Fecha de recepción 12/05/2023
Fecha de aceptación: 11/09/2023
Pp 22 – Pp. 31

ARK: https://n2t.net/ark:/87558/tekhne.26.2.3

Formulación de un Perfil UML para la Documentación de Sistemas Informáticos Bajo el Paradigma de Programación Orientada a Protocolos

Sofía Rodríguez[1], María Pérez[2]
srodriguez.18@est.ucab.edu.ve[1], mperezpu@ucab.edu.ve[2]
Universidad Católica Andrés Bello[12]

Resumen

En la actualidad, en el desarrollo de sistemas para el entorno Apple e incluso fuera de éste, se utiliza el lenguaje de programación Swift, considerado como el responsable del surgimiento del paradigma Programación Orientada a Protocolos (POP). POP presenta mecanismos de abstracción enfocados en proveer soluciones a defectos encontrados en el paradigma de Programación Orientada a Objetos (POO). Aunque POP es un paradigma ampliamente usado; no existe un estándar para su representación dentro del Lenguaje de Modelado Unificado (UML). Adicionalmente, el nivel de madurez del lenguaje Swift, afecta procesos y herramientas dentro del ecosistema del lenguaje, como es el caso del compilador de documentación DocC. Por lo anterior, en este trabajo se propone una extensión de UML mediante la formulación de un perfil POP, definido por un conjunto de estereotipos, restricciones y valores etiquetados que representan las principales propiedades a nivel de diseño, como protocolos, estructuras y extensiones. De esta manera, se introduce una extensión formal para la representación de este nuevo paradigma en UML. Este aporte refuerza el papel de la documentación como una herramienta de descubrimiento de diseño a partir de código, además de contribuir con el avance del lenguaje Swift, permitiéndole desarrollos basados en Model Driven Engineering, (MDE) o Model Driven Development (MDD).

Palabras clave: Documentación, POP, Swift, UML, Diagrama de Perfil

Formulation of a UML Profile for the Documentation of Computer Systems Under the Protocol-Oriented Programming Paradigm

Abstract

Currently, when developing systems for the Apple environment and even outside of it, programmers use a language called Swift. This language is responsible for the emergence of the Protocol-Oriented Programming (POP) paradigm. POP presents abstraction mechanisms focused on providing solutions to defects found in the Object-Oriented Programming (OOP) paradigm. Although POP is a widely used paradigm, there isn't a standard for its representation within the Unified Modeling Language (UML). Additionally, the maturity level of the Swift language affects processes and tools within the language's ecosystem, such as the DocC documentation compiler. Therefore, this work proposes an extension of UML by formulating a POP profile diagram, defined by a set of stereotypes, constraints, and labeled values that represent the main design-level properties, such as protocols, structures, and extensions. In this way, a formal extension is introduced for the representation of this new paradigm in UML. This contribution reinforces the role of documentation as a design discovery tool from code, as well as contributing to the advancement of the Swift language by allowing developments based on Model Driven Engineering (MDE) or Model Driven Development (MDD).

Keywords: Documentation, POP, Swift, UML, Profile diagram

SOFIÍA RODRIGUEZ, MARIA PÉREZ

Formulação de um Perfil UML para Documentação de Sistemas Computacionais Sob o Paradigma de Programação Orientada a Protocolos

Resumo

Atualmente, ao desenvolver sistemas para o ambiente Apple e até mesmo fora dele, os programadores utilizam uma linguagem chamada Swift. Esta linguagem é responsável pelo surgimento do paradigma da Programação Orientada a Protocolos (POP). POP apresenta mecanismos de abstração focados em fornecer soluções para defeitos encontrados no paradigma de Programação Orientada a Objetos (OOP). Embora o POP seja um paradigma amplamente utilizado, não existe um padrão para sua representação na Unified Modeling Language (UML). Além disso, o nível de maturidade da linguagem Swift afeta processos e ferramentas dentro do ecossistema da linguagem, como o compilador de documentação DocC. Portanto, este trabalho propõe uma extensão da UML formulando um diagrama de perfil POP, definido por um conjunto de estereótipos, restrições e valores rotulados que representam as principais propriedades em nível de design, como protocolos, estruturas e extensões. Desta forma, é introduzida uma extensão formal para a representação deste novo paradigma em UML. Esta contribuição reforça o papel da documentação como ferramenta de descoberta de design a partir de código, além de contribuir para o avanço da linguagem Swift ao permitir desenvolvimentos baseados em Model Driven Engineering (MDE) ou Model Driven Development (MDD).

