Legacy System Migration Approaches

G. Salvatierra, C. Mateos, M. Crasso, A. Zunino and M. Campo

***Abstract*— Legacy Systems have become a challenge for many enterprises, which have been migrating its legacy systems towards new computational environments because of interoperability issues with modern technologies, high maintenance costs, and hiring expert staff who possess high legacy technologies background. In order to find a strategy to solve legacy systems concerns, several approaches and experiences were created for the last years. This paper presents a strengths and weaknesses analysis of 18 migration approaches.**

***Keywords*— Legacy System Migration, System Replacement, System Modernization, Direct Migration, Indirect Migration, White-Box Modernization, Black-Box Modernization, Semi-Automatic COBOL Migration, Services-Oriented Architecture.**

I. Introduccion

LA EVOLUCIÓN de cualquier producto de software es inevitable. La misma es causada por errores que deben ser reparados, nuevos requerimientos que se suscitan, o nuevas tecnologías que aparecen durante el ciclo de vida del producto [15]. En consecuencia, los sistemas legados de información son candidatos a evolucionar. Por definición, un sistema legado es software en funcionamiento que todavía permanece activo, pero implementado con criterios de diseño y tecnologías desactualizados.

Un ejemplo emblemático de sistema legado son los sistemas monolíticos en *mainframes* [39], que usualmente pertenecen a bancos o entidades gubernamentales. Estos sistemas mantienen décadas de información en un *mainframe* que les aporta disponibilidad, escalabilidad, seguridad y gran capacidad para la realización de consultas, transacciones y almacenamiento de datos utilizando tecnología específica para realizar transacciones como COBOL [17] (De acuerdo a Gartner (http://www.gartner.com), más de 200 billones de líneas de código fuente COBOL operativo están aún corriendo mundialmente.). Actualmente, algunas organizaciones se ven forzadas a evolucionar sus sistemas legados debido a la escasez de personal experto en tecnologías legadas, altos costos para incrementar la potencia de cómputo de los *mainframes*, y problemas de integración con tecnologías Web [14], [25], [23].

La evolución de un sistema implica tres etapas principales [37]: *mantenimiento*, *modernización* y *reemplazo*. Un sistema es creado y luego necesita ser mantenido a fin de reparar inconsistencias y errores menores. Cuando se requieren cambios mayores, el sistema entra en etapa de modernización. Finalmente, cuando el sistema original no alcanza para satisfacer la evolución de los requerimientos, este debe ser reemplazado. De acuerdo a esto y por definición, la migración en sí está comprendida dentro de la etapa de modernización de un sistema [2].

Una estrategia que ha ganado interés en los últimos años es la migración hacia Service-Oriented Architectures (SOAs) [21]. Con SOA, los sistemas son compuestos por funciones reutilizables e independientes entre sí denominadas *servicios*, los cuales son generalmente materializados mediante *Servicios Web*. La migración hacia SOA permite incrementar la reusabilidad de funcionalidades legadas generando interfaces de servicios independientes entre sí, y mejorar la integración con tecnologías modernas [21]. La mayoría de los sistemas legados aún no se han adaptado hacia SOA, como tampoco a arquitecturas Web convencionales. Por esto, existen enfoques de reemplazo que abarcan los aspectos relevantes que implica un proceso de migración, como la ingeniería reversa para el entendimiento de la funcionalidad legada y el relevamiento del modelo relacional de las bases de datos, pero generan un nuevo sistema.

El presente trabajo describe un estudio sobre los enfoques de migración existentes más representativos. Considerando la gran cantidad de trabajos sobre migración de sistemas legados, y a fin de mejorar la comprensión de los mismos, se categorizan los enfoques en dos grupos: Enfoques de Modernización y de Enfoques de Reemplazo [14]. A su vez, la modernización es categorizada en base a dos estrategias generales. Por un lado, la estrategia de encapsular la lógica legada utilizando una capa de software moderna, lo cual se denomina *wrapping*, *black-box* o migración directa. Por otro lado, existen las estrategias *white-box* o migración indirecta, las cuales reimplementan por completo el sistema legado utilizando principios de reingeniería.

En la Sección II se describen enfoques de migración directa e indirecta dentro de la categoría de enfoques de modernización. Seguidamente, en la Sección III se describen los enfoques de reemplazo más relevantes. Finalmente, la Sección IV muestra una comparación de los enfoques mencionados en el presente trabajo, a fin de exponer las conclusiones del estudio en la Sección V.

II. Enfoques de Modernización

Dentro de los enfoques de migración por modernización, se distinguen dos tipos principales: directo e indirecto. El primero es un enfoque *black-box* asociado generalmente al encapsulamiento de funcionalidad legada. El segundo es un enfoque *white-box* asociado la reingeniería de la funcionalidad legada.

