PerformanceReviewItem
- PerformanceReviewItemType
- PeriodType
- Position
- PositionClassificationType
- PositionFulfillment
- PositionReportingStructure
- PositionResponsability
- PositionStatusType
- PositionType
- PositionTypeClass
- PositionTypeRate
- RateType
- Receipt
- ResponsabilityType
- SalaryStep
- TerminationReason
- TerminationType
- UnemploymentClaim
- UnemploymentClaimStatusType
- ValidResponsability

Party /
- ContactMechanism
- ContactMechanismType
- Contractor
- EEOCClassificationType
- ElectronicAddress
- Employee
- GenderType
- IncomeClassificationType
- IndustrylassificationType
- InternalOrganization
- MaritalStatusItem
- MaritalStatusType
- MinorityClassificationType
- Organization
- Party
- PartyContactMechanism
- PartyRelationship
- PartyRelationshipStatusType
- PartyRole
- PartyType
- Person
- ostalAddress
- SizeClassificationType
- TelecommunicationNum

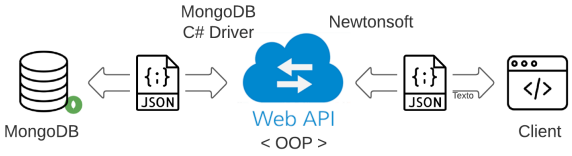
Esta implementación no pretende ser la única, ni la mejor, es solo una propuesta, es una interpretación de cómo se podría llevar a la práctica la idea de modelos o patrones universales de negocio utilizando una base de datos NoSQL, para aprovechar todas las ventajas que ofrecen.

Se va a describir lo que se hizo de forma general y se va a detallar, únicamente, en los modelos más representativos, para no extender de más este paper.

Serialización/Deserialización a Documento

La serialización[7] es un proceso mediante el cual se puede convertir objetos de un programa POO[8] en ejecución en flujos de bytes o texto capaces de ser almacenados en bases de datos o de ser enviados a través de la red y posteriormente, ser capaces de reconstruirlos donde sea necesario.

La ventaja más importante está relacionada con la eliminación del desajuste por Impedancia objeto-relacional, al lograr serializar y deserializar directamente los objetos de la aplicación al formato de documento utilizado en la base de datos y en la comunicación del API. Eso permite eliminar una capa adicional para realizar este mapeo y se puede almacenar el objeto completo en una sola colección de la base de datos, incluidas los miembros de objeto y listas de objeto.



Serialización/Deserialización

El siguiente es el documento JSON obtenido para el modelo "Persona" de la aplicación.


```
{
  "_t": "personClass",
  "Type": "Person",
  "RoleList": [
    {
      "_t": "employeeClass",
      "Type": "Employee"
    },
    {
      "_t": "contractorClass",
      "Type": "Contractor"
    }
  ],
  "ContactMechanismList": [
    {
      "ContactMechanism": {
        "_t": "postalAddressClass",
        "Type": "PostalAddress",
        "Country": "US",
        "State": "FL",
        "City": "Miami",
        "PostalCode": "11368",
        "Address1": "967 west st",
        "Address2": "",
        "Directions": ""
      },
      "FromDate": {
        "$date": "2023-09-19T20:03:24.799Z"
      },
      "ThruDate": null,
      "NonSolicitationIndicator": true,
      "Comment": "Postal Address test"
    }
  ],
  "FirstName": "Luis",
  "LasttName": "Ruiz",
  "MiddletName": "Alberto",
  "MothersMaidenName": "Arauz",
  "PersonalTitle": "Mr",
  "Sufux": "",
  "Nickname": "Macho",
  "BirthDate": {
    "$date": "1975-05-28T06:00:00.000Z"
  },
  "Gender": "Male",
  "Heigh": 1.85,
  "Weigh": 90,
  "SocialSecurityNo": "1234567890",
  "PassportNo": "0987654321",
  "PassportExpireDate": {
    "$date": "2025-09-19T06:00:00.000Z"
  },
  "TotalYearsWorkExperience": 15,
  "Comment": "Test Person",
  "MaritalStatusList": [
    {
      "Type": "Married",
      "FromDate": {
        "$date": "2015-09-19T20:03:24.799Z"
      },
      "ThruDate": null
    }
  ],
  "EEOCClassification": "Hispanic",
  "IncomeClassification": "From20kTo50k"
}
```


JSON[9] (JavaScript Object Notation) es un formato estándar de texto utilizado principalmente para transmitir datos entre un servidor web y una aplicación web (API). Inicialmente extraído de JavaScript, actualmente es independiente del lenguaje de programación. En MongoDB se utiliza el formato BSON[10] para almacenar los documentos, que no es más que una extensión de JSON que se representa de manera binaria.

