Análisis del Uso de Términos Frecuentes en el Nombrado de Componentes Arquitectónicos

Analysis of the Use of Frequent Terms in Architectural Component Naming

Paul Mendoza del Carpio *

RESUMEN

La identificación de componentes arquitectónicos es de relevancia para actividades de mantenimiento de software, sin embargo ésta es dificultada en muchas ocasiones por la ausencia de documentación arquitectónica y el volumen de código fuente. Por otro lado, es usual encontrar términos empleados en forma frecuente al nombrar componentes definidos por un framework o biblioteca. Este trabajo presenta un análisis del uso frecuente de términos en el nombrado de tipos de objetos, ello enfocado a la identificación de componentes arquitectónicos de una aplicación. Para tal análisis se tomaron proyectos de aplicación obtenidos de repositorios de GitHub y se evaluaron algunos escenarios de nombrado para la identificación de términos frecuentes. Los resultados lograron obtener algunos términos representativos que corresponden a nombres de tipos de componentes.

Palabras clave: componente arquitectónico, nombrado de componentes.

ABSTRACT

The identification of architectural elements is relevant for activities of software maintenance, however that is harder due to the lack of architectural documentation and the volume of source code. On the order hand, it is usual to find terms used frequently in naming components defined by a framework or a library. This work presents an analysis about the frequent use of terms in naming object types, focusing the identification of architectural components in an application. For this analysis, application projects were gotten from GitHub repositories and some naming scenarios were evaluated for the identification of frequent terms. The results got some representative terms that match to names of component types.

Key words: architectural component, component naming.

*Magister en Ingeniería de Software y Candidato a Doctor en Ciencias de la Computación. Profesional IBM Certificado. Docente de la UAP.
E-mail: pmendozadelcarpio@gmail.com
Twitter: @paulnetpro

INTRODUCCIÓN

El entendimiento de una aplicación de software consume una considerable cantidad de tiempo en actividades de mantenimiento de software. Frente a ello, el tener conocimiento delos principales elementos de la aplicación, como los componentes arquitectónicos de la misma, es de gran utilidad para la comprensión del software a mantener. Sin embargo, la identificación de componentes arquitectónicos es dificultada en muchas ocasiones por la ausencia de documentación arquitectónica y el volumen de código fuente respecto a cantidad de archivos y número de líneas de código.

Muchas aplicaciones son desarrolladas empleando frameworks o bibliotecas que definen componentes con determinados nombres que son empleados también en los nombres de las aplicaciones. Se pueden encontrar frameworks estándar que son definidos por la plataforma (e.g. Java EE, .Net, Java ME), los cuales a su vez definen términos comúnmente incluidos en los nombres de sus principales componentes (e.g., Bean, Controller, View, Frame). También se tienen estándares para el nombrado de clases, interfaces y paquetes en diversos lenguajes de programación. Luego, se pueden distinguir términos empleados en forma repetitiva al seguir convenciones de codificación establecidas en forma estándar por la industria o por el equipo de desarrollo de software en particular. Este trabajo presenta un análisis del uso frecuente de términos en el nombrado de tipos de objetos (ice, clases, interfaces), enfocado a mostrar su posible utilidad para la identificación de componentes arquitectónicos de una aplicación.

Hipótesis

La obtención de términos frecuentemente empleados en nombres de tipos de objetos, puede apoyar la identificación de términos que representan tipos de componentes.

MATERIALES Y MÉTODOS

Material

A. *Aplicación Java*: aplicación implementada para el recorrido y análisis de archivos de código fuente, registro de términos empleados en los nombres de clases, y obtención de términos frecuentes.

B. *Neo4j*: base de datos basada en grafos donde se almacenan nodos y relaciones entre nodos, con información acerca de los tipos de objetos, paquetes, términos empleados en nombres, y relaciones entre ellos.