Palavras-chave: Documentação, POP, Swift, UML, Diagrama de Perfil

i. INTRODUCCIÓN

Un proyecto de *software* bien documentado tiene más probabilidad de éxito que uno escasamente documentado; por ello se unen esfuerzos en el desarrollo de herramientas que faciliten esta actividad, que a pesar de su importancia se le presta poca atención. Swift es un lenguaje que se basa en el paradigma de Programación Orientada a Protocolo (POP), cada vez más usado por la comunidad Apple. En este sentido, en este trabajo se propone un perfil UML para la documentación en POP. Previo a ello se hace una reflexión del contexto de esta propuesta considerando los tópicos: Documentación del Diseño y POP; luego se plantea el problema, se hace la propuesta del perfil POP y se incorpora un caso de uso. Se cierra con las conclusiones y próximas acciones

ii. ANTECEDENTES

A. *Documentación del Diseño*

Todo proceso de desarrollo debe incluir la descripción de la solución propuesta. Cuatro son los conceptos que se manejan en este contexto cuando de descripción hablamos **[1].** La **especificación,** la cual tiende a denotar una arquitectura presentada en un lenguaje formal, pero las especificaciones formales no son prácticas; ni son siempre necesarias. La **representación**, que connota un modelo, una abstracción, una interpretación de una cosa separada o diferente de la cosa misma. La **descripción**, la cual ha sido replanteada por la comunidad del lenguaje de descripción de la arquitectura (ADL). Y la **documentación**, ella denota la creación de un artefacto: es decir, un documento, que puede ser, por supuesto, archivos electrónicos, páginas web o papel. Por lo tanto, documentar una arquitectura de *software* se convierte en una tarea concreta: producir un documento que describa la arquitectura de *software*.

La **arquitectura** es diseño, ella establece restricciones en las actividades posteriores, y esas actividades deben producir artefactos, diseños y código de grano más fino, que cumplen con la arquitectura. Ella es el vínculo crítico entre el diseño y la ingeniería de requisitos, ya que identifica los principales componentes estructurales de un sistema y las relaciones entre ellos. El resultado del proceso de diseño arquitectónico es un modelo arquitectónico que describe cómo se organiza el sistema según la comunicación de sus componentes Es probable que la arquitectura sea costosa **[2].**

La documentación del diseño se usa comúnmente para representar referencias comparables, establecer analogías y abstracciones, refinar y evaluar propuestas e interpretar todos estos dibujos a través de procesos cognitivos. Estas actividades refuerzan el papel de la documentación como una herramienta de descubrimiento de diseño, como un medio para refrescar la perspectiva de un proyecto. Es mucho más probable que un proyecto bien documentado esté bien construido que un proyecto pobremente documentado **[3].**

Sin embargo, a pesar de su importancia, según **[4]** la documentación de la arquitectura de *software* tiene la reputación de ser notoriamente mala. Le quita tiempo a la escritura de código. Siempre parece estar desactualizada, por lo general, está escrita en algún formato de archivo binario patentado que no se puede editar, y además de todo eso, ¡nadie lo lee de todos modos! ¡No es de extrañar que algunas personas llamen SAD (por sus siglas en inglés) a las descripciones de arquitectura de *software*! La esencia es redactar y mantener actualizados los resultados de las decisiones arquitectónicas para que los *stakeholders* de la arquitectura (las personas que necesitan saber qué trabajo hacer) tengan la información que necesitan de forma accesible y sin ambigüedades **[1].**

Esto último es la razón de por qué la notación UML sigue siendo utilizada para documentar las decisiones arquitectónicas; en lo que hay que avanzar es en mejorar la eficiencia en la elaboración y actualización de esta documentación. Las herramientas de desarrollo de *software* (CASE) son programas que se utilizan para dar soporte a las actividades del proceso de ingeniería de *software*. Por lo tanto, estas herramientas incluyen dar soporte a la actividad de diseño, editores, diccionarios de datos, compiladores, depuradores, herramientas de construcción de sistemas, etc. **[2].** Estas herramientas se pueden combinar dentro de un marco llamado Entorno de Desarrollo Interactivo (IDE), el cual proporciona un conjunto común de herramientas que soportan el proceso de desarrollo lo que las hace más fáciles de usar porque se comunican y operan de manera integrada. Según **[5]** a través de la interfaz de un IDE, un desarrollador o equipo de desarrolladores puede compilar y ejecutar código de manera incremental y administrar los cambios en el código fuente de manera uniforme. Éstos, generalmente están diseñados para integrarse con bibliotecas de control de versiones de terceros, como GitHub y Subversion de Apache. Algunos entornos pueden admitir el desarrollo basado en modelos (MDD) por lo que en un IDE también se registran modelos y los cambios realizados en ellos.