G. Salvatierra, Universidad Nacional del Centro de la Provincia de Buenos Aires, Tandil, Buenos Aires, Argentina, gonzafs84@gmail.com
C. Mateos, Universidad Nacional del Centro de la Provincia de Buenos Aires, Tandil, Buenos Aires, Argentina, cmateos@conicet.gov.ar
M. Crasso, Universidad Nacional del Centro de la Provincia de Buenos Aires, Tandil, Buenos Aires, Argentina, mcrasso@conicet.gov.ar
A. Zunino, Universidad Nacional del Centro de la Provincia de Buenos Aires, Tandil, Buenos Aires, Argentina, azunino@conicet.gov.ar
M. Campo, Universidad Nacional del Centro de la Provincia de Buenos Aires, Tandil, Buenos Aires, Argentina, mcampo@exa.unicen.edu.ar

A. Migración Directa

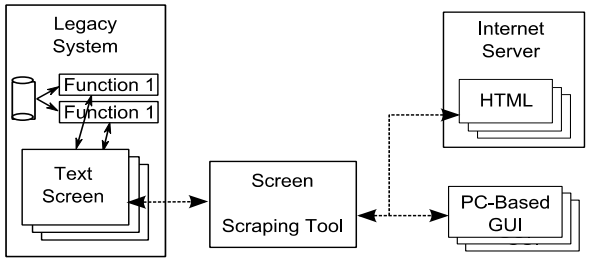

Figura 1. Screen Scraping.

El enfoque *Screen Scraping* (Fig. 1) consiste en la modernización de la User Interface (UI) de un sistema legado, para mejorar la usabilidad [14]. Generalmente, las interfaces gráficas de los sistemas legados consisten en un pantallas basada en texto. En contraste, la nueva interfaz puede ser gráfica o incluso basada en tecnologías Web. Este enfoque puede ser aplicado fácilmente, proveyendo al sistema legado con una nueva UI. La nueva interfaz de usuario interactúa con el sistema legado utilizando un *middleware* (*Screen Scraping Tool*) que se encarga de la interoperación. De la misma manera, la nueva interfaz envía las acciones del usuario al *middleware*, el cual se encarga de mapear cada acción al módulo legado correspondiente para la ejecución de la lógica legada apropiada. Por lo tanto, una sola interfaz de usuario puede comunicarse con varios sistemas.

En [32], [31] se presenta un método soportado por una herramienta para el mantenimiento de código legado dentro de un ambiente SOA. El código legado es encapsulado detrás de una capa SOA, la cual permite que funciones individuales en los programas sean ofrecidas como Servicios Web. Los módulos de código que representan una modificación de datos o bien servicios son identificados utilizando herramientas de *clustering*, la cual es una técnica que permite agrupar automáticamente individuos con categorías similares. Luego, en base a esta técnica, los flujos de datos son extraídos para construir un nuevo componente. A este nuevo componente se le asigna una interfaz Web Services Definition Language (WSDL) y un *framework* Simple Object Access Protocol (SOAP) es utilizado para empaquetar dicho componente. Finalmente, un *proxy (*Un *proxy*, en una red informática, es un programa o dispositivo que realiza una acción en representación de otro.) es generado para conectar los nuevos servicios dentro de la capa SOA. La viabilidad de la técnica ha sido mostrada en base a una función de calendario COBOL extraída del software legado de un banco suizo, y el enfoque ha sido aplicado integrando SOA con programas COBOL y C++. Según el autor, el enfoque es más adecuado para programas pequeños ya que la identificación y exposición de las funciones de negocios puede demandar mucho tiempo.

En [7], los autores proponen un método *black-box* basado en encapsulamiento para lograr que las funcionalidades interactivas de sistemas legados sean accesibles como Servicios Web. El *wrapper (*En programación, un *wrapper* es un objeto utilizado para encapsular cierto comportamiento u otros objetos.) que "envuelve" a los módulos del sistema legado interpreta una Máquina de Estados Finitos, la cual describe el modelo de las interacciones entre el cliente de un servicio y el sistema legado. Para probar su enfoque, los autores generaron un *wrapper* para Pine, una aplicación para consultar una casilla de correo electrónico, la cual provee una interfaz SOA para obtener la funcionalidad de cada correo. En su trabajo más reciente [8], esta técnica de encapsulamiento es utilizada como parte de una migración completa cuyo proceso consiste de la selección de los servicios deseados, encapsulamiento de los casos de uso seleccionados, y desarrollo y validación de éstos. La principal desventaja de este método es que requiere mucho trabajo manual. Si bien el modelo utilizado para representar las interacciones entre cliente y servicio se obtiene mediante técnicas de ingeniería inversa, el enfoque se centra en la generación manual de *wrappers* concibiendo la lógica como una Máquina de Estados Finitos.

En [34], [35], los autores proponen un proceso global de migración de módulos legados utilizando el método CelLEST. Con dicho método, las interacciones de los usuarios con el sistema legado son sometidas a un proceso de ingeniería inversa y los segmentos de tareas específicas de estas interacciones son encapsulados en nuevos *front-ends* accesibles desde la Web mediante la generación de una GUI. De esta manera, se obtiene como resultado una interfaz de usuario abstracta especificada en XHTML. El método consiste de tres etapas: (1) recolectar rastros de interacciones entre el sistema y el usuario utilizando un *middleware* no intrusivo, (2) aplicar ingeniería inversa al comportamiento dinámico de la interfaz del sistema en términos de las pantallas que presenta para el usuario y la navegación del usuario a través de las mismas, y (3) analizar las rutas de navegación de tareas específicas para extraer un modelo de las tareas de interés para el usuario, en términos de navegación de la interfaz y el intercambio de información implicado. CelLEST ha sido probado sobre infoMcGill, una aplicación legada de la Universidad McGill. El caso de estudio fue realizado por una sola persona, la cual estaba familiarizada con el sistema legado. Sin embargo, según los autores, CelLEST no requiere que se conozca el código legado. El modelado de las tareas realizadas por los usuarios sobre la aplicación legada se encuentra basado en el seguimiento de sus interacciones.