C. *Cypher*: lenguaje de consulta de Neo4j, el cual permite la consulta de nodos y relaciones de acuerdo a etiquetas y propiedades de los nodos. También soporta la ejecución de consultas agregadas, consultas de importancia en el análisis, para el cálculo de la frecuencia de uso de términos.

D. *Apache Commons Lang*: conjunto de bibliotecas con utilitarios para la manipulación de cadenas, números, y serialización entre otros. Principalmente es empleada para obtener los términos contenidos en los nombres de tipos de objetos, ello considerando el estilo de nombrado Camel Case.

E. *GitHub*: servicio de hosting Web basado en Git, cuenta con una vasta cantidad de proyectos de código abierto. Utilizado como fuente de repositorios donde se alojan aplicaciones de diversos dominios.

F. *Hojas de cálculo*: hojas de cálculo Excel con el registro de datos cuantitativos correspondientes a los proyectos de aplicación evaluados. También empleadas para la generación de gráficas comparativas.

METODOLOGÍA

A. Definir escenarios comunes de nombrado de componentes

Como una buena práctica, los programadores emplean términos representativos del contenido del código fuente en los nombres de tipos de objetos, los cuales son utilizados como prefijos o sufijos, sobre todo en aquellos tipos que tienen especial distinción en la estructura de la aplicación, por tratarse de componentes de un framework empleado (e.g. Controller, Entity, Bean) o de una infraestructura propia (Servicio, Ventana). Asimismo, siendo la arquitectura un diseño de alto nivel, los términos empleados frecuentemente serán considerados candidatos a componentes arquitectónicos por ser comunes y representativos de varios otros elementos que contienen el término en su nombre.

A fin de identificar términos frecuentes, se han tomado en cuenta tres escenarios comunes, donde se presentan repeticiones de términos en los nombres de tipos de objetos, éstos son descritos a continuación.

El primer escenario consiste en términos que son empleados repetidamente en tipos de objetos incluidos en paquetes con un mismo nombre y en un mismo nivel de la jerarquía de paquetes, la posición del término es considerada para distinguir si se trata del primero, segundo, penúltimo, último u otro intermedio. La figura 1 muestra un ejemplo para este escenario.

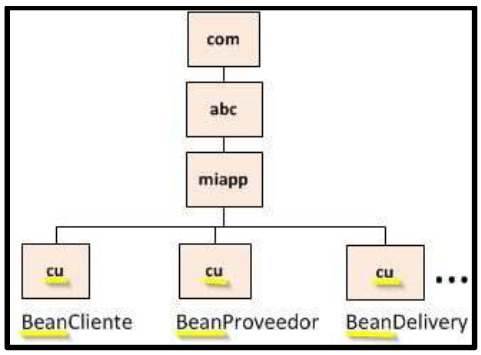

Figura1. Primer escenario de uso frecuente de términos.

El segundo escenario consiste en términos que son empleados repetidamente en tipos de objetos incluidos en paquetes de distinto nombre pero en un mismo nivel de la jerarquía de paquetes, la posición del término también es considerada. La figura 2 muestra un ejemplo para este escenario.

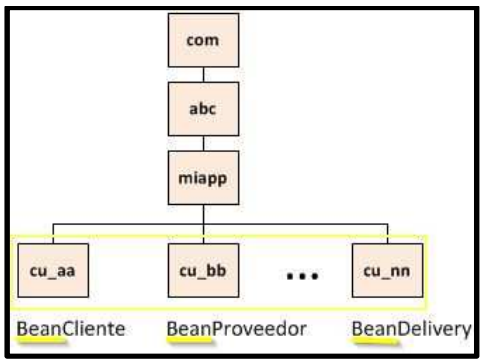

Figura2. **Segundo escenario de uso frecuente de términos.**

El tercer escenario consiste en términos que son empleados repetidamente en tipos de objetos incluidos en un mismo paquete. La figura 3 muestra un ejemplo para este escenario.