B. Programación Orientada a Protocolos y un Perfil de UML

Los protocolos se utilizan para definir un "modelo de métodos, propiedades y otros requisitos que se adaptan a una tarea o función en particular" **[6].** Swift verifica los problemas de conformidad del protocolo en tiempo de compilación, lo que permite a los desarrolladores descubrir algunos errores fatales en el código, incluso antes de ejecutar el programa. Éstos permiten a los desarrolladores escribir código flexible y extensible en Swift sin tener que comprometer la expresividad del lenguaje. Las extensiones de protocolo se basan en su genialidad **[6].**

Según **[7]** los protocolos permiten agrupar métodos, funciones y propiedades similares. Swift le permite especificar estas garantías de interfaz en tipos de *class*, *struct* y *enum*. Solo los tipos de *class* pueden usar clases base y herencia. Una ventaja de los protocolos en Swift es que los objetos pueden ajustarse a múltiples protocolos. Al escribir una aplicación de esta manera, el código se vuelve más modular Esto es una característica de calidad que promueve POP.

Piénsese en los protocolos como bloques de construcción de funcionalidad. Cuando se agrega una nueva funcionalidad al conformar un objeto a un protocolo, no se crea un objeto completamente nuevo, eso lleva mucho tiempo. En su lugar, se agregan diferentes bloques de construcción hasta que su objeto esté listo **[7].** La idea de la Programación Orientada a Protocolos (POP) es dejar atrás, en la medida de lo posible, los tipos por referencia (Objetos) y centrarse en los tipos por valor (*Structs*, *Enums*), eliminando varios de los problemas que se generan al utilizar Programación Orientada a Objetos (POO) **[8].** Ahora bien, dado que POP se ha transformado en un estándar para desarrollos en Swift, surge la necesidad de documentar los diseños que se implementarán bajo este paradigma. Unified Modeling Language (UML), definido por [9], es un lenguaje de propósito general usado para especificar, visualizar y construir artefactos de sistemas de *software*. Como es un lenguaje de propósito general, la descripción de dominios de aplicación específicos requiere de la definición de un nuevo lenguaje que describe el

metamodelo, utilizando la extensión propiamente de UML a través de un mecanismo denominado Perfiles UML. Los perfiles UML proveen un mecanismo de extensión genérico para construir modelos UML en dominios particulares. Están basados en estereotipos, restricciones y valores etiquetados adicionales que son aplicados a los elementos o relaciones de un diagrama **[10].** Es oportuno aclarar que los perfiles no son una capacidad de extensión de primera clase (es decir, no permiten crear nuevos metamodelos), más bien, la intención de los perfiles es brindar un mecanismo directo para adaptar un metamodelo existente con construcciones que son específicas para un dominio, plataforma o método en particular. Cada una de estas adaptaciones se agrupa en un perfil. Aunque no es posible eliminar ninguna de las restricciones que se aplican a UML cuando se le utiliza, pero si es posible agregar nuevas restricciones que sean específicas del perfil.

iii. EL PROBLEMA

Según **[11]** los programadores hacen uso de la documentación en sus actividades diarias, pero a su vez desatienden la creación de la misma, por lo que cada vez se realizan más esfuerzos para la creación de sistemas de autogeneración de documentación.