En [33], se propone un método para identificar secciones de código legado, encapsulando la navegación y ejecución de un sistema legado, para ser invocados desde una aplicación Web, la cual es accesible mediante un navegador estándar. El método permite identificar bloques individuales de código legado que son encapsulados y reutilizados como Servicios Web utilizando un proceso de siete pasos: Minería de Funciones, Encapsulamiento de Funciones, Creación de un XML Schema Definition (XSD), Generación de un Servidor *Stub (*Un *stub* en computación distribuida es una pieza de código usada para convertir parámetros durante una llamada a procedimiento remoto.), Generación de una Clase Cliente, Conexión de Servidores, y Enlace de Servicios Web. En este orden, los bloques individuales de código legado son extraídos en base a la información provista por una herramienta,

denominada SoftWrap. Sin embargo, la desventaja de este método es que requiere un estudio manual por parte de expertos previo a la detección y selección de los bloques de funcionalidad legada.

En [30], los autores describen un conjunto de herramientas para construir Servicios Web desde programas COBOL que se ejecutan sobre un *mainframe*. La primera herramienta –*COBAudit*– está destinada a identificar Servicios Web candidatos. La segunda herramienta –*COBStrip*– sirve para extraer solamente las porciones de código requerida para realizar los servicios. La tercera herramienta –*COBWrap*– encapsula el código extraído y lo convierte en un componente ejecutable. La cuarta herramienta –*COBLink*– conecta el componente encapsulado a la Web generando una interfaz WSDL. Las herramientas han sido evaluadas sobre un sistema de 25 millones de líneas de código COBOL.

Los autores han iniciado las investigaciones con un proyecto piloto. Los objetivos de estas investigaciones fueron (1) medir los tamaños, complejidad y calidad de los programas COBOL candidatos con el fin de evaluar la magnitud del trabajo a realizar, (2) evaluar la posibilidad de reutilizar los componentes candidatos como Servicios Web, (3) determinar la posibilidad de extraer piezas de código desde los programas COBOL existentes para reutilizarlos como Servicios Web, (4) demostrar que los programas seleccionados pueden ser encapsulados para trabajar independientemente de la tecnología original, y (5) probar si los componentes encapsulados pueden ser accedidos como Servicios Web.

Para el primer objetivo, los autores procesaron 7297 programas COBOL con la herramienta COBAudit. Para las mediciones tomaron en cuenta las líneas de código fuente, el número de sentencias, de estructuras de datos, archivos y accesos a base de datos, el número de decisiones, el número de sub-rutinas y el número de invocaciones a las mismas, entre otras. La complejidad de los programas ha sido medida con un coeficiente sobre una escala de 1 a 0, con 0.5 como complejidad media, entre 0 y 0.4 indica complejidad baja, entre 0.4 a 0.6 indica complejidad media, entre 0.6 a 0.8 indica complejidad alta y superior a 0.8 indica que el código no es reutilizable. Mediante modularidad, reusabilidad, flexibilidad, portabilidad, convertibilidad, testeabilidad, conformidad y mantenimiento, se midió calidad. Una vez que el código ha sido medido, se determinó si era factible reutilizar ciertos programas como Servicios Web, medido como la percepción de qué porciones de los programas podrían ser encapsuladas en servicios sin requerir demasiado esfuerzo. Para probar la reusabilidad de los programas, los autores tomaron varios programas de ejemplo ejecutándolos sobre la herramienta de encapsulamiento automática.

Luego, el enfoque aplica la técnica de *Code Stripping* que selecciona caminos de ejecución en un programa mezclando todos los datos y sentencias no utilizados. En el contexto de encapsulamiento, la técnica es utilizada para generar múltiples instancias del mismo programa. Cada instancia puede ser un servicio Web independiente. Una instancia particular corresponde a una regla de negocios particular. Anteriormente, los programas eran escritos para cumplir con varias reglas de negocios a la vez, por razones de eficiencia. Esto llevó a la interrelación de las funciones de negocios entre sí, lo que constituye el principal problema para reutilizar programas como servicios debido a que un servicio debería solamente corresponder a una única regla de negocios.

Con fines experimentales, los autores seleccionaron ejemplos de programas complejos y los ejecutaron sobre la herramienta COBStrip, la cual requiere de interacción humana. Dado un programa a migrar, la herramienta muestra al usuario una vista de los datos de salida definidos en el área de definición de datos del programa. El usuario debe marcar cuáles datos de salida desea de los programas. Luego, la herramienta localiza todos los párrafos COBOL requeridos para producir los datos seleccionados y el resto del código COBOL es comentado. El resultado consiste de procedimientos incluyendo los bloques de código seleccionados y un área de definición de datos con las estructuras utilizadas desde dicho código.

El siguiente paso del proceso de automatización propuesto consiste en encapsular el código seleccionado en una interfaz WSDL utilizando COBWrap y COBLink. COBWrap permite reemplazar las invocaciones a operaciones de terminal con llamados a módulos encapsulados. Luego, los módulos encapsulados son conectados a Internet mediante un componente intermedio (*middleware*) utilizando la herramienta COBLink. Esta herramienta genera dos módulos *wrapper* que serán conectados a los programas *wrapper*. La entrada a la herramienta es el código COBOL y en base a la declaración de los parámetros de entrada, COBLink crea un esquema WSDL para la solicitud de invocación al servicio y genera un módulo COBOL trasladando dicha solicitud a los parámetros de entrada del programa servidor. En una segunda ejecución, la herramienta genera otro esquema WSDL para la respuesta del servicio y a la vez, genera otro módulo COBOL para transferir los parámetros de salida del programa servidor hacía la respuesta WSDL.

