A proposal to Assist the Mashup Co-evolution when web APIs evolve

Una propuesta para asistir a la Coevolución de Mashup cuando las APIs web evolucionan

ARTICLE HISTORY

Received 01 September 2021
Accepted 23 November 2021

Sandra Casas
GISP – Instituto de Tecnología Aplicada
Universidad Nacional de la Patagonia Austral
Río Gallegos – Santa Cruz
sicasas@uarg.unpa.edu.ar

Graciela Vidal
GISP – Instituto de Tecnología Aplicada
Universidad Nacional de la Patagonia Austral
Río Gallegos – Santa Cruz
gvidal@uarg.unpa.edu.ar

Franco Herrera
GISP – Instituto de Tecnología Aplicada
Universidad Nacional de la Patagonia Austral
Río Gallegos – Santa Cruz
fherrera@uarg.unpa.edu.ar

Una propuesta para asistir a la Coevolución de Mashup cuando las APIs web evolucionan

A proposal to Assist the Mashup Co-evolution when web APIs evolve

Sandra Casas
GISP – Instituto de Tecnología Aplicada
Universidad Nacional de la Patagonia Austral
Río Gallegos – Santa Cruz
sicasas@uarg.unpa.edu.ar

Graciela Vidal
GISP – Instituto de Tecnología Aplicada
Universidad Nacional de la Patagonia Austral
Río Gallegos – Santa Cruz
gvidal@uarg.unpa.edu.ar

Franco Herrera
GISP – Instituto de Tecnología Aplicada
Universidad Nacional de la Patagonia Austral
Río Gallegos – Santa Cruz
fherrera@uarg.unpa.edu.ar

***Resumen*—**A medida que evolucionan las interfaces de programación de aplicaciones web (APIs), los contratos establecidos previamente cambian y por lo tanto, pueden afectar el comportamiento, funcionamiento y ejecución de aplicaciones de consumo como Mashup. En estos casos, estas aplicaciones necesitan ser actualizadas para seguir funcionando, es un proceso llamado coevolución. Identificar y localizar las operaciones que se ven afectadas por la evolución de las APIs web y estimar el impacto que generan son tareas necesarias que ayudan al desarrollador a actualizar el código. Este trabajo presenta una propuesta para asistir a la coevolución de Mashup. Específicamente a partir de un grafo de operaciones de mashup identificamos y ubicamos las operaciones afectadas por algunos cambios en las APIs web. También proponemos un conjunto de métricas simples, las que permiten estimar el impacto de estos cambios en el mashup. El grafo y las métricas de operaciones de Mashup ayudan a los desarrolladores web en las tareas de coevolución. La propuesta fue aplicada a dos mashup que actualmente se encuentran disponibles en la web. Los resultados preliminares muestran que la propuesta es aplicable.

***Index Terms*—** *impacto del cambio de API, APIs web, co-evolution, web mashup*

***Abstract*—** As web application programming interfaces (APIs) evolve, previously established contracts change and therefore can affect the behavior, operation, and execution of consumer applications such as Mashup. In these cases, these applications need to be updated to continue working, it is a process called coevolution. Identifying and locating the operations affected by the evolution of web APIs and estimating the impact they generate are necessary tasks that help the developer to update the code. This work presents a proposal to assist the coevolution of Mashup. Specifically, from a mashup operations graph we identify and locate the operations affected by some changes in the web APIs. We also propose a set of simple metrics, which allows us to estimate the impact of these changes on the mashup. Mashup's graph and operations metrics assist web developers in coevolution tasks. The proposal was applied to two mashups that are currently available on the web. Preliminary results show that the proposal is applicable.

***Index Terms*—** *API change impact, web APIs, co-evolution, web mashup*

I. Introducción

Las APIs (Interfaz de Programación de Aplicaciones) permiten reutilizar componentes de software en el contexto del desarrollo de software y así mejorar la productividad [1][2]. En particular las APIs web facilitan el intercambio de datos inter e intra organizacionales disponibles a través de puntos finales específicos (URL) [3].

Los Mashups [4] invocan a las APIs web mediante el envío de solicitudes HTTP a una URL dedicada utilizando alguno de sus métodos HTTP compatibles; los datos requeridos son enviados como parámetros. De esta forma, la aplicación consumidora accede a servicios web, utilizando APIs web, invocadas a través de la red que dependen de tecnologías web como protocolos de transporte o XML (eXtensible Markup Language) y JSON (Javascript Object Notation) como formatos de datos. En la práctica, estas APIs son a menudo del estilo arquitectónico REST (Transferencia de Estado representativo) [5].

El extendido uso de APIs ha mostrado nuevos problemas a estudiar y resolver en relación a la documentación, usabilidad, seguridad, etc [6][7][8][9], y el que atañe a esta propuesta, la asistencia a la coevolución de los Mashups ante cambios y evolución de las APIs web.

Se ha denominado coevolución [10], cuando el software del cliente debe modificarse para mantenerse compatible con los cambios de las APIs. En el caso de APIs web, puede ser crítico, en tanto las aplicaciones clientes pueden perder la robustez, disponibilidad y confiabilidad, si la evolución tiene efectos conocidos como “breaking changes” y requiere que se reescriba el código correspondiente [11][12]. Los cambios de metadatos o cambios sintácticos y cambios semánticos, pueden requerir un esfuerzo significativo de reparación e incluso descubrimiento a partir del fallo.

Por otro lado, varias plataformas facilitan el desarrollo de mashups simples al ofrecer entornos visuales que permiten a los usuarios seleccionar componentes predefinidos y combinarlos, tales como PIPES, DeployPlace y Netvibes, entre otras. También es posible desarrollar mashups a partir de frameworks web o lenguajes como Javascript o PHP. Mientras las herramientas mashups se enfocan principalmente

ISSN:1390-9266 e-ISSN:1390-9134 LAJC 2021

en modelar el flujo de datos y el desarrollo rápido, se observa que existen menos enfoques y herramientas dedicados a la resolución de aquellos problemas que se ocasionan debido a la evolución de los componentes externos [13].