Como apoyo a esta actividad, se desarrollan herramientas de *software* tales como DocC, el compilador de documentación *open-source* creado por Apple y la herramienta oficial para la documentación de proyectos de *software* basados en lenguaje de programación Swift **[12].**

Según **[13]** entre los lenguajes de programación más populares de 2022 se encuentran Python, Java, Rust, C++ y Swift, todos estos lenguajes tienen su propia herramienta de generación de documentación, PyDoc, JavaDoc, RustDoc, Doxygen, DocC en el orden correspondiente. DocC es la herramienta más reciente entre las mencionadas anteriormente, fue anunciada en 2021 y publicada como *open-source* a finales de ese mismo año, aún se encuentra en un estado de desarrollo activo y no iguala el nivel de funcionalidades que ofrecen las herramientas pertenecientes a otros lenguajes de importancia similar. En **[14]**, se indica que una de las funcionalidades que tienen estas herramientas de documentación, excepto DocC, es la generación de diagramas de manera automática mediante el análisis estructural del código compilado. En otras palabras, para que la documentación de los sistemas de *software* se considere completa, esta debiera contener tanto la especificación de las interfaces ofrecidas como una vista general de la arquitectura del sistema **[15];** de esta manera se resalta la importancia de la generación de diagramas de clase en las herramientas de generación de documentación.

No se encontraron estudios donde se hagan análisis comparativos entre DocC y otros generadores de documentación, posiblemente debido a lo reciente de la herramienta, pero se puede afirmar a partir de lo expuesto que el entorno de generación de documentación Swift, no promueve la completitud porque carece de un generador de diagramas.; esto afecta la experiencia del desarrollador y consecuentemente su reputación. En este sentido, se promueve la mejora a DocC incorporando la capacidad de documentar la vista lógica de la arquitectura de aquellas aplicaciones desarrolladas con Swift y por ende con el uso de POP, pero para ello, se requiere primero definir un perfil para este paradigma de programación.

iv. PERFIL PARA POP

Para la formulación del perfil de UML para POP se hizo un análisis de la guía oficial del lenguaje Swift **[16]** el cual consistió en su comparación con la guía de especificación UML 2.5 por OMG **[17].**

Como resultado de este análisis se identificaron los **tipos existentes** integrados en el lenguaje Swift, en la sección de referencia del lenguaje, apartado titulado "*Types*". Esta guía hace mención a los siguientes tipos: Los tipos existentes se categorizan en tipos con nombre y tipos compuestos. Los tipos con nombre son aquellos a los que se les puede dar un nombre particular al ser definidos, estos incluyen clases, estructuras, enumeraciones, protocolos, *arrays*, diccionarios, *sets*, y todos aquellos tipos que se consideran como primitivos en otros lenguajes (números, caracteres, texto). Los tipos compuestos son tipos sin nombre, estos son las funciones y las tuplas. **[16].**

La Tabla 1 muestra una descripción de cada una de las estructuras de datos mencionadas anteriormente. Estas descripciones permitieron trazar una base conceptual para luego hacer la comparación entre las estructuras de datos provistas por Swift y las definidas por UML, identificando así cuáles se debían estereotipar

Tabla I: Tipos existentes en el lenguaje Swift 5.7

Nombre	Descripción
Clases	"Construcción que nos permite encapsular las propiedades, métodos e inicializadores de un objeto en un solo tipo." [a] Los objetos se denominan instancias Pueden relacionarse con otras clases mediante herencia Pueden relacionarse con protocolos Las instancias se comparten mediante referencia, apuntando a la misma dirección en memoria Es un tipo con nombre Se puede denominar subclase o superclase dependiendo de su relación con otra clase mediante herencia
Estructuras	"Construcción que nos permite encapsular las propiedades, métodos e inicializadores de una instancia en un solo tipo." [a] No pueden heredar comportamiento ni atributos Pueden relacionarse con protocolos Las instancias se comparten mediante valor, creando copias Es un tipo con nombre
Enums	"Tipo que define una lista de posibles valores. Adopta muchas características tradicionalmente admitidas solo por clases, como propiedades, métodos, inicializadores. Se puede extender para expandir su funcionalidad más allá de su implementación original y puede ajustarse a los protocolos para proporcionar una funcionalidad estándar." [a]
Protocolos	"Tipo que define un modelo de métodos, propiedades y otros requisitos que se adaptan a una tarea o funcionalidad en particular. Luego, el protocolo puede ser adoptado por una clase, estructura o enumeración para proporcionar una implementación real de estos requisitos. Cualquier tipo que satisfaga los requisitos de un protocolo se dice que conforma a ese protocolo." [a]
Tipos estándar (*Int*, *Double*, *String*, etc.)	"Los tipos primitivos de cualquier lenguaje de programación, están implementados por Swift como estructuras por lo que se pueden extender." [a]
Colecciones (*array*, *set*, *dictionary*)	"Tipos que actúan como colecciones para guardar valores. Los *array* son colecciones ordenadas de valores, los set son colecciones desordenadas de valores únicos y los diccionarios son colecciones desordenadas de asociaciones entre un valor y una clave." [a]
Funciones	"Tipo que representa los tipos de los métodos, funciones o *closures* y están conformados por parámetros y un tipo de retorno" [a]
Tuplas	"Las tuplas agrupan múltiples valores en un solo valor compuesto. Los valores dentro de una tupla pueden ser de cualquier tipo y no tienen que ser del mismo tipo entre sí" [a]
Opcionales	"Tipo que manejan la ausencia de un valor (...). Un opcional representa dos posibilidades: O hay un valor y puede desenvolver el opcional para acceder a ese valor, o no hay ningún valor" [a]