Los autores probaron las herramientas sobre 7297 programas COBOL, de los cuales 5 fueron seleccionados para ser extraídos. La herramienta COBStrip mostraba el área de definición de datos de estos programas al especialista a cargo para la definición de variables en los Servicios Web seleccionados. Luego, el código ha sido extraído conteniendo sólo los párrafos requeridos para producir los resultados esperados de cada servicio. En un segundo paso, COBWrap ha reemplazado las invocaciones a módulos legados por invocaciones a los nuevos programas *wrapper*. Seguidamente, COBLink ha generado los Servicios Web a partir del código fuente de los programas encapsulados, definiendo explícitamente las estructuras de intercambio de datos para la operación asociada a nivel servicios.

Por otra parte, en [1], se presenta un *framework* y guías para la identificación de servicios para la migración de sistemas legados hacia SOA. El enfoque y herramienta propuestos se centran en la identificación de servicios en términos de granularidad, reusabilidad, mantenimiento, rendimiento y flexibilidad. Los autores han adoptado la Business Process Modeling Notation (BPMN) (BPMN es una

notación gráfica estandarizada que permite el modelado de procesos de negocio, en un formato de *workflow*.) para proporcionar un modelo de comportamiento del sistema legado representando gráficamente las interacciones y colaboraciones entre participantes. Además, los autores proponen un método de transformación entre BPMN y UML facilita la representación de las funcionalidades legadas a fin de identificar servicios. El proceso de identificación de servicios propuesto consta de tres etapas:

- Etapa de Análisis y Re-ingeniería: Agrupa tres modelos, a) análisis de modelos para comprensión de alto nivel del sistema, b) generación de un diagrama de actividades representando la estructura del sistema, y c) generación de un diagrama de procesos que describe el comportamiento del sistema.
- Etapa de Identificación de Servicios: Determina la granularidad óptima de servicios. El resultado de esta etapa es un conjunto de servicios identificados mediante métricas diseñadas previamente.
- Etapa de Evaluación de Servicios: La clasificación y categorización de tipos de servicios establece una base para definir la granularidad de un servicio. Elementos tales como el WSDL de un servicio o su implementación pueden contribuir a la clasificación y definición. Esta etapa genera un meta-modelo que proporciona una comprensión de los procesos de negocios y los servicios generados. Para la clasificación, los autores han definido un conjunto de reglas basadas en la lógica negocios encapsulada por las operaciones de servicios.

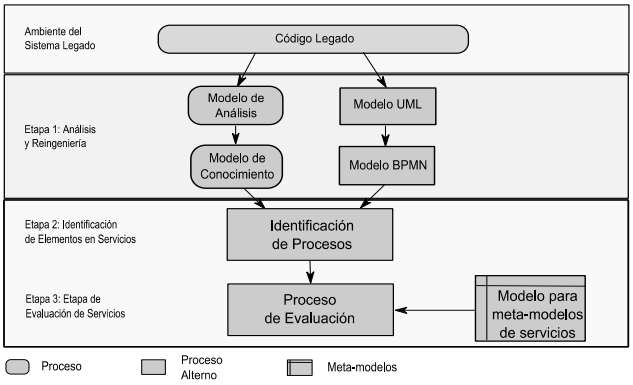

Figura 2. Framework para identificación de servicios.

Los autores validaron su enfoque sobre una aplicación piloto, con código legado, para comercio electrónico. El *framework* propuesto se ilustra en la Fig. 2, indicando la entrada a la herramienta y los pasos involucrados en cada una de las tres etapas del proceso.

En [28] se propone un enfoque de migración asistida de sistemas legados hacia SOA. El objetivo del enfoque es asistir a los desarrolladores durante la migración a SOA de un sistema legado, en vez de intentar automatizar un proceso de migración completo. Para cumplir con este objetivo el enfoque de migración asistida consta de tres etapas principales, como se observa en la Fig. 3.

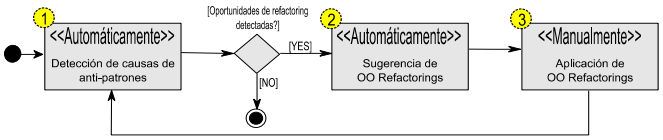

Figura 3. Etapas del enfoque de migración asistida.

La etapa de "Detección de causas de anti-patrones" se realiza automáticamente empleando una herramienta que recibe como entrada un conjunto de módulos legados, y genera por cada módulo legado un conjunto de malas prácticas sobre el código como nombres ambiguos, duplicación, etc., que de no resolverse generan malas especificaciones WSDL. Así, se obtiene un conjunto de posibles problemas a solucionar durante la migración hacia el nuevo sistema SOA. La etapa de "Sugerencia de OO *refactorings*" también es llevada a cabo automáticamente por la herramienta de la primera etapa y consiste en generar un conjunto de *refactorings* a forma de sugerencia, que pueden ser aplicados para evitar las causas de anti-patrones detectadas previamente. Estas sugerencias de *refactoring* vienen acompañadas de un conjunto de guías de *refactoring*, las cuales facilitan la aplicación de los *refactorings* Orientados a Objetos en ambientes SOA. La etapa de "Aplicación de *refactorings*" consiste en la aplicación manual de las sugerencias de la etapa 2, utilizando como base las guías de *refactoring* propuestas por el enfoque de migración asistida. El resultado de esta etapa es un conjunto de interfaces de Servicios Web que respetan los estándares de interoperabilidad en Servicios Web –Web Services Interoperability (WS-I) (Basic Profile Version 1.1: http://www.ws-i.org/Profiles/BasicProfile-1.1.html)–.