Con el objetivo de facilitar la coevolución de mashups, como consecuencia de la evolución de las APIs web, este trabajo propone asistir a desarrolladores en la identificación y cuantificación de los efectos de los cambios. En concreto se propone:

(i) identificar los impactos a partir de un grafo de operaciones y flujos de datos, que representa la funcionalidad del mashup que consume las APIs web;

(ii) estimar el impacto de los cambios a partir de dicho grafo, mediante la aplicación de un conjunto de métricas.

En cuanto a los tipos de cambios, ocasionados por la evolución de las APIs, se ha escogido un subconjunto, reportados en otros estudios [14][15]. Esta propuesta no está vinculada directamente a una herramienta específica de desarrollo o edición, aun así, se presenta una herramienta prototipo que automatiza el proceso de identificación de operaciones impactadas y la estimación basada en métricas.

En la Sección II se exponen algunos problemas que motivan este trabajo, en la Sección III se describe el enfoque propuesto, en la Sección IV se aplica el enfoque a dos mashups disponibles en la web, en la Sección V se mencionan los trabajos relacionados y en la Sección VI se desarrollan la discusión y las conclusiones.

II. Problemas y motivaciones

La evolución de las APIs web es un área de estudio muy reciente [10]. La criticidad de la problemática radica en el hecho de que cuando una solicitud se dirige a una URL que no existe o envía datos que no cumplen con los requisitos de la API web, se produce un error en tiempo de ejecución. Este mecanismo de invocación frecuente para las APIs web, no permite la verificación de seguridad de tipos. Diversos cambios producto de la evolución de las APIs web [14][15] [16] [17] han sido reportados.

A. Escenarios y problemas

Supongamos un web mashup muy simple e imaginario, que consiste en que dada una personalidad, se rastrea su actividad reciente en las redes sociales Facebook y Twitter. El gráfico de la Figura 1 muestra una secuencia de operaciones posibles que requiere el desarrollo de la situación propuesta.

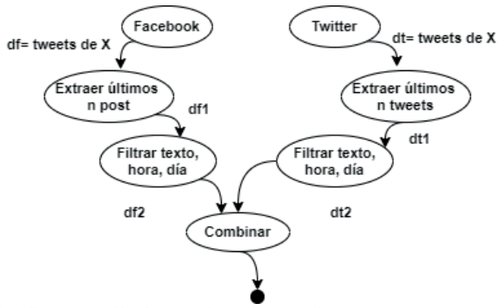

Fig.1 Representación de operaciones de un mashup

La implementación del mashup inicia con las peticiones de los posteos y tweets de una personalidad (X), a continuación en cada caso se extrae una determinada cantidad de ítems (n) de la respuestas (df y dt) en JSON o XML, y posteriormente se filtran los datos que en particular interesan (texto, hora y día de los posteos y tweets), para finalmente generar la salida, producto de la combinación resultante.

En la Figura 2, se han transcripto parcialmente las peticiones a las APIs web de Facebook y Twitter y sus respuestas en formato JSON. En caso de Facebook (a), la petición indica el identificador de la página y el post requeridos, por defecto la respuesta (b) incluye los campos created_time, message e id, sin embargo el usuario puede especificar los campos. En el caso de Twitter (c), la petición retorna una colección de tweets relevantes que coinciden con el screen_name solicitado (nombre del usuario). La respuesta (d) incluye no solo los tweets del usuario, sino también los retweets y replicas en cada caso. En ambos casos se requiere la misma información pero las peticiones son diferentes, y aunque retornan los resultados en JSON, las estructuras y contenidos de las respuestas no son compatibles. Algunas situaciones que pueden generar fallos o errores en la ejecución de este simple e imaginario mashup son las siguientes:

(i) alguna de las URL de las peticiones no funciona debido a que el proveedor cambió la ruta;

(ii) cierta petición no funciona dado que la lista de parámetros enviada no coincide con la lista esperada;

(iii) las operaciones de filtrado no funcionan porque los valores de mes y/o año cambiaron de formato;

(iv) alguna de las operaciones de filtrado no funciona debido a que se ha eliminado alguno de los campos que requiere el mashup;

(v) la operación combinar no se puede ejecutar en razón que una de las entradas ha producido un error.

```
FB.api(
        "/{page_id}_{post_id}/",
        function(response){
        if(response && !response.error){
              /* handle the result*/
        } } );
                              (a)
{  "data": [
    { "created_time":"2017-12-08T01:08:57+0000",
"message":"Love this puzzle.One of my favorite puzzles",
"id": "post-id" },
    { "created_time": "2017-12-07T20:06:14+0000",
"message": "You need to add grape as a flavor.",
"id": "post-id"}  ] }
                              (b)
https://api.twitter.com/1.1/statuses/
user_timeline.json?screen_name=twitterapi&count=2
                              (c)
[
{ "created_at": "Thu Apr 06 15:28:43 +0000 2017",
  "id": 850007368138018817,
  "id_str": "850007368138018817",
  "text": "RT @TwitterDev: 1/ Today we're sharing our
vision for the future of the Twitter API
platform!nhttps://t.co/XweGngmxlP","truncated": false,
                              (d)
```

Fig.2 Solicitud y respuesta de las APIs web de Facebook y Twitter

Además de lidiar con estos cambios externos, otra dificultad que se presenta a la hora de gestionar la co-evolución se refiere a la localización de las operaciones afectadas en el código del mashup. Las operaciones de acceso

a las fuentes, filtrar, extraer, combinar u otras, están inmersas en la codificación de diferentes formas, podrán ser cada una modularizada o no, en funciones o métodos, situación que varía según las herramientas de desarrollo usadas (lenguaje de script, frameworks, etc.) y por sobre todo, el estilo de programación del programador.

B. Cambios y evolución de APIs web

El estudio de [17] encontró que un número significativo de aplicaciones móviles fallará a la luz de los cambios en las APIs web que consumen. En [6] se enfatiza la necesidad de generar soluciones al soporte mediante herramientas para desarrolladores que escriben código que usan APIs web, automatización de tareas manuales, tediosas y propensas a errores, sobre los cuales hay algunas propuestas [18], aún insuficientes.

En particular para este trabajo, se escoge un subconjunto de cambios, que posteriormente será ampliado. A continuación, se describen brevemente:

(i) Remoción de campos (RC) [17]. Consiste en la eliminación de uno o más campos de la respuesta de la API web.

