Herramienta para Comprensión de Programas

Asistencia a dos perfiles distintos que mantienen código

Alejo Nuñez[1], Fabiana Corina Castro Abdallah[1],

Micaela Anahí Poppino[1], Rocío Irina Salazar Lind[1]

[1] Grupo Sancor Seguros, Sunchales, Santa Fe, Argentina

alejon541@gmail.com,ingenieria_fc@hotmail.com,
micaela_a_poppino@hotmail.com, rocioirinasalazarlind@gmail.com

Resumen. La disciplina "Comprensión de programas" –CP- se define como el proceso por el cual los desarrolladores estudian y comprenden el comportamiento y operación de un sistema o software, utilizando el código fuente como referencia principal. Si se tiene un servicio restful íntegramente diseñado, desarrollado y ejecutado en un Sistema de Gestión de Reglas de Negocio, construido por personas con formación informática pero también por usuarios de negocio, existe conocimiento (información, premisas, criterios de diseño, abstracciones) que queda distribuido entre ambos perfiles complementarios. Si a eso se suman los frecuentes cambios de las normativas impositivas en Argentina que constituyen el ámbito de aplicación de este software y el dinamismo que hoy en día experimentan los equipos de desarrollo por el recambio de sus integrantes, transcurrido un lapso de tiempo no tan prolongado, el único conocimiento actualizado e íntegro queda atrapado dentro del código fuente. Como paliativo para esta problemática, se desarrolla una herramienta que da soporte al ciclo de vida propio de este sistema. El conocimiento del dominio de ambos perfiles se extrae del código fuente, para modelarlo y representarlo visualmente, colaborando con el entendimiento de quienes deban realizar tareas de mantenimiento, entrenando nuevos recursos o acompañando a los existentes.

Palabras Claves: Comprensión de Programas; Modelos; Herramientas; BRMS.

1 Introducción

En un contexto de gran dinamismo, se debe dar soporte a la evolución de un software íntegramente diseñado, desarrollado y ejecutado en un Sistema de Gestión de Reglas de Negocio (BRMS).

Su ámbito de aplicación es el de las normativas impositivas de Argentina que la empresa debe aplicar por el motivo de estar designada como agente de percepción por diversos fiscos: nacional, provinciales y municipales.

Sus veinte mil líneas de código fuente, implementan más de treinta impuestos, que al ejecutarse deciden cuáles se aplican a cada operación de venta, considerando las características del cliente combinadas a las particularidades del objeto o servicio vendido.

Ese código es el resultante de reglas, la mayoría de ellas modeladas en tablas de decisión, otra minoría con sintaxis "si-entonces"; sobre una base de desarrollos para transformaciones, validaciones, lógica de comunicación y ejecución de las reglas.

Está desplegado como un servicio restful, que es invocado por otros sistemas de la compañía que le consultan los impuestos adecuados.

Construido por personas con formación informática –IT- y por usuarios de negocio, tiene conocimiento distribuido entre ambos perfiles.

Durante la etapa inicial de mantenimiento surgen algunos riesgos y problemas.

En este específico contexto, se buscaron estrategias para evitar pérdida de conocimiento y aprovechar el ya existente en el código, pero no se encontraron antecedentes.

El alcance de este trabajo describe la solución encontrada para la problemática de evolución de este software puntual, utilizando técnicas de *CP* con la contribución de estrategias para identificar en el código fuente el conocimiento propio de cada perfil, extraerlo, procesarlo y plasmarlo en modelos diferenciados pero interrelacionados y útiles para los procesos de entrenamiento, comprensión y mantenimiento colaborativo.

2 Descripción de la Problemática

Dos roles realizan la evolución del software cada uno con dificultades distintas:

- El analista impositivo –usuario- plasma los frecuentes cambios impositivos en las tablas de decisión, utilizando una interface relativamente amigable que ofrece el IDE del BRMS. Su tarea consiste en agregar una o más combinaciones en ellas. Transcurridos varios meses, se aprecia que algunas tablas alcanzaron una dimensión considerable. Para un usuario novato o inclusive para el que tiene experiencia, el IDE no facilita hallar la explicación a cierto comportamiento o ejecución del sistema.
- El ingeniero de software adecua el código restante, que requiere un mantenimiento de menor frecuencia –mayor a seis meses promedio–. Aún tratándose de la misma persona, cuando lo revisa le resulta desconocido. Con el vertiginoso recambio que sufren los equipos de desarrollo, es probable que la tarea la realice otra persona.