Nota. [a] Apple (2023)

En la Tabla II se muestran las posibles relaciones que pueden tener los tipos de la Tabla I.

Tabla II: Relaciones existentes en el lenguaje Swift 5.7

Nombre	Descripción
Herencia	"Relación mediante la cual se puede agregar o anular funcionalidades mediante composición de clases. La herencia es un comportamiento fundamental que diferencia las clases de otros tipos en Swift" [a]
Conformance	"Relación mediante la cual una clase, estructura o enumeración adopta y provee una implementación a los requerimientos de un protocolo" [a]
Herencia de protocolos	"Relación mediante la cual un protocolo puede heredar uno o más protocolos y puede agregar más requisitos además de los requisitos que hereda" [a]
Tipos anidados	"Relación mediante la cual se define un tipo dentro del contexto de otro tipo. para permitir esto Swift permite definir tipos anidados, el cual anida enumeraciones, clases y estructuras compatibles dentro de la definición del otro tipo" [a]
Extensión	"Relación mediante la cual se le agrega nuevas funcionalidades a clases, estructuras, enumeraciones o protocolos ya existentes (...). Las extensiones a protocolos sirven para proveer implementaciones por defectos a sus métodos y propiedades." [a]

Nota. [a] Apple (2023)

Se formuló la suposición que las entidades y relaciones a estereotipar son a las que hizo referencia [18], estas son: Protocolos, Estructuras, Enumeraciones, Extensiones y relaciones de *Conformance*. Desde entonces, el lenguaje Swift ha avanzado a tres versiones mayores [19] y la especificación de UML ha avanzado una versión [20], se hizo necesario una revisión de ambos para validar que la

suposición es correcta. Para lograr esto, como la guía de Swift no hace referencia explícita a cuáles de los tipos mencionados son necesarios para poder hacer uso de POP, se procedió a identificar para las tablas I y II cuál es su estructura equivalente definida por UML; lo cual se analizó para cada tipo, esta selección se hizo en base a la comparación de sus nombres y definiciones. Luego, si se encontraba alguna diferencia entre ellas a nivel conceptual se determinaba que era necesario realizar un estereotipo. Este análisis se resume en la tabla III.

Tabla III: Comparación entre las estructuras definidas por el lenguaje Swift 5.7 y las definidas por UML 2.5

Swift	UML Ref.	A estereotipar	Dif. / Obs.
Clases	11.4. Clases		
Estructuras	11.4. Clases	X	Aunque las clases, al igual que las estructuras, sirven para especificar las características y comportamientos de los objetos, estas pueden tener ancestros mediante relaciones de herencias, no soportadas por las estructuras en Swift.
Enums	10.2.3.3. Enumeraciones	X	A pesar de que UML define el modelado de las enumeraciones en su apartado 10.2.3.3 estos no son equivalentes, ya que en UML las enumeraciones son consideradas tipos de datos que pueden almacenar únicamente valores enumerados, mientras que las enumeraciones en Swift pueden además de esto almacenar propiedades y métodos.
Protocolos	10.4. Interfaces	X	Existen dos diferencias entre los protocolos y las interfaces por los cuales no resultan estructuras equivalentes, estas son: A diferencia de las enumeraciones en UML, los atributos declarados en los protocolos si deben ser implementados por la estructura que la realiza. Los protocolos pueden ser extendidos para proveer implementaciones a sus métodos y atributos, lo cual no es posible con las interfaces
Tipos estándar	21. Tipos Primitivos	-	-
Tipos de colecciones	7.5. Tipos y Multiplicidad	-	-
Funciones	9.6. Operaciones	-	-
Tuplas	11.5. *Associations*	-	-
Opcionales	8.6.10. *LiteralNull*	-	-

En la Tabla IV se muestra un resumen que incluye las relaciones de la Tabla II.