(ii) Cambios en los tipos de datos (CTD) [16]. Consisten en dos tipos de cambios, reemplazar un entero por un string o la operación inversa. Esto genera un problema especialmente si el mashup está codificado en un lenguaje tipado o los datos se van a aplicar a operaciones de tipo (aritméticas u otras).

(iii) Modificación de los puntos finales (MPF) [14]. Los proveedores de API exponen recursos al proporcionar puntos finales para acceder a ellos. Los proveedores pueden desconectar los puntos finales (eliminar), agregar nuevos, cambiar la ruta o incluso reemplazar un punto final existente por uno nuevo.

(iv) Modificación de los parámetros (MP)[14]. Los parámetros se utilizan para refinar recursos, pueden ser parámetros de ruta o de consulta. Los parámetros pueden ser opcionales u obligatorios. Los proveedores de API pueden agregar nuevos parámetros a los recursos, los existentes se pueden eliminar, renombrar, cambiar el tipo o incluso cambiar de opcional ha requerido, y viceversa.

(v) Cambios en los niveles de autoridad (CA) [14]. Para interactuar con recursos específicos, los usuarios deben tener el nivel de autoridad requerido. El conjunto de niveles de autoridad para un recurso puede cambiar agregando nuevos niveles, eliminando o simplemente cambiando la autoridad necesaria.

III. Asistencia a la coevolución de mashup

La propuesta tiene por objetivo descubrir qué operaciones y sectores de la implementación del mashup consumidor pierden robustez o se quiebran ante un cambio en la respuesta de la API del proveedor. Los artefactos que la integran son un grafo denominado Grafo de Operaciones Mashup (GOM) y un conjunto de métricas. El grafo representa operaciones propias de los entornos de desarrollo rápido tales como Yahoo Pipes, Pipes Digital o Microsft Popfly.

A. Grafo de Operaciones Mashup

De manera general y simplificada se define el Grafo de Operación de Mashup (GOM), $G = <O, F>$ como un grafo dirigido y acíclico que expresa las dependencias de datos y operaciones implementados en un mashup, dónde,

- *O es un conjunto finito de operaciones* $O = \{o_1, o_2, ..., o_n\}$, *y n es el número de operaciones y* $n > 2$;
- *F es un conjunto finito de flujos de datos entre las operaciones del mashup* $F = \{f_1, f_2, ..., f_m\}$, *y m es el número de dependencias y* $m > 1$.

Las operaciones, que constituyen los nodos o vértices del grafo, tienen su propia semántica, pero es posible en cada caso definirlas como tuplas que se describen al menos por los siguientes elementos, *o = <tipo, identificador, DE, DS, PREC, código>*. Donde *tipo* define el comportamiento de la operación específica en un conjunto previamente establecido. En este caso, *tipo = source o combine o filter o extract o merge o duplicate o build doc, identificador* es un atributo que identifica a la operación particular, *DE* y *DS* son conjuntos de flujos de datos de entrada y salida respectivamente. Por lo general *DE* y *DS* son colecciones de estructuras de datos compuestas por datos simples, pueden implementarse de diferentes formas (XML, JSON, vector, variables, u otro). *PREC* es un conjunto de condiciones o restricciones que deben cumplirse para que la operación *o*, se pueda ejecutar, las mismas están acotadas al *tipo* de operación y a los flujos de datos de entrada, *DE*. Por último, *código* es la referencia a la implementación de la operación en el mashup, pudiendo ser los identificadores de clase, método, función u otro. Este último atributo es el que permite ubicar o localizar a la operación en los programas específicos independientemente de las herramientas usadas (Javascript, PHP, Java, etc.).

El grafo siempre inicia con operaciones que acceden a las APIs web y luego se aplican las diferentes operaciones que procesan estas entradas, generando salidas diversas. Las operaciones representadas en el GOM son:

(i) Source: Entrada de datos externa (APIs, RSS, etc.). Se debe especificar la URL sobre el cual se obtienen los datos en un formato determinado (XML, JSON). Las condiciones refieren a que la URL y listas de parámetros deben ser válidas y los niveles de autorización son adecuados.

(ii) Duplicate: genera dos estructuras idénticas al *DE* sin realizar manipulación de los datos.

(iii) Combine: Combina dos o más flujos de datos de entrada en una sola salida. Se requiere que las estructuras de los DE sean consistentes (contenido, orden y cantidad de elementos).

(iv) Merge: Combina los ítems de dos flujos de datos de entrada, uno a continuación del otro respetando el orden de las entradas. Se requiere que las estructuras de los DE sean consistentes (contenido, orden y cantidad de elementos).

(v) Extract: Extrae contenido de un flujo de datos de entrada a partir de un elemento indicado por el desarrollador. Todos los elementos coincidentes serán el contenido del flujo de datos de salida de esta operación. Se requiere que los elementos de la comparación existan en DE y mantengan el formato.

(vi) Filter: A partir de una palabra clave o una expresión regular y un campo especificado, filtra información de un flujo de entrada. Se requiere que el término de búsqueda se encuentre disponible en el DE y no vacío.

(vii) Build Doc: A partir de varios flujos de entrada de los cuales se seleccionan diferentes contenidos, se crea una salida específica. Se requiere que al menos deben existir dos *DE* entradas disponibles y en el formato esperado.

ISSN:1390-9266 e-ISSN:1390-9134 LAJC 2021

En la Tabla I, se detalla que tipo de cambio puede afectar a cada operación si no se cumplen las precondiciones definidas para cada operación.

TABLA I. RELACIÓN OPERACIONES Y TIPOS DE CAMBIO

Operación	Tipo de cambio
Source	Modificación de puntos finales (MPF) Modificación de los parámetros (MP) Cambio nivel de autoridad (CNA)
Duplicate	---
Combine	Remoción de Campos (RC) Cambio en los tipos de datos (CTD)
Merge	Cambio en los tipos de datos (CTD) Remoción de Campos (RC)
Extract	Remoción de Campos (RC) Cambio en los tipos de datos (CTD) Modificación de los parámetros (MP)
Filter	Remoción de Campos (RC) Cambio en los tipos de datos (CTD)
Build Doc	Cambio en los tipos de datos (CTD) Remoción de Campos (RC)

B. Métricas para analizar el impacto

Para dimensionar los efectos que un cambio c_i ocasiona en un mashup se proponen un conjunto de métricas calculadas a partir del GOM, las cuales son presentadas en la Tabla II.