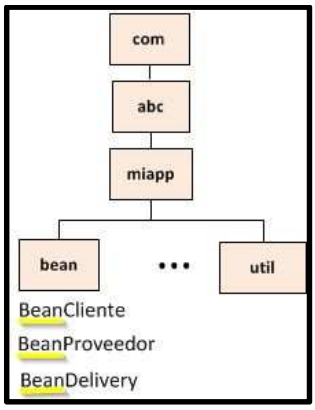

Figura3. **Tercer escenario de uso frecuente de términos.**

B. Recolección de código fuente desde una fuente que proporcione diversidad de aplicaciones

Se han descargado aplicaciones desde la plataforma GitHub, donde se tienen una gran diversidad de proyectos de aplicación con su código fuente y recursos, los cuales son utilizados como fuentes de datos en diversas investigaciones. Navegando entre los repositorios de GitHub se han seleccionado proyectos desarrollados en lenguaje Java, considerando tanto aplicaciones empresariales como software base (i.e., bibliotecas, servidores, frameworks) que contengan al

menos 30 archivos de código fuente. El número de archivos y la cantidad de líneas de código para cada proyecto son mostrados en la tabla 1.

Tabla 1. Proyectos seleccionados de GitHub. NA indica el número de archivos. LOC (Lines of Code) indica el número de líneas de código.

ID	Proyecto	NA	LOC	ID	Proyecto	NA	LOC
1	busbuddy	31	2677	17	spring-mongodb	232	32645
2	camel	1251	192725	18	spring-data-rest	72	6024
3	canchita	32	3268	19	spring-hadoop	47	8056
4	geoportal	1252	200634	20	spring-integration	498	49286
5	ipfixconfig	89	27563	21	spring-petclinic	44	2916
6	jmeter	372	61968	22	sqlite-jdbc	34	10467
7	mockito	327	23252	23	stetho	91	7156
8	mr4c	278	25652	24	suri-spring	55	5010
9	opendata-schema	54	4194	25	szysrc	55	3176
10	openentity-api	32	3130	26	tapestry	1034	96096
11	pdfbox	511	110796	27	ticket2rock	94	6838
12	plato	86	15415	28	tomcat	1504	397318
13	poi	993	175049	29	topicindex	188	69946
14	quizzo-web	45	1940	30	wayback-core	517	61431
15	spring-data-commons	327	35629	31	webservice	38	3038
16	spring-data-jpa	94	12003	32	wicket	808	159227

Cabe señalar que entre los elementos de la tabla 1 se tienen proyectos desarrollados por las organizaciones Spring y Apache, dos populares organizaciones en el campo de software. Los proyectos de Spring se pueden distinguir por utilizar el prefijo *spring*, mientras que entre los proyectos de Apache se tienen: camel, jmeter, pdfbox, poi, tapestry, tomcat, wicket.

C. Implementación de una aplicación para el análisis de código fuente de los proyectos obtenidos

Se implementó una aplicación Java para el análisis de código de cada proyecto de aplicación. El autor de este trabajo desarrolló un framework, el cual fue utilizado para la implementación, el cual registra la información recopilada en una base de datos basada en grafos. Esta base de datos cuenta con un visualizador de grafos que permite mostrar los grafos almacenados, así como ejecutar consultas sobre el grafo a fin de tener una vista más precisa de los elementos, véase la figura 4.El framework durante su recorrido habitual realizado sobre los archivos de un proyecto a analizar, crea nodos para cada término usado en el nombre de un tipo de objeto, registrando su nivel en la jerarquía de paquetes y su posición de acuerdo a un índice numérico (ice, inicio, final, otro cercano al inicio o final). Luego, contando con un registro de cada término utilizado en un tipo de objeto, es posible ejecutar una consulta agregada sobre el grafo para identificar los escenarios de términos frecuentemente empleados (véanse las figuras 1, 2, 3).Cabe mencionar que el framework permite establecer los porcentajes de ocurrencia mínimamente dados para considerar a un término como frecuente.