Para comprender completamente las etapas de este enfoque es necesario exponer varios aspectos importantes. Primero, el enfoque de migración asistida consiste de un proceso iterativo. Es decir, el resultado de una iteración puede ser utilizado como entrada de la siguiente iteración. Segundo, el proceso de detección de causas de anti-patrones aprovecha el estudio realizado en [27], en el cual los autores confeccionaron un catálogo de malas prácticas que atentan contra el diseño de documentos WSDL y dificultan el descubrimiento de los Servicios Web. Por este motivo, los autores del catálogo denominaron *anti-patrones de descubrimiento* a estas malas prácticas. Así, el enfoque de migración asistida plantea un conjunto de heurísticas destinadas a la detección de anti-patrones de descubrimiento "latentes" en sistemas legados migrados a SOA mediante *wrapping*.

La principal ventaja del enfoque es la generación de interfaces en WSDL de calidad similar a la obtenida en una migración indirecta, pero disminuyendo los costos de migración y generando resultados en menor tiempo tal y como sucede en un enfoque de migración directa. Esto resulta beneficioso para organizaciones que prefieren resultados a corto plazo y con menor costo de migración sin descuidar la calidad del sistema resultante. Adicionalmente, las interfaces de Servicios Web generadas por la migración asistida pueden ser reutilizadas en un enfoque de migración indirecta, debido a que utilizando SOA las interfaces o descripciones de los servicios son independientes de su implementación. La

principal desventaja del enfoque es sin embargo que el nuevo sistema es dependiente del sistema legado original. Adicionalmente, si bien el enfoque no requiere comprensión del sistema legado por parte de los desarrolladores, la migración asistida implica que el desarrollo de los documentos WSDL debe ser realizado manualmente.

B. Migración Indirecta

Esta sección describe aquellos trabajos relacionados con enfoques de migración indirecta que abordan conceptos de re-ingeniería o re-implementación completa. En [12], los autores describen un proyecto en el cual una herramienta legada de derivación y prueba de teoremas –denominada Bertie3– es sometida a reingeniería y modernización hacia SOA, resultando en una nueva herramienta –llamada Service-Oriented Bertie (SoBertie)–. Para lograr este objetivo, el núcleo de Bertie3 es almacenado en un servidor y expuesto como servicio Web. En un trabajo más reciente [10], los autores presentan una metodología de re-ingeniería de software orientada a servicios (SoSR) diseñada para la modernización de sistemas legados hacia SOA. La metodología SoSR, como una síntesis de buenas prácticas, es centrada en la arquitectura, orientada a servicios, orientada a roles específicos, y manejada por modelos. Esto ha sido conceptualizado por los autores desde un modelo de tres participantes de servicios, utilizando una vista 4+1 (Vista para describir la arquitectura de sistemas de *software* basada en el uso de múltiples vistas complementarias.) y gráficos de asignación de responsabilidades. El enfoque se evaluó mediante un caso de estudio de un departamento de compras de un sistema de inventarios de ventas. El análisis de los resultados determina que esta metodología puede ser utilizada por los desarrolladores de software para modernizar sistemas legados de información altamente acoplados, generando nuevos sistemas desacoplados, ágiles y orientados a servicios. Sin embargo, el enfoque puede resultar complejo a raíz de la utilización de diversas vistas. Adicionalmente, el desarrollador debe tener conocimientos en arquitectura de sistemas ya que este enfoque combina diferentes tipos de arquitecturas: *3-tier* para diseño del modelo de negocios, *n-tier* para la etapa de *deployment* de servicios, y SOA para la integración con el sistema legado.

En [18], los autores presentan un enfoque holístico para re-diseñar aplicaciones legadas para la Web, utilizando un *framework* denominado Ubiquitous Web Applications Design Framework (UWA) y una versión extendida del mismo denominada UWA Transaction Design Model (UWAT+). La idea consiste en diseñar tecnologías de recuperación de información legada y métodos de diseño avanzados para sistemas orientados a la Web. El proceso utilizado por los autores para producir el diseño conceptual UWA/UWAT+ de la nueva aplicación consiste de tres etapas: elicitación de requerimientos (identificación de interesados, sus necesidades y los requerimientos de la migración del sistema legado), ingeniería inversa (abstracción de información del sistema legado, representando la misma en base a modelos UWA para obtener el diseño preliminar del nuevo sistema), e ingeniería directa (utilización de los requerimientos definidos en el primer paso para refinar el diseño generado en el segundo paso).

El enfoque ha sido utilizado en la re-ingeniería de un sistema legado denominado GPA, el cual ha sido desarrollado en 1990 para soportar procesos de contratación del sector público. El resultado de la modernización ha sido la generación de una aplicación Web destinada al usuario final, preservando las restricciones de negocios del sistema original. La principal desventaja del enfoque es que no se ha realizado una amplia investigación sobre tecnologías aplicables a los tres pasos del enfoque, lo cual indica la necesidad de contratar personal experto para la aplicación efectiva de dicho enfoque. Otro problema es que el enfoque consume mucho tiempo para alcanzar la modernización según el proceso indicado.