TABLA II. METRICAS

Métrica	Descripción
QOM	Cantidad de operaciones del mashup
QOMP	Cantidad de operaciones del mashup de procesamiento (excluye a las entradas)
QI (c_i)	Cantidad de operaciones impactadas por el cambio c_i.
QID (c_i)	Cantidad de operaciones directamente impactadas por el cambio c_i
QII (c_i)	Cantidad de operaciones indirectamente impactadas por el cambio c_i
IT(c_i)	Impacto de cambio c_i. Se obtiene de QI/QOM
ID (c_i)	Impacto directo del cambio c_i. Se obtiene de QID/QOM
II (c_i)	Impacto indirecto del cambio c_i. Se obtiene de QII/QOM
IP(c_i)	Impacto en la operaciones de procesamiento (excluye las entradas) QI/QOMP

Donde QI equivale a QID + QII, e IT a ID + II. Las operaciones impactadas por un cambio c_i (QI) es un conjunto formado por operaciones en las que el cambio ocasiona que no se cumpla al menos una de las precondiciones (PREC) establecidas (operaciones directamente impactadas - QID) y el conjunto de operaciones sucesoras o adyacentes a esta (operaciones indirectamente impactadas - QII) en el grafo dirigido.

En cuanto a las métricas IT, ID, II e IP, son proporciones que se establecen de manera simple en relación con la/s operación/es impactadas y todas las operaciones del grafo. Este conjunto de métricas permite analizar y dimensionar el efecto y propagación de un determinado cambio. La cantidad de operaciones impactadas (QI) proporciona una estimación de la cantidad de código y/o partes de código a rever y modificar. Los valores de IT, ID, II e IP oscilan entre $0 >= x <= 1$, resultando más grave cuanto más se acerque a 1. Lo que implicaría que más código del mashup debe ser revisado y modificado ante un cambio.

Para ilustrar el esquema presentado, la Figura 3 representa al GOM de la situación planteada en la sección II.

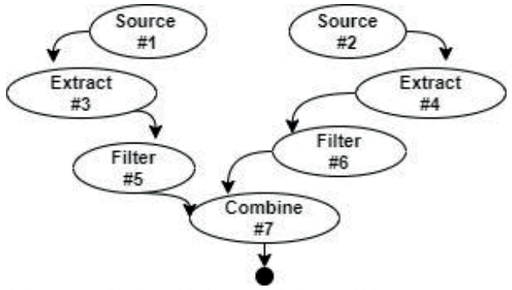

Fig.3. Representación de un Mashup a través de un GOM

Es un mashup de siete operaciones, dos de entrada (Source #1:Facebook y Source #2:Twitter) y el resto de procesamiento (filter, extract, etc). Suponiendo que se produce un cambio, en el API web de Twitter, coincide con la operación Source #2, a partir de allí, se analiza el camino, (<Extract#4, Filter#6, Combine#7>). Para cada una de las operaciones de este camino se considera su vulnerabilidad ante el cambio (Tabla I). Si el cambio rompe alguna de las precondiciones de Extract#4, entonces se consideran todos los caminos para calcular como afecta el cambio. Resultando, QOM = 7, QOMP = 5 (<Extract#3, Extract#4, Filter#5, Filter#6 y Combine#0>7), QI = 3 (<Extract#4, Filter#6 y Combine#7>), QID es 1 (<Extract#4>), QII es 2 (<Filter#6 y Combine#7>). Luego IT = 0,43, ID = 0,14, II = 0,29 y IP = 0,6.

C. Implementación del GOM y cálculo de métricas

Se ha desarrollado una herramienta prototipo, que implementa el GOM mediante una matriz de adyacencia. Las operaciones son objetos que disponen de todos los datos definidos en la sección 3.A, o = <tipo, identificador, DE, DS, PREC, código>. En el caso de la operación Source, además se describe la API web y como precondición la URL base o el punto final al cual se realizan las solicitudes, pueden ser más de uno. Los cambios también son objetos que describen ciertos aspectos tales como el API web que lo produce, el tipo de cambio y atributos del mismo, como se explicó en la sección 2.B.

El prototipo automatiza la identificación de la/s operación/es impactadas por un determinado cambio y el cálculo de las métricas. Entonces, dado un GOM y un cambio determinado, el algoritmo primero ubica la operación afectada. Para ello, inicia evaluando todas las operaciones de tipo "source", hasta identificar la coincidencia con el API web del cambio. A partir de este vértice, luego recorre todos los caminos que desde el mismo se inician. En el recorrido de cada camino se analiza por cada operación incluida (vértice), si es afectada por el tipo de cambio (Tabla I). En el caso que la operación es vulnerable al cambio, entonces se verifica el cumplimiento de las precondiciones. A partir de estas evaluaciones se determinan los cálculos de las métricas. Para ello, solo es necesario continuar recorriendo los caminos. Por último, como cada operación del grafo, también referencia el código de la implementación, además de identificar la operación se identifica el sector en los programas que la contiene y que deberán ser examinados.

La Figura 4 muestra la interfaz de la herramienta, en la parte superior se visualiza el grafo de operaciones que representa el

mashup en estudio, luego las referencias a las operaciones. A continuación, se presenta el tipo de cambio que se ha realizado (texto azul), que incluye la API en la cual se produjo y las condiciones modificadas. El prototipo además muestra las operaciones afectadas directamente por el mismo y una referencia al código que debe ser revisado (texto rojo). Finalmente, se listan los resultados de las métricas que permiten analizar el impacto del cambio realizado en la API web sobre el mashup.