Ambos perfiles, gestionan conocimientos de distinta índole o dominios, pero el rol de IT es quien tiene el proceso de comprensión más complejo, porque debe vincularlos: "*Durante la implementación del sistema el desarrollador va construyendo (desde su perspectiva) la relación entre los dominios y de alguna manera la misma se encuentra activa durante el ciclo de desarrollo. Sin embargo, en la etapa de mantenimiento y evolución del software es muy posible que dicha relación pierda fuerza y en determinados casos llegue a desaparecer*" [1].

Pronto el sistema deberá incorporar la operatoria de nuevas empresas, impuestos, clases de operaciones de ventas; en conclusión: más conocimiento y complejidad.

Sin documentación adecuada, actualizada, y menos aún que vincule ambos dominios de conocimiento, será dificultosa la evolución del sistema.

Con la certeza que el código fuente tiene y tendrá encapsulado el conocimiento actual y unificado, el problema es brindárselo filtrado, simplificado, modelado, cuándo y cómo cada uno lo requiera.

3 Solución Propuesta

Considerando la línea de investigación expuesta en los artículos [1,2,3,4], se mapearon algunos conceptos al contexto puntual de este trabajo y se propuso una estrategia de extracción diferenciada para ambos dominios:

- *Dominio del Problema*: comportamiento/salida de normativas impositivas.
- *Dominio del Programa*: operación -rutinas, parámetros, variables-, que dan el marco de funcionamiento a las reglas del comportamiento impositivo.

Luego se tomaron técnicas de la *CP* para efectuar los procesos de filtrado, simplificación, unificación, abstracción hasta obtener los modelos deseados para la *Representación del Problema* -relativos al Dominio del Problema-, y/o para la *Representación del Programa* -relativos al *Dominio del Programa*- vinculados a los anteriores.

Todos esos procesos se automatizaron desarrollando una nueva herramienta.

4 Aplicación

La herramienta (LHCP) se construyó con distintas piezas, interconectando la salida de unas con las entradas de otras (ver Fig. 5). Al ejecutarla, toma el código fuente y lo somete a un *proceso de ingeniería*, de acuerdo a lo que plantea la *CP* [3].

4.1 Extracción de conocimiento del Dominio del Problema

Denominado como "*Extractor R*" se construyó un microservicio que es consumido sin parámetros de entrada; invoca métodos de una API que provee el BRMS; recupera condiciones, acciones y valores de las reglas; y finalmente los procesa dándole una estructura de salida útil para la siguiente pieza en el proceso (ver Fig.1).

IVA Débito Fiscal > **Aplica IVA General**

Ubicación del riesgo	Fecha de la operación		IVA - Débito Fiscal	
	Fecha desde	Fecha hasta	Alícuota	Tipo Alícuota
Buenos Aires	≥ 01-jul-2020		21	porcentaje
Capital Federal	≥ 01-jul-2020		21	porcentaje
Catamarca	≥ 01-jul-2020		21	porcentaje
. . .	≥ 01-jul-2020		21	porcentaje
Tucumán	≥ 01-jul-2020		21	porcentaje


```
{
 "Regla": {
     "name": "Aplica IVA General",
     ...
     "Filas": {
         "Item": {
             "Columnas": {
                 "Item": {
                     "NombreColumna": "C3",
                     "Valor": "Buenos Aires"
                 },
                 "Item": {
                     "NombreColumna": "C0",
                     ...,
                     "Valor": "01/07/2020 …"
                 },
                 "Item": {
                     "NombreColumna": "A0",
                     "Valor": "21",
                     "Valor": "porcentaje"
                 }
```


Fig. 1. Ejemplo de Tabla de Decisión (izquierda) de la que se extrajo información (derecha).

Dado que las reglas contienen la información de la mayor parte del conocimiento del Dominio del Problema, se logra así obtener la mayoría del conocimiento del dominio.

4.2 Extracción de conocimiento del Dominio del Programa

Construido como un analizador sintáctico, el denominado "Extractor C", recupera el flujo de invocaciones entre métodos, seteos o utilizaciones de variables-bandera, a partir de ciertos puntos candidatos para la extracción del conocimiento (como por ejemplo: variables-bandera o *verbalizaciones*), ideales para vincular ambos dominios; logrando representaciones útiles para los roles de IT.

Las *verbalizaciones* - en lenguaje natural del usuario- son "apodos", colocados a atributos o métodos de las clases que conforman el código fuente (Dominio del Programa) y también son utilizadas en las precondiciones, condiciones o en las acciones de las reglas (Domino del Problema).

Con otros puntos candidatos para extracción –como las clases de "entrada" y "salida" del servicio-, se completa la recuperación del conocimiento de la operación global.