En [9] se propone un enfoque basado en análisis de características de las funcionalidades del sistema legado para soportar una re-ingeniería orientada a servicios. El análisis de características se basa en la identificación de las funcionalidades del sistema, construcción de un modelo global de características para organizar los rasgos detectados, e identificación de la implementación en el sistema legado a través de técnicas de localización de características. Algunas de las características que se utilizan en el enfoque son la similitud semántica entre funciones legadas y la granularidad de datos. Desde un punto de vista general, el enfoque se basa en la recopilación de los rasgos de bajo nivel asociados a la funcionalidades de un sistema legado con el propósito de evitar malas prácticas en la generación de los Servicios Web resultantes y para facilitar la identificación de servicios candidatos.

Los autores utilizaron un sistema de gestión de información–MIS por sus siglas en Inglés– en un campus digital para validar el enfoque. El MIS es analizado utilizando una técnica *top-down* de descomposición de dominio y análisis de características. Varios servicios han sido identificados a lo largo del sistema junto a sus características. Luego, la implementación de los servicios identificados ha sido generada mediante una herramienta de encapsulamiento de Servicios Web. La herramienta es capaz de generar el *glue code (*En programación computacional, código que sirve para conectar dos partes código que no pueden estar unidas directamente.) para los Servicios Web y código fuente asociado a los métodos de los mismos. Sin embargo, la complejidad del enfoque puede verse afectada en relación a sistemas que manejan grandes volúmenes de datos.

En [15], los autores proponen un proceso para la recuperación de la arquitectura de un sistema legado con el fin de identificar el plan que debe ser llevado a cabo en la modernización de tal sistema. Los autores utilizan un enfoque de migración indirecta en base a la modificación del código legado existente. El enfoque es representado mediante un proceso de tres pasos que consta de la recuperación de la arquitectura legada, creación de un plan de evolución, y ejecución de dicho plan. La recuperación de la arquitectura soporta la creación de documentación apropiada para la misma. El plan de evolución consiste de cuatro fases:

selección de la arquitectura, definición de ciclos de la evolución, planeamiento de los ciclos y comprobación de factibilidad preliminar. Finalmente, el proceso es completado mediante la ejecución del plan definido previamente.

En cuanto a los aspecto técnico del enfoque, el proceso de recuperación de la arquitectura legada se basa en QAR (QUE-es Architecture Recovery). QAR es un *workflow* de recuperación de arquitectura que utiliza tres tareas principales: análisis de documentación, análisis estático y análisis dinámico. Luego, la definición del plan de evolución inicia con la selección de la arquitectura. Para esta tarea se utiliza la tecnología Open Services Gateway Initiative (OSGi) como base para la nueva arquitectura. Seguidamente, los pasos del plan de evolución se guían mediante el *workflow* QAR para definir la migración del sistema original hacia el nuevo modelo OSGi planteado previamente. Finalmente, la evolución es llevada a cabo generando un conjunto de servicios. Los autores probaron su enfoque a partir de un caso de estudio centrado en la evolución de un sistema de imágenes médicas. Según los autores, los resultados son aceptables pero el enfoque sólo es aplicable en sistemas pequeños.

En [17] se intenta automatizar la etapa de recolección de información del sistema legado, que luego puede ser utilizada como base para diferentes enfoques de migración, incluyendo migración a SOA, ya que esta etapa es independiente de la tecnología final utilizada y es una parte vital del proceso de modernización de un sistema. Los autores buscan descubrir el costo estimado de las etapas de un proyecto de renovación de un sistema, prestando principal atención a la etapa de recolección de información sobre este tipo de sistemas. Para lograr tal objetivo, los autores decidieron analizar dos grandes sistemas COBOL legados del área bancaria. El enfoque propuesto analiza el código legado utilizando herramientas genéricas que no poseen conocimiento previo del sistema analizado. Los resultados son almacenados en un repositorio central, que puede ser consultado, impreso y visualizado. El proceso de almacenado es iterativo, incorporando en cada iteración un grado de profundidad mayor en el conocimiento del sistema. Esta iteración continúa hasta que el ingeniero encargado del análisis del sistema legado gana suficiente conocimiento del mismo.

La herramienta construye varios grafos de dependencias a varios niveles de abstracción que ayudan a determinar si el sistema contiene código reutilizable, las entidades de negocios involucradas, las sentencias que describen reglas de negocios, y qué porciones de código son dependientes de la plataforma. El objetivo principal es identificar módulos y estructuras de datos legados reutilizables durante un proceso de migración.

C. Combinación de enfoques directo e indirecto

En la presente sección se resumen aquellos enfoques híbridos que utilizan, de diversas maneras, conceptos de encapsulamiento o *wrapping* (asociados a migración directa) y conceptos de re-ingeniería o re-implementación (asociados a migración indirecta).