Tabla IV: Comparación entre las relaciones definidas por el lenguaje Swift 5.7 y las definidas por UML 2.5

Swift	UML Ref.	A estereotipar	Dif./Obs.
Herencia	9.9.7. Generalización	-	-
Conformance	10.5.6. *InterfaceRealization*	-	-
Herencia de protocolos	9.9.7. Generalización	-	-

Swift	UML Ref.	A estereotipar	Dif./Obs.
Tipos anidados	-	X	No se encontró en la guía UML una descripción de cómo representar la relación de tipos anidados por lo que se deberá estereotipar.
Extensión	22.3. Estereotipos estándar - Refinamiento	X	Aunque el refinamiento coincide con la descripción de extensión entre clases, estructuras y enumeraciones, es necesario refinarlo ya que en Swift se permite extender protocolos para proveer implementaciones a todos los tipos que lo conforman.

Con base en los análisis de las tablas III y IV se concluyó que, para poder representar un sistema basado en POP, coincidiendo parcialmente con la suposición inicial, es necesario estereotipar los siguientes tipos y relaciones:

- Estructuras
- Enumeraciones
- Protocolos
- Tipos anidados
- Extensiones

Aunque la relación de tipos anidados no es única a POP, ya que se usa en lenguajes de programación con un enfoque POO como Java, es una relación que también se usa por POP y no está documentada por la versión más reciente de UML.

La formulación del perfil incluye los estereotipos con sus nombres de etiqueta, la metaclase o relación del cual extienden y su descripción. Ver la Tabla V, en ella se usó la misma representación que el OMG usa para los estereotipos estándar de UML 2.5 en su apartado 22.3.

Tabla V: Diagrama de Perfil que incluye tipos y relaciones estereotipadas para la diagramación de sistemas implementados en Swift 5.7

Nombre	Aplica a	Descripción
<<nested>>	Composición	Una dependencia de composición que denota una relación de propiedad. Esta relación se usa para representar tipos anidados donde un tipo se declara en el ámbito de otro tipo. El tipo principal declara el tipo interno.
<<extension>>	Abstracción	Especifica una relación de refinamiento de una clase, estructura o enumeración. Una relación de extensión entre protocolos provee implementaciones a los atributos y métodos de ese protocolo.
<<Enum>>	Clases	Una clase que soporta, además de propiedades y métodos, enumeraciones literales.
<<Extension>>	Clases	Una clase del mismo tipo de la clase que extiende. Especifica refinamiento mediante agregación de elementos y atributos. Una extensión relacionada a un protocolo implica proporcionar implementaciones de métodos, y propiedades para los tipos conformes al protocolo.
<<Protocol>>	Interfaces	Una interfaz que permite definir propiedades y métodos como requerimientos, además de ser extendido para proveer implementaciones por defecto y conformado mediante la relación *conformance*.
<<Structure>>	Clases	Una clase que no hereda de otras clases y las instancias se copian por valor, a diferencia de la clase tradicional que copia por referencia.

Una vez formulado el perfil para sistemas basados en POP se propone su representación gráfica, ver **Figura 1.**

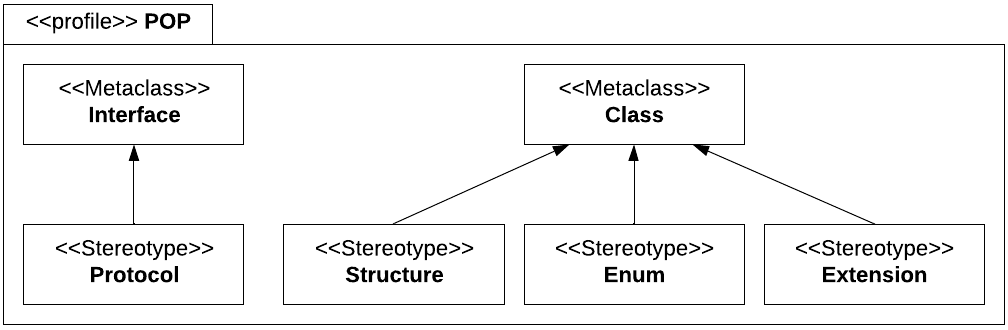

Figura 1: Diagrama de perfil UML para sistemas codificados bajo el paradigma POP
Nota. Diagrama de perfil realizado según lo señalado en *UML 2 Toolkit* por OMG (p.39,2004)

v. CASO DE USO DEL PERFIL POP

Para ejemplificar la aplicación del perfil POP se realizó un diagrama de clases con compartimentos suprimidos modelando algunas de las colecciones provistas por la librería estándar de Swift, *Array*, *Set* y *Dictionary*, la cuáles hacen uso extendido de la composición por protocolos para refinar su comportamiento **[21]**.