4.3 Filtrado, Simplificación, Unificación y Conversión

Se construyeron componentes que permiten escoger *filtros* para segmentar la información obtenida por los extractores; así el conocimiento queda acotado a lo que interesa comprender. Los filtros ofrecidos al analista impositivo son las condiciones "denominador común" de las tablas, como por ejemplo "*Periodo"* (ver Fig. 3); y al desarrollador, se le permite elegir *verbalizaciones* (ver Fig. 4), variables-bandera, etc.

Luego se realizan procesos de simplificación resumiendo información; de complementación explicitando los circuitos-complemento y/o de unificación para la conformación de vistas globales cuidando de quitar redundancias o duplicados.

Se pueden obtener representaciones parcializadas según se requiera. Esto se debe a que normalmente el proceso de comprensión de programas implica visualizar aspectos de alto, medio y bajo nivel del sistema [4].

La información resultante se persiste en un *Repositorio de Información* y se estructura como *matrices de aristas* (ver Fig. 2) que servirán de entrada para la última pieza.

Nodo Origen	Nodo Destino	Rótulo Arista	Atributos
REGLA Agente de Reten y o percepcion	VERB "No aplica IVA Percepción General"	entonces	
...	VERB "No aplica IVA Percepción General"	entonces	
REGLA Exceptuado	VERB "No aplica IVA Percepción General"	entonces	
REGLA Tierra del Fuego	VERB "No aplica IVA Percepción General"	entonces	
REGLA Recupero de gastos	VERB "No aplica IVA Percepción General"	entonces	
VERB "No aplica IVA Percepción General"	VAR this.excepcionIVARegimenGeneral	ACCION (SET)	
VAR this.excepcionIVARegimenGeneral	VERB "no tiene excepciones del régimen genera	Navegacion (GET)	
VERB "no tiene excepciones del régimen general de percepción"	TABLA IVA Percepción General	Precondición	https:\\....

Fig. 2. Ejemplo de Matriz de Aristas para Verbalización "No aplica IVA Percepción General".

4.4 Visualización

La pieza desarrollada toma cada matriz de aristas y genera multigrafos como modelos de visualización para obtener las representaciones para ambos roles. Son útiles para evidenciar inter-relaciones, caminos o circuitos y las direcciones de cada relación.

En la Representación del Problema, los nodos son condiciones o acciones de las tablas decisión y los rótulos de las aristas indican los valores de las primeras (Ver Fig.3).

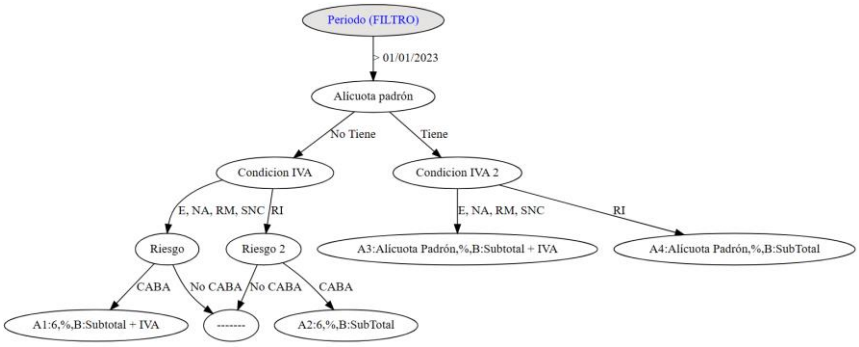

Fig. 3. Ejemplo: REGLA "IIBB CABA-NS", filtrada por PERIODO

Para las Representaciones del Programa, los nodos se destinan a verbalizaciones, nombres de métodos, variables o reglas/tablas –estas últimas pertenecientes al Domino del Problema-. Los rótulos de las aristas tienen significancias técnicas (Ver Fig. 4).

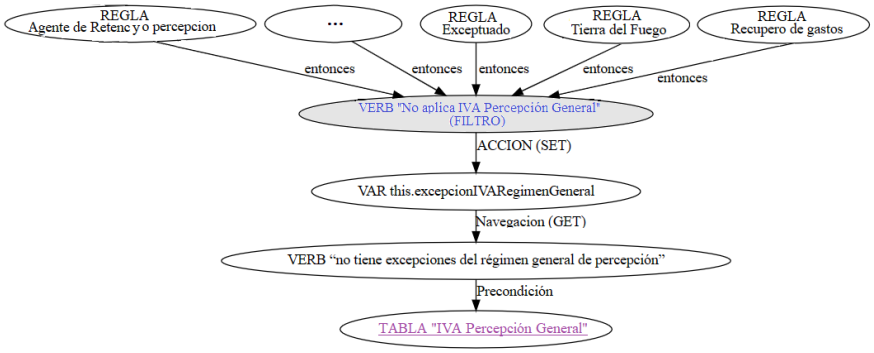

Fig. 4. Ejemplo: Verbalización "No aplica IVA Percepción General" (enlace en el nodo inferior).