Nótese cómo en un solo documento incluye propiedades de datos como nombre y fecha de nacimiento, el objeto innumerable como género, de lista de objetos como la lista de roles, mecanismos de contacto, historia de estatus maritales entre otros..

En el siguiente link se muestra como se realiza el mapeo de un objeto de negocio del modelo relacional a formato JSON

https://www.ibm.com/docs/es/baw/20.x?topic=formats-javascript-object-notation-json-format

Herencia

C# es un lenguaje de programación orientado a objetos, un paradigma de programación que se basa en la definición de clases y objetos para representar entidades del mundo real.

La herencia es uno de los principios básicos de este paradigma y permite abstraer el comportamiento de los objetos reales, donde una clase objeto padre "pasa" o hereda propiedades y funcionalidades sus hijos.

[7]https://codingornot.com/que-es-la-serializacion-o-marshalling

[8] Programación Orientada a Objetos.

[9]https://www.ibm.com/docs/es/baw/20.x?topic=formats-javascript-object-notation-json-format

[10] https://aula301.com/tipos-datos-podemos-utilizar-mongodb/

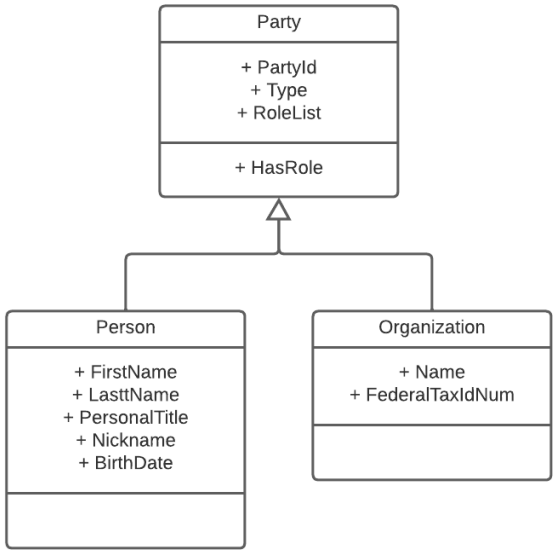
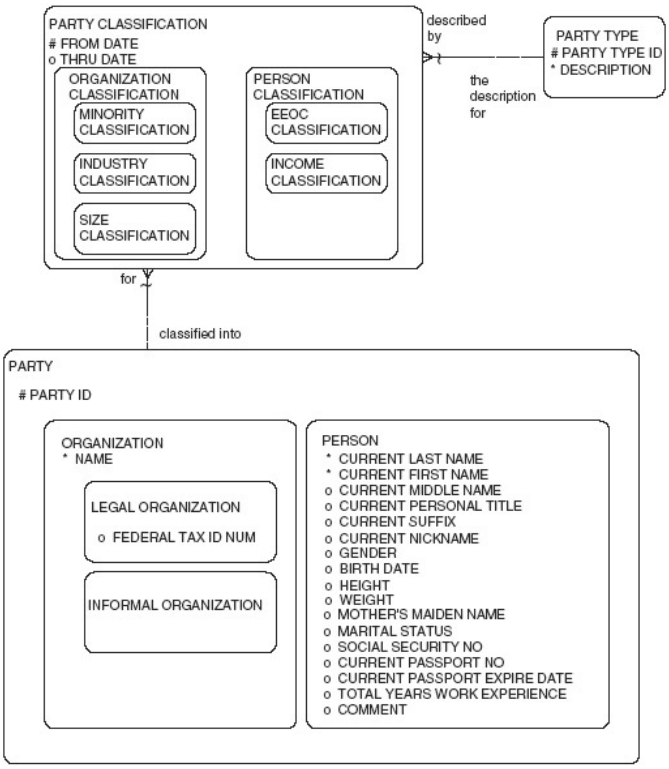


Diagrama de clases de la entidad Party



UDM de la entidad Party

De esta forma, las entidades hijas Persona y Organización son a su vez entidades Party, que es la clase padre. Dado que una colección de base de datos es solo una forma de agrupar documentos, ya que no posee una estructura de datos definida, hace lógico pensar que las entidades padres e hijas se pueden almacenar en la misma colección, a diferencia del modelo relacional en el que se necesita una tabla para cada entidad.