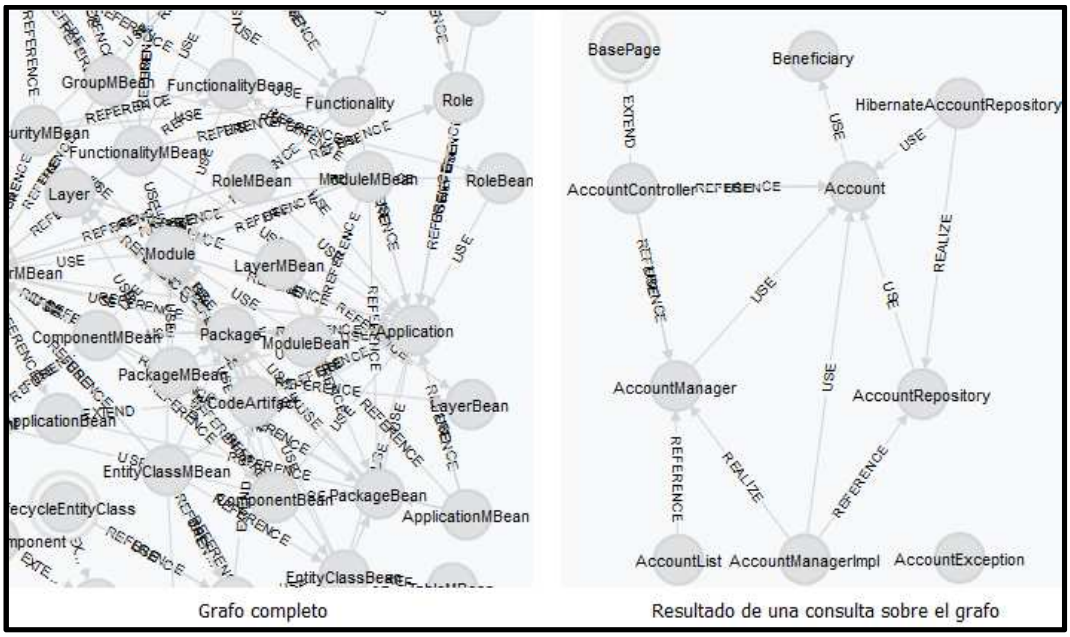

Figura 4. **Grafos con nodos de ejemplo.**

RESULTADOS

Para el análisis en cuestión se evaluaron los proyectos de aplicación de la tabla 1 mediante la aplicación mencionada en la sección C de la metodología. Se descubrieron términos frecuentemente empleados según los escenarios mostrados en las figuras 1, 2, y 3, y con un porcentaje de ocurrencia mínimo de 90%.Luego, los resultados obtenidos son mostrados en la tabla 2.

ANÁLISIS Y DISCUSIÓN

En general, los resultados obtenidos muestran que más del 81% de proyectos presentaban al menos tres términos frecuentes, y en varios proyectos se hallaron términos familiares que están definidos como componentes en diversas plataformas (e.g., Servlet, Bean), frameworks (e.g., Event, Action, Service), patrones de diseño (e.g., Factory, Strategy, Adapter, Controller, DAO) o documentación del proyecto de software (e.g., Parser, Handler, Task, Query). Adicionalmente, se debe mencionar que la tabla está mostrando sólo los principales términos que son de interés para identificar componentes de la aplicación, pero aún se tienen otros términos más según el número de términos indicado, acerca de ellos, se ha identificado que para el caso de aplicaciones empresariales (i.e., no software base, bibliotecas, o frameworks) es habitual encontrar términos relacionados al negocio que atiende la aplicación.