El diagrama realizado aplica algunos de los estereotipos propuestos en el diagrama de perfil POP, ver **Figura 2**.

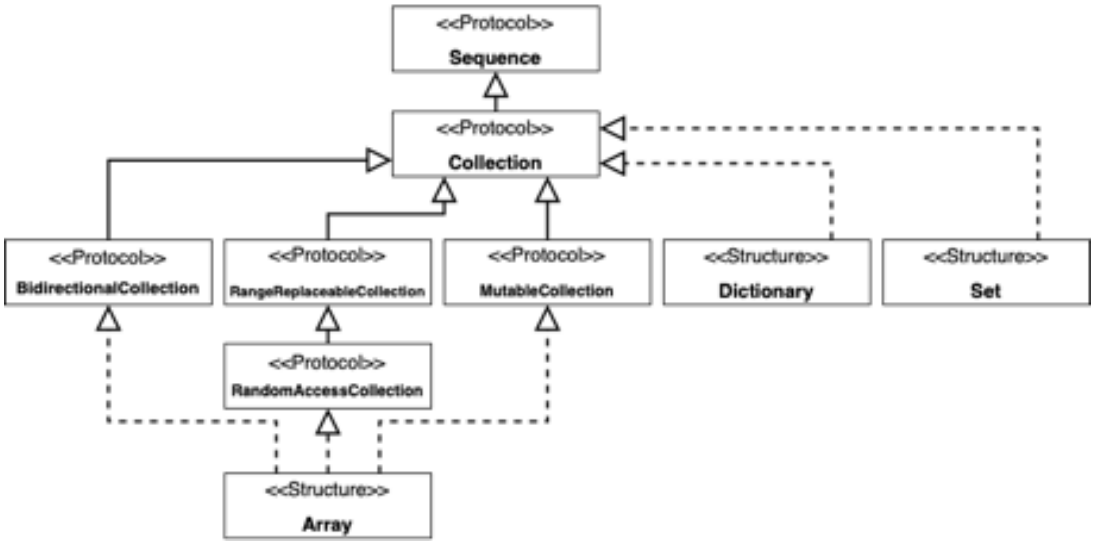

Figura 2: Modelado de protocolos para las colecciones *Array*, *Set* y *Dictionary* mediante un diagrama de clases UML aplicando el estereotipo POP

vi. CONCLUSIONES

La Programación Orientada a Protocolos se está convirtiendo en un paradigma de programación maduro, y por esto se argumenta la necesidad de formular un modelo formal para el diseño y documentación de sistemas que hagan uso de este.

La documentación de la vista lógica de la arquitectura se usa comúnmente para representar referencias comparables, establecer analogías y abstracciones, refinar y evaluar propuestas e interpretar todos estos modelos a través de procesos cognitivos, de allí su importancia.

El perfil UML para POP propuesto constituye un producto importante que permite cerrar la brecha existente entre las herramientas de diseño destinadas al modelado de objetos y el modelado de protocolos. Además de contribuir al avance del lenguaje Swift, permitiendo desarrollos basados en MDD, como el de la autogeneración de diagramas, en su herramienta de generación de documentación DocC.

Para su formulación se realizó una revisión documental de la definición formal de UML y la documentación oficial de Swift, comparando y equivaliendo definiciones, para así concebir estereotipos y valores etiquetados compatibles con el metamodelo UML 2.5.2, formulados en un diagrama de perfil UML.

Finalmente, se ejemplifica el uso del perfil propuesto, y se aplica al modelado de los protocolos y estructuras de las colecciones provistas por la librería estándar de Swift.

vii. ACCIONES FUTURAS

Aún queda camino por recorrer para lograr formalizar un perfil que exprese todas las bondades del POP. Para que este perfil madure es necesario que la comunidad de programadores de Swift revise su definición, verificando y validando lo aquí propuesto. Específicamente, se recomienda ahondar en las siguientes áreas:

- Representación de restricciones de protocolos mediante tipos asociados en los diagramas de clase UML.
- Aplicación del perfil propuesto a diagramas de comportamiento UML.
- Aplicación del perfil propuesto a diagramas estructurales UML.
- Generación de diagramas como parte del proceso de documentación de sistemas haciendo uso del perfil POP.