Hay enlaces incorporados para facilitar la navegación al punto del código fuente de interés. Así el *proceso de ingeniería* queda materializado dentro de LHCP como un bucle, que permite el acceso directo al conocimiento que cualquiera de los roles está aprendiendo, repasando o revisando para posibles adecuaciones (ver Fig.5).

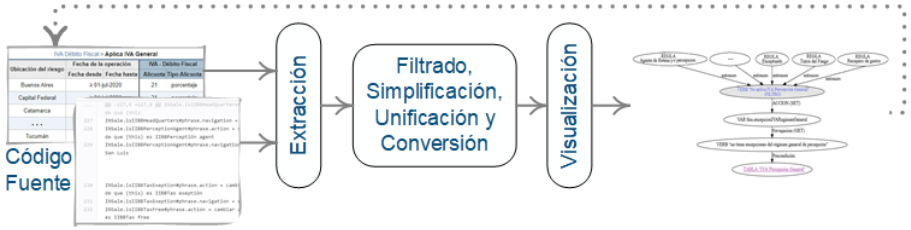

Fig. 5. Proceso de Ingeniería para Comprensión del Servicio.

5 Conclusiones

LHCP recupera, modela y visualiza el conocimiento con representaciones adecuadas para ambos dominios; y también provee facilidades para poder escoger entre una representación global u otra enfocada a la porción seleccionada, con la suficiente precisión y utilidad, sin abrumar con otra información innecesaria.

Como contribuciones de interés se propusieron estrategias para lograr la extracción del conocimiento del Dominio del Problema y luego interrelacionarlo al del Programa, aprovechando facilidades de la implementación particular del servicio en el BRMS.

Recuperando el conocimiento y generando documentación en forma automática, siempre se dispone de una versión actualizada e íntegra, coherente con la operación actual del servicio, dando el soporte deseado para cada tarea de mantenimiento.

Los *modelos del Problema* obtenidos, permiten al usuario novato o al usuario entrenado focalizarse en la representación filtrada del comportamiento para una normativa impositiva seleccionada, visualizado la información que solo le es útil en ese momento; junto a posibles premisas de implementación.

El desarrollador, utilizando los modelos del Programa interrelacionados a los modelos del Problema, puede navegar de un modelo a otro o del modelo-grafo a la plataforma BRMS; entrenarse o repasar cualquiera de los dominios que le resulte desconocido y reconstruir la relación entre ellos; escoger distintos niveles de análisis, desde lo global hacia lo particular, visualizando dependencias múltiples o seriadas de las componentes del sistema -en contraste con las dependencias inmediatas que el IDE de desarrollo le provee-; y agilizar el entendimiento de las relaciones complejas o series de relaciones entre las clases de código definidas, comprendiendo la propagación de cierta operación ante un ajuste que se requiere realizar al servicio.

Como próxima evolución a LHCP se agregará una funcionalidad para que la extracción dinámica posibilite conocer cuáles componentes son utilizadas para una ejecución específica [3]. Los logs históricos de ejecuciones, se utilizarán para indicar el camino recorrido sobre los grafos estáticos ya generados –Repositorio de Información-, logrando así una representación dinámica para cada ejecución. Resolverá una problemática adicional, que justificará *qué* impuestos respondió el servicio y además *por qué*.

Referencias

1. Enrique A. Miranda, Corina Abdelahad, Mario Berón, Daniel E. Riesco, *Técnicas, estrategias y herramientas de comprensión de programas para facilitar el entendimiento de sistemas multiparadigmas*, http://sedici.unlp.edu.ar/handle/10915/103992 (2020)
2. Enrique A. Miranda, Mario Berón, Germán Montejano, Daniel Riesco, Pedro R. Henriques, Maria J. Pereira, *Visualización de Software: Conceptos, Métodos y Técnicas para Facilitar la Comprensión de Programas*, http://sedici.unlp.edu.ar/handle/10915/20111 (2011)
3. Mario M. Berón, Pedro R. Henriques, Roberto Uzal y Maria J. Varanda Pereira: *Comprensión de Programas, http://sedici.unlp.edu.ar/handle/10915/19748* (2009)
4. Mario M. Berón,Roberto Uzal,Pedro R.Henriques,Maria J.Varanda Pereira: *Comprensión de Programas por Inspección Visual y Animación*, http://sedici.unlp.edu.ar/handle/10915/20382 (2007)