Tomando en consideración que en aplicaciones orientadas a objetos, habitualmente cada tipo de objeto es declarado en un archivo propio, se ha prestado atención al

número de archivos pues ello estaría directamente relacionado a la cantidad de nombres y términos que se podrían obtener para cada proyecto. Luego, se han agrupado los términos frecuentes por intervalos de cantidad de archivos, comprobándose que el número de términos se incrementa habitualmente conforme se incrementa el número de archivos presentes como se muestra en la figura 5.

Tabla 2.Términos descubiertos en el análisis de los proyectos. NT indica el número de términos frecuentes encontrados.

ID	Proyecto	NT	Principales términos
1	busbuddy	8	Controller, DAO, Jdbc
2	camel	43	Binding, Control, Converter, Rest
3	canchita	5	Converter, DAO, Service
4	geoportal	49	Adapter, Protocol, Query, Servlet
5	ipfixconfig	1	Nc
6	jmeter	14	Bean, Converter, Processor
7	mockito	14	Invocation, Mock, Runner, Unit
8	mr4c	9	Bean, Config, Resource, Runner
9	opendata-schema	6	Extractor, Schema, Service
10	openentity-api	3	I, Result, Search
11	pdfbox	18	Pattern, Sequence, Stream, Text
12	plato	8	Action, Converter, Service, View
13	poi	25	Cell, Crypto, Extractor, Reader
14	quizzo-web	4	Chat, Controller, Game, Response
15	spring-data-commons	11	Data, Repository, Spring, Strategy
16	spring-data-jpa	5	Jpa, Persistence, Repository, Unit
17	spring-mongodb	0	
18	spring-data-rest	1	Event
19	spring-hadoop	3	Hadoop, Script, Utils
20	spring-integration	23	Store, Strategy, Value
21	spring-petclinic	12	Impl, Jdbc, Repository, Service
22	sqlite-jdbc	7	Connection, Data, JDBC, Lite, SQ
23	stetho	2	Response, Result
24	suri-spring	12	Converter, Handler, Renderer
25	szysrc	9	Filter, Listener. Mapper, Servlet
26	tapestry	17	Locator, Renderer, Transformer
27	ticket2rock	8	Admin, Bean, Locator, Service
28	tomcat	27	Factory, Filter, Servlet, Task
29	topicindex	0	
30	wayback-core	35	Filter, Handler, Parser, Resource
31	webservice	0	
32	wicket	47	Action, Adapter, Filter, Renderer

Cabe mencionar una particularidad de los proyectos de Spring, para algunos de ellos no se identificaron términos frecuentes. Para esos casos se revisó el código fuente en forma manual, encontrándose que algunos paquetes contenían archivos con diferentes prefijos que se repetían, ello sugiere que tales archivos podrían ser mejor organizados creando nuevos paquetes y distribuyendo los archivos.

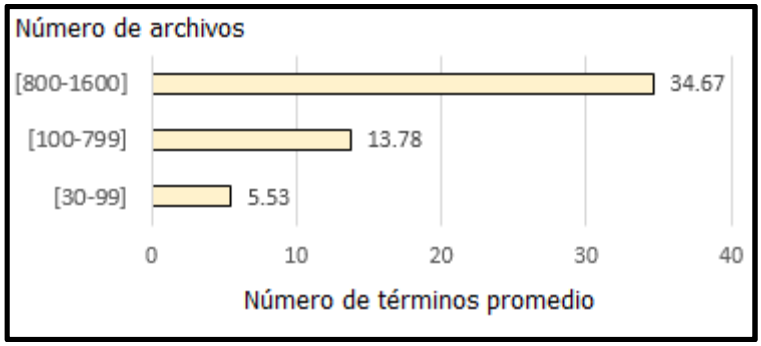

Figura 5. **Intervalos de número de archivos y sus términos frecuentes en promedio.**

Respecto a los proyectos de Apache, se puede ver una mejor uniformidad en cuanto a sus proyectos, lo cual se ve reflejado en lo significativos que son los principales términos encontrados para estos proyectos. Los proyectos camel y wicket, dos frameworks de Apache, figuran entre los proyectos que cuentan con la mayor cantidad de términos frecuentes al lado del proyecto geoportal de la compañía Esri de California.