REFERENCIAS

[1] P. Clements, L. Bass, y D. Garlan, *Documenting software architectures: Views and beyond*. Boston, MA: Addison–Wesley, 2003.

[2] I. Sommerville, *Software Engineering*. Boston, MA: Addison-Wesley, 2020.

[3] Design Workshop, *Landscape architecture documentation standards: Principles, guidelines, and best practices*. Chichester, England: John Wiley & Sons, 2015.

[4] H. Gomaa, *Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures*. Cambridge, England: Cambridge University Press, 2011.

[5] A. S. Gillis y V. Silverthorne, "integrated development environment (IDE)", *Software Quality*, 2018. [En línea]. Disponible en: https://www.techtarget.com/searchsoftwarequality/definition/integrated-development-environment. [Consultado: 23-abr-2023].

[6] A. Gaidukov, "An introduction to protocol-oriented programming in swift", *Toptal Engineering Blog*, 2016. [En línea]. Disponible en: https://www.toptal.com/swift/introduction-protocol-oriented-programming-swift. [Consultado: 23-abr-2023].

[7] Izertis, "Introducción a la programación orientada a protocolos con Swift", *Solid GEAR*, 2017. [En línea]. Disponible en: https://ahorasomos.izertis.com/solidgear/introduccion-programacion-orientada-protocolos-swift/. [Consultado: 23-abr-2023].

[8] P. Carrascal, "¿Qué es la programación orientada a protocolos?", *Quora*, 2018. [En línea]. Disponible en: https://es.quora.com/Qu%C3%A9-es-la-programaci%C3%B3n-orientada-a-protocolos. [Consultado: 23-abr-2023].

[9] G. Booch, J. Rumbaugh, y I. Jacobson, *The unified modeling language user guide*, 2a ed. Boston, MA: Addison-Wesley, 2005.

[10] R. Rodríguez y M. Goncalves, "Perfil UML para el modelado visual de requisitos difusos", *Enl@ce*, vol. 6, núm. 3, pp. 29–46, 2009.

[11] A. LeClair, S. Haque, L. Wu, y C. McMillan, "Improved code summarization via a graph neural network", *ICPC '20: Proceedings of the 28th International Conference on Program Comprehension*, pp. 184–195, 2020.

[12] Apple, "Swift-DocC is Now Open Source", *Swift.org*, 13-oct-2021. [En línea]. Disponible en: https://www.swift.org/blog/swift-docc/. [Consultado: 23-abr-2023].

[13] "TIOBE index", *TIOBE*, 12-dic-2022. [En línea]. Disponible en: https://www.tiobe.com/tiobe-index/. [Consultado: 12-dic-2022].

[14] A. Terrasa, J. Private, y G. Tremblay, "Documentation generators support for program comprehension: Where are we?", feb. 2015.

[15] A. Soukka, T. Hastrup, T. J. Lukka, y B. Fallenstein, "Bridging Javadoc and design documentation via UML diagram image maps". 2021.

[16] Apple, "The Swift Programming Language. Swift 5.7", *Swift.org*, 2023. [En línea]. Disponible en: https://docs.swift.org/swift-book/documentation/the-swift-programming-language.

[17] OMG, "OMG Unified Modeling Language (OMG UML) Version 2.5.1", 2017.

[18] D. Abrahams, "Protocol-Oriented Programming in Swift", 2015.

[19] Apple, "Document Revision History", *Swift.org*, 2023. [En línea]. Disponible en: https://docs.swift.org/swift-book/documentation/the-swift-programming-language/revisionhistory.

[20] OMG, "About the unified modeling language specification version 2.5.1", *Omg.org*, 2023. [En línea]. Disponible en: https://www.omg.org/spec/UML/2.5.1/About-UML.

[21] Apple, "Sequence and collection protocols", *Apple Developer Documentation*, 2023. [En línea]. Disponible en: https://developer.apple.com/documentation/swift/sequence-and-collection-protocols. [Consultado: 24-abr-2023].