En [24], los autores presentan una estrategia que identifica y utiliza componentes legados como servicios. Una reconstrucción arquitectónica es utilizada para identificar dependencias entre componentes para la migración a servicios y, por lo tanto, proporcionar una forma de organización con una mejor comprensión para los procesos de toma de decisiones. La herramienta utilizada para llevar a cabo esta tarea es denominada Architecture Reconstruction and MINing (ARMIN). ARMIN utiliza la información que es extraída desde el código fuente legado en forma de RSF (*Rigi Standard Format*: http://www.rigi.csc.uvic.ca/downloads/rigi/doc/node52.html). Luego, se realiza la identificación de servicios en base a las dependencias entre componentes legados. Estas dependencias incluyen dependencias funcionales donde un componente utiliza funcionalidades de otros componentes del sistema, y dependencias de datos donde los datos globales son compartidos por varios componentes del sistema. Utilizando esta información, ARMIN genera una vista de componentes en base a todas las dependencias entre componentes del sistema.

Los autores han utilizado un caso de estudio de un sistema Command and Control (C2), donde el objetivo era realizar una evaluación preliminar de la posibilidad de convertir un conjunto de componentes del sistema hacia SOA. El sistema original se ejecutaba sobre un sistema operativo Windows y se encontraba escrito en C++, constando de 800,000 líneas de código y 2500 clases. En este sistema se han identificado 7 servicios potenciales, los cuales constaban de 29 clases en total. Según los resultados obtenidos, los autores reportaron problemas en el enfoque debido a inconsistencias causadas por diferencias de implementación y documentación en el código fuente, indicando la necesidad de familiarizarse previamente con dicho código.

En [21], [20] y [29] se discute una técnica de migración llamada Service-Oriented and Reuse Technique (SMART) que ayuda a las organizaciones a analizar sistemas legados para decidir si sus funcionalidades pueden ser razonablemente expuestas como servicios en una SOA. Esta técnica ha sido diseñada con el objetivo de modernizar sistemas militares mediante *wrapping*, pero se ha evolucionado considerando aquellos casos donde el encapsulamiento no es viable. SMART considera las interacciones específicas que serán requeridas para alcanzar una SOA y considera cualquier modificación que deba ser realizada sobre los componentes legados. Esto implica reunir una amplia gama de información acerca de los componentes legados, el sistema objetivo y los potenciales servicios para producir una estrategia de migración de un servicio como su artefacto primario.

SMART requiere de información inicial obtenida mediante reuniones entre el grupo encargado de la migración y los encargados del sistema. De esta manera, se definen los interesados de la migración, se identifica el sistema SOA objetivo y se realiza una descripción de alto nivel de la arquitectura del sistema legado. Estas tareas se encuentran dirigidas por una Migration Interview Guide (SMIG). La SMIG contiene preguntas directamente asociadas con las arquitecturas (arquitectura original y arquitectura final), diseño, código fuente y esfuerzos de migración a servicios. Utilizando SMIG se puede medir el costo, esfuerzo y riesgos

de la migración a SOA de un sistema legado específico. Mediante la SMIG se definen tres actividades que marcan el inicio del enfoque: establecer interesados y su contexto en relación al sistema, describir la funcionalidad existente y describir el sistema SOA a alcanzar.

Una cuarta actividad surge luego de la etapa de relevamiento: analizar la "distancia" entre el sistema actual y el sistema objetivo. Este análisis puede sugerir *trade-offs* entre la nueva arquitectura y la original. Esta actividad consta de la identificación de componentes reutilizables, identificación de información adicional, y análisis de las relaciones componentes/servicios. Finalmente, la última actividad consiste en el desarrollo de una estrategia de migración. Las estrategias pueden variar de acuerdo al dominio. Por ejemplo, en algunos casos se utiliza una estrategia de encapsulamiento *quick-and-dirty* o encapsulando mediante una herramienta automática, seguido por re-estructuración en capas de la aplicación y modificaciones para utilizar otros servicios.

Los autores evaluaron su enfoque sobre el sistema C2 mencionado también en [24]. SMART permitió identificar 7 servicios de 16 clases en total (16,163 líneas de código) con niveles bajos de dificultad de migración y riesgos. El enfoque SMART resulta flexible en términos de combinación de estrategias (por ejemplo, *quick-and-dirty* y re-estructuración). Sin embargo, el enfoque no considera ningún aspecto técnico asociado a la migración de un sistema legado hacia SOA.

III. Enfoques de Reemplazo

Esta sección resume los enfoques más relevantes que aplican conceptos de reingeniería a fin de reemplazar un sistema legado por completo, lo cual implica reescribir el código fuente, diseñar una nueva arquitectura, plantear el uso de nuevas tecnologías y definir un nuevo esquema de bases de datos. En este sentido, tres son los enfoques generales más conocidos: *Cold Turkey*, *Chicken Little* y *Butterfly*. La particularidad de estos tres enfoques es que los últimos dos surgen, cronológicamente, como solución a los problemas presentados por el primero.

El enfoque *Cold Turkey(*El término *Cold Turkey* se utiliza en medicina para describir el cese abrupto a una adicción.) [5], [6], representa la base de los enfoques de reemplazo de sistemas legados [14] y consiste en reemplazar un sistema legado generando un nuevo sistema completamente desde cero, utilizando una arquitectura moderna corriendo sobre una nueva plataforma. Principalmente, este enfoque implica mantener dos sistemas hasta que el nuevo sistema está listo para reemplazar al sistema legado. Esto implica la necesidad de contratar recursos para mantener ambos sistemas simultáneamente durante el tiempo que dure la migración.