Este proceso de serialización y deserialización de objetos de diferentes tipos desde y hacia una colección de bases de datos, no es automática. El driver de conexión MongoDB oficial, permite mediante funcionalidad los atributos de decoración clase BsonKnownTypes y JsonDerivedType que permiten especificar las clases hijas de la clase padre. A continuación se muestra como se utilizaron estos atributos en la clases Party, Persona y Organización:

```
[JsonDerivedType(typeof(Person), typeDiscriminator:
"personClass")]
[JsonDerivedType(typeof(Organization),
typeDiscriminator: "organizationClass")]
[BsonKnownTypes(typeof(Person),
typeof(Organization))]
public abstract class Party
…

[BsonDiscriminator("personClass")]
public class Person : Party
…

[BsonDiscriminator("organizationClass")]
public class Organization : Party
…
```

Enumeradores de Tipo

En los UDM y en cualquier modelo de datos se determina que una gran cantidad de las tablas o entidades corresponden a tipos o clasificación de los objetos. Estos tipos generalmente poseen un campo identificador, un nombre y en algunos casos una descripción más amplia. Esta semejanza permite que se pueda utilizar una colección, o incluso una tabla del modelo relacional para almacenar estos tipos.

Incluso se puede crear una clase padre y derivar el resto de tipos a esa clase, de esta forma se podría utilizar una sola colección incluso para aquellos hijos que, en casos especiales, requieran algún campo o propiedad adicional.

En esta propuesta se decidió dar un paso más y eliminar por completo la necesidad de almacenar los tipos en la base de datos. Para ello, se hizo uso de enumerados (Enum) del lenguaje de programación para representar estos tipos en la aplicación. El tipo enum en C# Es un tipo de dato especial que se utiliza para declarar un conjunto de constantes enteras. Las constantes de un enum se denominan elementos y cada uno tiene un valor entero asignado por defecto, comenzando en 0 y continuando en incrementos de 1

A continuación se presenta un ejemplo del uso de enum en C# para implementar un tipo de modelo de datos. El estatus marital es un tipo, y en este caso, sus valores toman el valor entero comprendido entre 0 para “None” o ninguno, hasta 4 para “Widowed” o viudo.

```
namespace UDMNoSQL.Api.Models.Party
{
        public enum MaritalStatusType
        {
                None,
                Single,
                Married,
                Divorced,
                Widowed
        }
}
```

Cuando se usa un enum como propiedad de otro objeto o clase, se puede especificar si se desea que el valor del enum se almacene transporte como cadena de caracteres en lugar de su valor entero, para esto el miembro se debe decorar con atributos de miembro. A continuación se muestra como se agregan estos atributos a la propiedad en un Gender o género.

```
[JsonConverter(typeof(StringEnumConverter))]
[BsonRepresentation(BsonType.String)]
public GenderType Gender { get; set; }
…
```

V. CONSIDERACIONES FINALES

Durante el proceso de desarrollo se identificaron algunos puntos de mejora y oportunidades de implementación.

Se podría mover los modelos e incluso la funcionalidad de interacción a la bases de datos a un proyecto de clases o librería aparte, de esta forma se tendría un componente de Objetos Universales de Negocio UBL (Universal Business Objects) que se pueda reutilizar en otras aplicaciones y extender según las necesidades. Incluso se puede pensar en más de un componente UBD para que solo se utilicen los que realmente requiera la aplicación.

La desventaja más evidente de utilizar un modelo no estructurado NoSQL, es que no se tienen los mecanismos para asegurar la integridad y transacciones de los motores de bases de datos relacionales. Este no es un problema menor, ya que estos principios son básicos en un sistema empresarial. Por lo tanto se recomienda investigar la mejor manera de implementar estos principios directamente en la aplicación, al menos en aquellos objetos que más lo requieran. Además, se recomienda seguir un sistema de persistencia políglota, de tal manera que se pueda utilizar gestores de base de datos relacionales, en aquellos áreas que requieran un uso riguroso de transacciones o sensibles de integridad referencial.