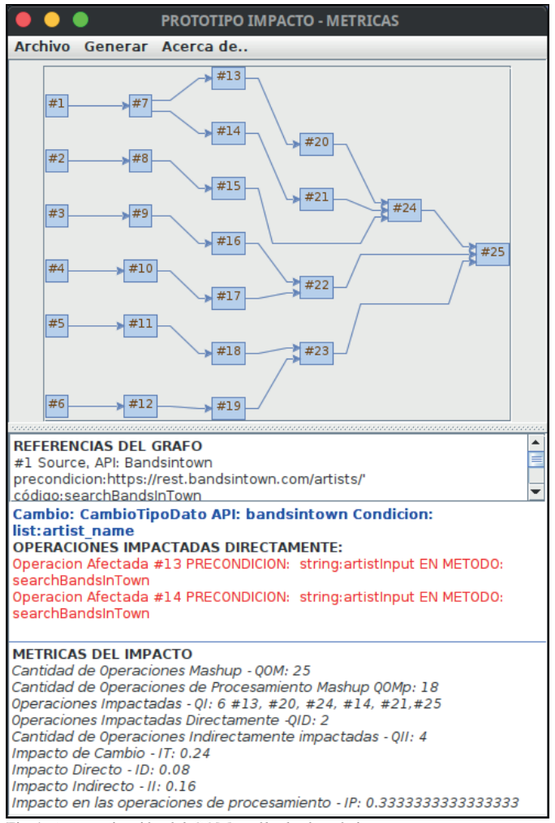

Fig.4. Automatización del GOM y cálculo de métricas

IV. Aplicaciones

En esta sección, se representa la aplicación de la propuesta a dos mashups. En ambos casos se ha analizado manualmente la implementación de estas aplicaciones para la generación de los GOM, las mismas están disponibles en el repositorio GitHub (https://github.com) y están implementadas en HTML y Javascript.

Kiteflite

La web mashup Kiteflite (https://clayjuneau.com/Kiteflite) es un proyecto desarrollado en la Universidad de Texas A&M, esta aplicación integra las APIs web OpenWeatherMap, SkyScanner y Yelp. Kiteflipe permite al usuario encontrar la ubicación ideal (20 estados de EEUU) para volar una cometa, según las condiciones climáticas.

En la Figura 5, se presenta el GOM que representa las operaciones del mashup Kiteflite. El grafo consta de veinte operaciones, de las cuales 2 son de tipo source, 4 realizan copia de los datos de entrada (duplicate) y catorce son de procesamiento.

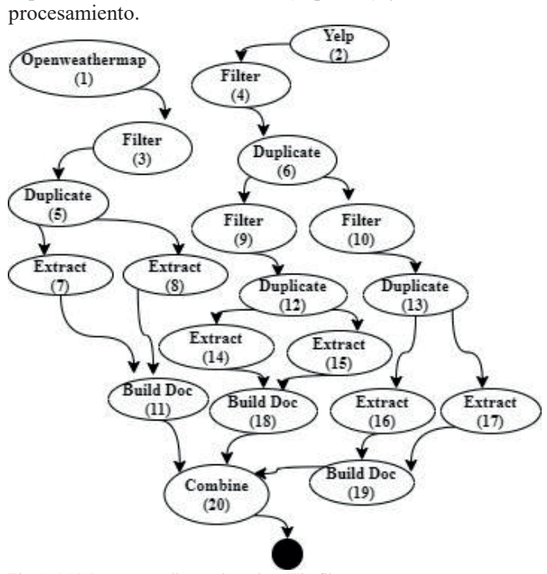

Fig.5. GOM correspondiente al mashup Kiteflite

Se han analizado los cambios #1, #2 y #3 sobre el GOM, que afectan directamente las operaciones 11, 4 y 20, del mashup kiteflite. En la Tabla III se presentan los detalles.

TABLA III. ANALISIS DE CAMBIOS

CAMBIO #1
Tipo: Remoción de Cambio API: OpenWeatherMap Condición: {campo = speed ∉ DE}
OPERACIÓN AFECTADA
Tipo: Build Doc, Identificador:#11, PREC: {campo = speed ∈ DE } Código: weather.js – index.htm
CAMBIO #2
Tipo: Cambio de Tipo de Dato API: YELP Condición: {campo = city, tipo = String ∉ DE}
OPERACIÓN AFECTADA
Tipo: Filter, Identificador:#4, PREC: {campo = city, tipo= String ∈ DE} Código: yelp.js, yelp-bundle.js, index.html
CAMBIO #3
Tipo: Remoción de campo API: YELP Condición: {campo = businesses ∉ DE}
OPERACIÓN AFECTADA
Tipo: Combine, Identificador:#20, PREC: {campo = businesses ∈ DE} Código: yelp.jss, index.html

La operación Build doc (11) requiere el campo *speed* dentro de su estructura, el cual ya no está disponible. En este caso será necesario examinar el código de weather.js e index.html. La operación Filter (4) utiliza el campo city de

ISSN:1390-9266 e-ISSN:1390-9134 LAJC 2021

tipo String, sin embargo este tipo de dato cambia, por lo tanto no será posible obtener la información de la ciudad requerida y esto afectará a otras 12 operaciones que procesan la información resultante. El código que debe ser analizado es yelp.js, yelp-bundle,js e index.html. El cambio #3 sobre Combine (20), remoción del campo *businesses* no permitirá el acceso a este dato. Esto significa que debe verificarse el código de yelp.js e index.html. En la Tabla IV, se presentan los resultados de las métricas.

TABLA IV. DETALLE DE MÉTRICAS CALCULADAS POR CAMBIO

Cambio	#1	#2	#3
QOM	20	20	20
QOMP	14	14	14
QI	2	13	1
QID	1	1	1
QII	1	12	0
IT	0,1	0,65	0,05
ID	0,05	0,05	0,05
II	0,05	0,6	0
IP	0,14	0,93	0,07
Código	weather.js index.html	yelp.js,yelpbundle.js, index.html	yelp.js index.html

Ilovemusic

En esta sección, se representa la aplicación de la propuesta al mashup Ilovemusic (http://ilovemusic.mariomelchor.com/) Este mashup permite a los usuarios realizar búsquedas de artistas y acceder a las últimas noticias, lanzamientos y próximos eventos. La aplicación utiliza diversas APIs web como Bandsitown, Wikipedia, Youtube, Spotify y Facebook e Instagram.

En la Figura 6, se representa el GOM del web mashup en estudio, cuenta con 25 operaciones, de las cuales seis son fuentes de información (APIs web), 18 son de procesamiento de información y un (duplicate) realiza una copia de los datos sin modificarlos.