CONCLUSIONES

1. Los términos obtenidos como frecuentemente empleados en nombres de tipos de objetos, en su mayoría identificaban a tipos de componentes en los proyectos evaluados. Luego, estos términos pueden ser tratados y seleccionados para identificar en forma más precisa términos que representan tipos de componentes.
2. La identificación de tipos de componentes mediante el análisis del uso frecuente de términos en nombres, puede presentar mejores resultados al tratar proyectos de software base como frameworks que han sido desarrollados por compañías que sigan en forma disciplinada convenciones de nombrado y una buena organización de paquetes.
3. Los proyectos de aplicaciones con mayor cantidad de archivos, habitualmente hacen mayor uso de convenciones de nombrado y directivas de organización de paquetes de código fuente.

REFERENCIAS BIBLIOGRÁFICAS

- Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland, and David Svoboda. 2013. *Java Coding Guidelines: 75 Recommendations for Reliable and Secure Programs* (1st ed.). Addison-Wesley Professional.
- Hani Abdeen, Stephane Ducasse, and Houari Sahraoui. 2011. Modularization Metrics: Assessing Package Organization in Legacy Large Object-Oriented

Software. In *Proceedings of the 2011 18th Working Conference on Reverse Engineering* (WCRE '11). IEEE Computer Society, Washington, DC, USA, 394-398. DOI=10.1109/WCRE.2011.55 http://dx.doi.org/10.1109/WCRE.2011.55

- Karthik Dinakar. 2009. Agile development: overcoming a short-term focus in implementing best practices. In *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications* (OOPSLA '09). ACM, New York, NY, USA, 579-588. DOI=10.1145/1639950.1639952 http://doi.acm.org/10.1145/1639950.1639952
- Len Bass, Paul Clements, and Rick Kazman. 2012. *Software Architecture in Practice* (3rd ed.). Addison-Wesley Professional.
- Peter Macko, Daniel Margo, and Margo Seltzer. 2013. Performance introspection of graph databases. In *Proceedings of the 6th International Systems and Storage Conference* (SYSTOR '13). ACM, New York, NY, USA, , Article 18 , 10 pages. DOI=10.1145/2485732.2485750 http://doi.acm.org/10.1145/2485732.2485750
- Petra Mutzel and Peter Eades. 2001. Graphs in Software Visualization - Introduction. In *Revised Lectures on Software Visualization, International Seminar*, Stephan Diehl (Ed.). Springer-Verlag, London, UK, UK, 285-294.
- Pierre Caserta and Olivier Zendra. 2011. Visualization of the Static Aspects of Software: A Survey. *IEEE Transactions on Visualization and Computer Graphics* 17, 7 (July 2011), 913-933. DOI=10.1109/TVCG.2010.110 http://dx.doi.org/10.1109/TVCG.2010.110
- Simon Butler. 2012. Mining Java class identifier naming conventions. In *Proceedings of the 34th International Conference on Software Engineering* (ICSE '12). IEEE Press, Piscataway, NJ, USA, 1641-1643.
- Vanius Zapalowski, Ingrid Nunes, and Daltro José Nunes. 2014. Revealing the relationship between architectural elements and source code characteristics. In *Proceedings of the 22nd International Conference on Program Comprehension* (ICPC 2014). ACM, New York, NY, USA, 14-25. DOI=10.1145/2597008.2597156 http://doi.acm.org/10.1145/2597008.2597156
- Walid Maalej, Rebecca Tiarks, Tobias Roehm, and Rainer Koschke. 2014. On the Comprehension of Program Comprehension. *ACM Trans. Softw. Eng. Methodol.* 23, 4, Article 31 (September 2014), 37 pages. DOI=10.1145/2622669 http://doi.acm.org/10.1145/2622669