Por otro lado, utilizar un modelo NoSQL lleva consigo varias ventajas como el facilitar la escalabilidad horizontal, que sumado a los avances tecnológicos cada vez más acelerados en el procesamiento, almacenamiento, la seguridad, y el transporte de datos, entre otros, hace posible considerar el desarrollo de un repositorio común para varias empresas, permitiendo, así, compartir información general y agilizar aún más el desarrollo de aplicaciones.

V. CONCLUSIÓN

La poca popularidad de los UDM se debe, entre otros factores, a la complejidad asociada a su implementación y uso. Esto hace que la curva de aprendizaje sea más pronunciada, agravando el problema.

Se identificó al desajuste por impedancia objeto-relacional como la causa principal de esta complejidad y se propuso una solución basada en el cambio del modelo relacional por el modelo no estructurado NoSQL.

Se demostró que sí es posible llevar a cabo una implementación NoSQL de UDM, sin embargo, en la implementación realizada se toma en cuenta la aplicación como un todo, alejándose del mero modelaje de datos, para enfocarse en el modelado de objetos.

El desajuste por impedancia objeto-relacional se elimina gracias a que es posible serializar/deserializar directamente los objetos de la aplicación en la base de datos, eliminando la capa de mapeo necesaria cuando se utiliza una base de datos relacional.

Adicionalmente a esto, se reduce considerablemente todo el trabajo relacionado con el manejo de la base de datos, disminuyendo con esto la complejidad. Aún disminuyendo la complejidad de implementación, existe una curva de aprendizaje. Se requiere estudiar la teoría detrás de los UDM, entender sus beneficios y cómo funcionan, para poder aplicarlos y extenderlos correctamente.

Se recomienda seguir un modelo de persistencia políglota para asegurar la integridad referencial y reforzar el proceso transaccional en aquellas áreas que se consideren más sensibles y que así lo requieran, o incorporar estos mecanismos en el código de la aplicación.

Para lograr que los UDM se vuelvan más populares y se empiecen a aprovechar sus beneficios, además de disminuir el problema de la complejidad, es necesario que se de un esfuerzo de divulgación y generación de contenidos o recursos, tales como, artículos, libros, videos, entre otros, que ayuden en su proceso de implementación y fomente la creación de una comunidad participativa y colaborativa que permita, además, mejorar los nuevos Objetos Universales de Negocio UBO.

REFERENCIAS

[1] Len Silverston, *The Data Model Resource Book, Vol. 1: A Library of Universal Data Models for All Enterprises*, MA, USA: Wiley, 2011, pp. 1–65, 299-333.

[2] Pramod J. Sadalage and Martin Fowler, *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*, IN, USA: Addison-Wesley, 2014, pp. 3-12.

[3] Eliyahu M. Goldratt, *The Haystack Syndrome: Sifting Information Out of the Data Ocean*, MA, USA: North River Press, 1990, pp. 12–46.

Lus A. Ruiz nació en San José-Costa Rica el 28 de mayo de 1975. Realizó sus estudios secundarios en el Liceo de Nicoya. Se graduó en la Universidad de Costa Rica como Licenciado en Ingeniería Industrial en el año 2000. En el 2003 obtuvo su título de Ingeniero de Software en la Universidad Cenfotec.
Se ha dedicado al Desarrollo de Software en el sector privado.
Áreas de Interés: Empresa y negocio, Teoría de Restricciones TOC, Desarrollo y Arquitectura de Software, Bases de Datos.

José A. Cabezas, nació en San José, Costa Rica el 4 de diciembre de 1980. Realizó sus estudios secundarios en el Colegio Marista de Alajuela. Se graduó como Ingeniero en Sistemas en el año 2005. En el 2015 obtuvo su título de Maestría en Tecnologías de Bases de Datos en la Universidad Cenfotec.
Se ha dedicado al áre de Inteligencia de Negocios, Analítica y Ciencia de Datos en el sector privado.
Áreas de Interés: análisis de datos, bases de datos, arquitectura empresarial, inteligencia artificial.