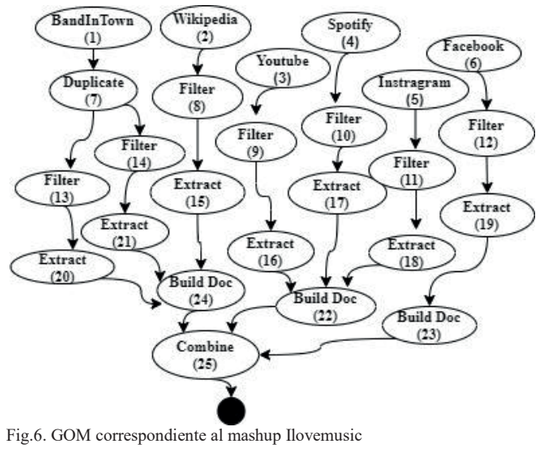

Fig.6. GOM correspondiente al mashup Ilovemusic

Entre las operaciones de procesamiento se identifican filter, este caso está basado en artistas ingresados por el usuario, esta operación se realiza sobre cada una de las fuentes utilizadas. Otra operación es extract que permite seleccionar determinados campos tales como la biografía del artista de la fuente Wikipedia. Luego, se utiliza la operación build doc la cual permite seleccionar campos específicos que forman parte del documento de salida. Finalmente, combine permite unir estas salidas en un documento final. A continuación se describen tres posibles cambios y sus efectos sobre las operaciones de Ilovemusic.

En la Tabla V, se detallan el análisis de 3 cambios; el cambio #1 sobre source BandsInTown (1), modificación de punto final no permitirá acceder a la fuente de información. Lo cual también afectará a las operaciones duplicate (7), filter (13), filter (14), extract (20), extract (21), build doc (24), y combine (25) y por lo tanto a los resultados finales de la consulta. El código a revisar se encuentra en la función searchBandsInTown(). El cambio #2 sobre extract (17), remoción de campo, ya que el campo *artistname* no exite en la respuesta. Lo cual también afectará a las operaciones build doc (22), combine (25) y por lo tanto a los resultados finales. El código a revisar es la función Spotifysearch() e index.html. El cambio # 3 sobre combine (25), modificación en el tipo de dato del campo videoId no permitirá el acceso a este dato para ser presentado en los resultados finales. El código de los archivos youtube.js e index.html deberá ser analizado ante este cambio.

TABLA V. ANALISIS DE CAMBIOS

CAMBIO #1
Tipo: Modificación de Punto Final API: BandsInTown Condición:{Path=https://rest.bandsintown.com/artists/{artistname} ∉ DE}
OPERACIÓN AFECTADA
Tipo: Source Identificador:#1, PREC:{Path=https://rest.bandsintown.com/artists/{artistname} ∈ DE} Código: searchBandsInTown()
CAMBIO #2
Tipo: Remoción de campo API: Spotify Condición: {campo = artistname ∉ DE}
OPERACIÓN AFECTADA
Tipo: Extract Identificador:#17, PREC: {campo = artistname ∈ DE} Código: Spotifysearch(), index.html
CAMBIO #3
Tipo: Modificación de Tipo de Dato API: Youtube Condición: {campo = videoId, tipo = Int ∉ DE}
OPERACIÓN AFECTADA
Tipo: Combine Identificador:#25, PREC: {campo = videoId, tipo = Int ∈ DE} Código: youtube.js index.html

En la Tabla VI, se muestran los resultados de las métricas obtenidos para cada uno de los cambios detallados.

TABLA VI DETALLE DE MÉTRICAS CALCULADAS POR CAMBIO

Cambio	#1	#2	#3
QOM	25	25	25
QOMP	18	18	18
QI	8	3	1
QID	1	1	1

QII	7	2	0
IT	0,32	0,12	0,04
ID	0,04	0,04	0,04
II	0,28	0,08	0,17
IP	0,44	0,16	0,05
Código	searchBandsInTown()	spotifySearch() index.html	youtube.js, index.html

V. Trabajos Relacionados

La evolución del software y el impacto de los cambios es un área de investigación de la ingeniería de software de importancia, diversas líneas se relacionan a este trabajo. En cuanto a los cambios producto de la evolución de las APIs, existen estudios sobre los cambios de las APIs locales (una API sin interacción de red) y el impacto en las aplicaciones que las usan, pero tanto las características de los cambios como las alternativas propuestas para su manejo (refactoring, migración, etc.) no son directamente aplicables al contexto de APIs web o al paradigma de servicios, según [16].

La revisión sistemática [19] analiza 60 estudios sobre el análisis y la propagación del impacto en los dominios BPM (Business Process Analysis) y SOA (Service Oriented Architecture). Según los autores, las soluciones del análisis de impacto basadas en la dependencia dominan claramente, y las técnicas que usan grafos son las adoptadas con más frecuencia, mientras que técnicas que permitan la cuantificación del impacto del cambio a través de ciertas métricas no han recibido tanta atención. Los autores resaltan que predominan los estudios de análisis y propagación del impacto a nivel inter-servicio e intra-servicio, lo que acentúa las diferencias con nuestra propuesta dirigida a mashup de datos. También en el dominio SOA, en [20] analizan que la evolución de los web services genera cambios en la interface de los servicios, cambios en el código del cliente y cambios en la utilidad del servicio, y proponen métricas para estos aspectos individualmente, para ello usan los WSDL (lenguaje descripción de servicios web) para los cálculos.

En lo referente a la coevolución de mashup por cambios en los componentes externos, no existen demasiados estudios. En [21] se presenta un estudio en el cual se identifican las dependencias en un mashup (de unión y de intersección de datos) desde la perspectiva orientada a patrones. A partir de un grafo que representa las operaciones y la definición de probabilidad de influencia, calculan el impacto de un cambio, la propuesta se implementa en torno a la plataforma Mashroom de desarrollo rápido y casual de mashup.

En [22] se presenta un enfoque para la depuración paso-a-paso y el mantenimiento de mashups declarativos, basado en la plataforma WS-Aggregation, lo que facilita la automatización de la propuesta. Esta herramienta es un framework que ofrece plantillas que permiten generar automáticamente las consultas a las APIs web.

Dada la importancia de migrar los programas de los clientes, se ha dedicado investigación a esta área y se han propuesto diferentes enfoques [6][13] para automatizar el proceso de migración. Sin embargo, en la práctica, rara vez se observa que estos enfoques se adopten en la migración de clientes de APIs web. La mayoría de las migraciones aún se realizan manualmente, sin soporte automatizado.

Otros aspectos del mantenimiento de mashups han sido abordados. Los mashups de servicio de datos son una clase especial de aplicación mashup que combina información “on the fly” de varias fuentes de datos. El problema de cómo lograr el mejor rendimiento con el mínimo costo de mantenimiento, cuando se actualizan las fuentes de datos primitivas, y la caché debe actualizarse, es tratado por [23][24] [25]. También sobre el mantenimiento de mashups, en [26] se presentan técnicas que ayudan a los desarrolladores de mashup a mantener las aplicaciones identificando cuándo y cómo cambian las IU de las aplicaciones originales, pero las técnicas que principalmente se utilizan son para buscar los widgets que mejor se adaptan al cambio. Las técnicas están soportadas al entorno de edición MashDesk.

Otra línea que se desprende de la coevolución, refiere a las áreas de adaptación de APIs, o de procesos de negocio, o de la composición de servicios web para cambiar el flujo de trabajo / procesos de negocio [27][28][29]. Nuestra propuesta no aborda este tipo de problemas.

VI. Discusión y Conclusiones

La propuesta presenta resultados preliminares que permiten en esta instancia conocer todas las operaciones que directa y/o indirectamente son afectadas ante determinados cambios ocasionados por la evolución de las APIs web y así asistir a las tareas de revisión y corrección de código.

En el dominio de servicios BPM y SOA, existen diversos trabajos sobre análisis del impacto de los cambios y su propagación [19]. Aquí observamos que difieren los mecanismos de la composición, mientras que BPM y SOA usan esquemas bien estructurados, como WSDL, en los mashup la composición se hace desde las aplicaciones que se ejecutan en el servidor y/o en el cliente, no siendo estructurada. En consecuencia, ocasiona diferencias en la forma de identific

ar, representar y analizar los cambios y su propagación, ya que no se dispone de un mecanismo estructurado para calcular métricas como en [20].

Las principales diferencias de [21] con nuestro trabajo, están en la categorización de las operaciones del mashup, y en que no se consideran los tipos de cambios que pueden generar impactos. Consideramos algunos puntos en común con [22], ya que analizan las dependencias de los datos externos a partir de un grafo, se aplica el concepto de “anomalía” en lugar de cambios y se identifican potenciales puntos de fallos según aserciones, en lugar de operaciones. Sin embargo, no se estima cuantitativamente los efectos de las anomalías.

La decisión de definir operaciones en un sentido abstracto pero con una semántica relacionada a las funcionalidades de los mashups, resulta del objetivo de no limitar el enfoque a herramientas determinadas. Siendo esta, la principal razón por lo que en esta instancia el GOM no se genera automáticamente. Sin embargo, al asociar las operaciones a los sectores de código que la implementan, la ubicación real se mantiene. Esta decisión más allá de ser una limitación, resulta una ventaja, en tanto las estrategias pueden integrarse a herramientas o editores existentes, que cuenten con estas operaciones o similares, como Yahoo Pipes, Pipes Digital y Microsoft Popfly. Se logrará ampliar la funcionalidad de las mismas, en tanto no impone restricciones importantes y pueden generarse automáticamente el GOM y calcularse las métricas. Lo que constituye una diferencia con otros trabajos que hemos relacionado, que son diseñados en función de

ISSN:1390-9266 e-ISSN:1390-9134 LAJC 2021

determinadas herramientas (MashDesk, WS-Aggregation, Mashroom, etc.).

Las métricas que se han propuesto aportan información cuantitativa sobre los efectos del cambio, sin embargo, un análisis del grafo junto con estás métricas ofrecen una visión general del verdadero alcance de un cambio sobre el mashup. Una discusión se plantea entre la clasificación de impacto directo e impacto indirecto. Esta diferenciación surge en tanto aún no es posible precisar que el cambio se propaga *necesariamente*, o bien, considerando que luego de reparada la operación que directamente es afectada, tal vez no sea necesario modificar el resto de las operaciones que le suceden. Las métricas apuntan a considerar operaciones, en lugar de líneas de código, es decir, acompañan el nivel conceptual de abstracción.

La propuesta está centrada particularmente en mashup web que consumen APIs web de datos. Otros tipos de mashup que consumen otros tipos de recursos no han sido considerados, por lo que las conclusiones no los abarcan, aunque se pueda asumir que con adaptaciones y / o extensiones el enfoque es aplicable a dichos modelos.

La investigación relacionada a la evolución de las APIs web y sus cambios es reciente, existen diversas líneas de trabajo en curso. En tanto estas líneas están abiertas y los resultados existentes sean incompletos para formular una teoría o modelo, los aportes para la co-evolución son también incompletos, ya que son dependientes en cierta medida de tipos, características y efectos de la evolución de las APIs. Desde esta perspectiva nuestra propuesta no está cerrada.

El trabajo futuro está orientado a completar el enfoque propuesto, en particular interesa ampliar el tipo de operaciones y de cambios para lograr una más completa asistencia a la co-evolución. Otro objetivo es integrar el enfoque a herramientas de desarrollo de aplicaciones web y refinar el conjunto de métricas aportadas.

Referencias

[1] R. Robbes, M. Lungu and A. Janes. “API fluency” ICSE-NIER '19: Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results. pp 97–100, 2019.

[2] W. Maalej and M. P. Robillard, “Patterns of knowledge in API reference documentation,” IEEE Trans. Softw. Eng., vol. 39, no. 9, pp. 1264–1282, 2013.

[3] U. Dekel and J. D. Herbsleb, “Reading the documentation of invoked API functions in program comprehension” ICPC’09, pp. 168–177, 2009.

[4] R. Trapero Burgos, D. Suarez Fuentes, José Del Alamo and A. Leon Martin. "Next Generation Mashups: How to Create my Own Services in a Convergent World," in IEEE Latin America Transac-tions, vol. 7, no. 3, pp. 390-394, 2009.

[5] R. T. Fielding, “Architectural styles and the design of network-based software architectures,” Ph.D. dissertation, U. of California, Irvine, 2000.

[6] E. Wittern, A. T.T. Ying, Y. Zheng, J. Laredo, J.Dolby, Ch. Young and A. Slominski., "Opportunities in Software Engineering Research for Web API Consumption," 1st International Workshop on API Usage and Evolution (WAPI), Buenos Aires, pp. 7-10, 2017

[7] A. Cummaudo, R. Vasa and J. Grundy, "What should I document? A preliminary systematic mapping study into API documentation knowledge", International Symposium on Empirical Software Engineering and Measurement (ESEM), Brazil pp. 1-6, , 2019

[8] D. Nam, A. Horvath, A. Macvean, B. Myers and B. Vasilescu, "MARBLE: Mining for Boilerplate Code to Identify API Usability Problems," 34th International Conference on Automated Software Engineering, USA, pp. 615-627, 2019

[9] C. Sadowski, K. T. Stolee and S. Elbaum, “How developers search for code: a case study,” in Proceedings of the 10th Joint Meeting on Foundations of Software Engineering. ACM, pp. 191–201, 2015

[10] A.M. Eilertsen and A.H. Bagge. “Exploring API / Client Co-Evolution”, Gothenburg, Sweden 2nd International Workshop on API Usage and Evolution.2018

[11] K.Jezek and J.Dietrich. “API Evolution and Compatibility: A Data Corpus and Tool Evaluation”. Journal of Object Technology, 2:1–23, 2017

[12] L. Xavier, A. Brito, A. Hora and M. T. Valente. “Historical and Impact Analysis of API Breaking Changes: A Large-Scale Study”. IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER). 2017.

[13] C. Prehofer and I. Gerostathopoulos. “Modeling RESTful Web of Things Services: Concepts and Tools”. Managing the Web of Things. Chapter 3, pp. 73-104, ISBN 9780128097649, 2017

[14] R. Koçi, X. Franch, P. Janovic. and A. Abelló. “Classification of Changes in API Evolution”. 23rd International Enterprise Distributed Object Computing Conference (EDOC) 2019

[15] T. Espinha, A. Zaidman, and H.-G. Gross, “Web API growing pains: Stories from client developers and their code,” Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering. IEEE CS, pp. 84–93, 2014

[16] J. Li, Y. Xiong, X. Liu and L. Zhang, “How does web service API evolution affect clients?” 20th Conf. on Web Services. IEEE, pp. 300–307, 2013

[17] T. Espinha, A. Zaidman and H.-G. Gross, “Web API Fragility: How Robust is Your Mobile Application?” Proceedings MOBILESoft, pp. 12–21, 2015

[18] E. Wittern, A. T. T. Ying, Y. Zheng, J. Dolby and J. A. Laredo, "Statically Checking Web API Requests in JavaScript," 39th International Conference on Software Engineering (ICSE), pp. 244-254, 2017

[19] Alam, K.A., Ahmad, R., Akhunzada, A., Nasir, M. and Khan, S. “Impact analysis and change propagation in service-oriented enterprises: A systematic review”. Inf. Syst., 54, 43-73 (2015).

[20] R. Kohar and N. Parimala. “A metrics framework for measuring quality of a web service as it evolves”. International Journal of System Assurance Engineering and Management 8, 1222–1236, 2017

[21] W. Ding, G. Wang, Y. Han and J. Wang. “A Pattern-Oriented Impact Analysis Approach for Mashups”. Fifth IEEE International Symposium on Service Oriented System Engineering 978-0-7695-4081-8/10 , 2010

[22] W. Hummer, P. Leitner and S. Dustdar. “A step-by-step debugging technique to facilitate mashup development and maintenance.” Mashups '09/'10. Cyprus, 2010.

[23] O. Hassan, L. Ramaswarny and J. Miller. “The MACE Approach for Caching Mashups”. International Journal of Web Services Research 7(4), pp 64–88, 2010.

[24] P. Zhang, H. Han and G. Wang. “An Efficient Data Maintenance Strategy for Data Service Mashup Based on Materialized View Selection”. ICSOC , LNCS, vol 7759. Springer, Berlin, Heidelberg, 2012.

[25] G. Wang and F. Zhang, "Freshness-Aware Sensor Mashups Based on Data Services," International Conference on Green Computing and Communications and Internet of Things and IEEE Cyber, Physical and Social Computing, Beijing, pp. 2018-2023, 2013

[26] M. Shevertalov and S. Mancoridis. “On the maintenance of UI-integrated Mashup Applications”. 27th International Conference on Software Maintenance, pp. 203–212, 2011

[27] R. Oberhauser.”A hypermedia-driven approach for adapting processes via adaptation processes”. 8th International Conference on Advanced Software Engineering & Its Applications. pp. 73{80. IEEE (2015)

[28] H.A. Nguyen, T.T. Nguyen, G. Wilson Jr, A.T Nguyen, M. Kim and T.N. Nguyen. “A graph-based approach to api usage adaptation”. ACM Sigplan Notices 45(10), 302-321 (2010)

[29] R. Bustamante and K. Garcés. “Managing Evolution of API-driven IoT Devices through Adaptation Chains”.CIBSE 2020.

AUTHORS

Sandra Casas

Sandra Casas es Dra. en Ingeniería de Software. Es Profesora en la Universidad Nacional de la Patagonia Austral, Argentina, desde el año 1995. Sus líneas de trabajo principales son, técnicas de desarrollo para mejorar la modularización y reutilización de software y calidad de aplicaciones software.

Graciela Vidal

Graciela Vidal es Magister en Informática y Sistemas, egresada de la Universidad Nacional de la Patagonia Austral. Se desempeña como docente e investigadora en el GISP desde el año 2007. Sus trabajos de investigación se orientan a desarrollar estrategias y lineamientos que mejoren el proceso de desarrollo de aplicaciones web que son tendencia en la actualidad.

Franco Herrera

Franco Herrera es Licenciado en Sistemas, egresado de la Universidad Nacional de la Patagonia Austral. Se desempeña como docente e investigador en el GISP desde el año 2005. Sus trabajos de investigación se enfocan a la automatización de desarrollo de software.

S. Casas, G. Vidal, F. Herrera, "A proposal to Assist the Mashup Co-evolution when web APIs evolve", Latin-American Journal of Computing (LAJC), vol. 9, no. 1, 2022.

ISSN:1390-9266 e-ISSN:1390-9134 LAJC 2022