Este enfoque implica un alto riesgo de fracaso. Por lo general, no sólo se utiliza para disminuir los costos de mantenimiento, sino también para mejorar la calidad del sistema original distribuyendo correctamente la funcionalidad legada y agregando nueva funcionalidad al nuevo sistema. La adición de la nueva funcionalidad incrementa el riesgo de fracaso debido al incremento de la complejidad en la migración. En general, una migración completa del código de un sistema legado puede tardar años en completarse. Durante ese tiempo de migración, el sistema original se mantiene en funcionamiento y las reglas de negocios continúan su evolución acorde a los nuevos requerimientos. Esta situación desencadena en posibles inconsistencias entre el sistema original y el nuevo sistema.

Con *Cold Turkey*, el sistema original continúa en funcionamiento. Por lo tanto, se dificulta migrar los datos actualizados de una base de datos legada, debido principalmente a la gran cantidad de datos contenidos en un sistema legado. Adicionalmente, debe existir una estrategia mediante la cual se adapte la información legada a los requerimientos de la nueva base de datos. Esto adhiere complejidad a *Cold Turkey* y puede ser motivo suficiente para evitar este enfoque.

En resumen, con este enfoque es necesario garantizar que el nuevo sistema contendrá, además de la funcionalidad provista por el sistema legado, nuevas características adicionales. Estas nuevas características incrementan la complejidad de la migración e incrementan el riesgo de fracaso porque generalmente estos sistemas suelen estar conectados a otros sistemas o tecnologías legadas. Por lo tanto, la re-implementación de un sistema legado puede indirectamente afectar a otros sistemas.

Según Brodie & Stonebraker [6], la metodología *Cold Turkey* ha fallado sobre varios casos de prueba con sistemas legados reales. Debido a esto, los mismos autores han generando una nueva metodología opuesta a *Cold Turkey*, denominada *Chicken Little*, la cual consta de 11 pasos iterativos: Análisis de la lógica del sistema legado, Descomposición el sistema legado en módulos, Diseño de las interfaces del nuevo sistema, Diseño de las aplicaciones del nuevo sistema, Diseño de la nueva base de datos, Instalación del nuevo ambiente, Creación e instalación de las nuevas puertas de enlace, Migración de la base de datos legada, Migración de las aplicaciones legadas, Migración de las interfaces legadas, y Finalización de la migración generando el nuevo sistema de información.

Con *Chicken Little*, el sistema legado y el nuevo sistema operan en paralelo y la funcionalidad legada es migrada gradualmente hacia el nuevo sistema. De esta manera, si un paso iterativo falla sólo es necesario repetir el paso fallido sin necesidad de repetir el proceso completo. Ambos sistemas utilizan una puerta de enlace en común (*Gateway*) hacia la información, logrando de esta manera la interoperabilidad necesaria durante el proceso de migración. Las puertas de enlace son la clave del enfoque *Chicken Little* y cumplen dos propósitos: funcionar como intermediario entre los componentes legados y los componentes del nuevo sistema; y ocultar al usuario los detalles del núcleo del sistema sin necesidad de conocer qué sistema (legado o nuevo) responde a cada funcionalidad requerida. Para cumplir con ambos propósitos, los autores definen 3 tipos de puertas de enlace: *Database Gateway*, *Application Gateway*, *Interface Gateway*. Los primeros dos tipos intermedian entre los componentes internos para conectar ambos sistemas, mientras que el tercero intermedia entre el usuario y el sistema.

Utilizando *Chicken Little*, la información es almacenada en ambos sistemas. En la mayoría de los casos se utiliza un *Middleware* entre la comunicación para mantener la consistencia de los datos. Sin embargo, los autores afirman que "Mantener consistencia entre sistemas de información heterogéneos es un problema técnico de alta complejidad el cual no posee una solución general hasta el momento y es un desafío a resolver." [6].

Finalmente, *Chicken Little* no provee guías sobre el *testing* aplicado durante el proceso de migración ya que, como se ha mencionado previamente, ambos sistemas operan paralelamente. Esto representa una de las desventajas del enfoque.

Por otra parte, el tercer enfoque alternativo, *Butterfly*, forma parte de un proyecto denominado MILESTONE [38]. El mismo surge como posible respuesta a las problemáticas presentadas por los enfoques *Cold Turkey* y *Chicken Little*. La metodología *Butterfly* divide el proceso de migración en 6 etapas principales: Preparación de la migración, Entendimiento de la funcionalidad del sistema legado y desarrollo de un esquema de base de datos tentativo, Elaboración de un repositorio de prueba (*Sample Data Store*) basado en el esquema previo, Migración de todos los componentes (excepto los datos) del sistema legado al nuevo sistema, Migración de los datos legados gradualmente hacia el nuevo sistema y entrenar a los desarrolladores en las nuevas tecnologías, y Pasaje del nuevo sistema al ambiente de ejecución.

Antes de describir cada etapa es necesario describir dos puntos importantes. Primero, la metodología *Butterfly* almacena información acerca del sistema legado durante la migración. Adicionalmente, el nuevo sistema no estará en ambiente de producción hasta la finalización de la migración. Esto implica una desventaja debido a la necesidad de sincronizar la base de datos original y la nueva. Segundo, *Butterfly* propone un motor de migración de datos legados, por lo que el sistema legado solo debe ser "apagado" por un período corto de tiempo. Esto difiere de metodologías del estilo *Cold Turkey* donde el sistema legado debe ser desconectado por un largo período de tiempo para facilitar la migración de los datos antes de habilitar el nuevo sistema.

En la primer etapa, la metodología *Butterfly* considera los requerimientos del usuario para la migración y se diseña la arquitectura el sistema objetivo. Se deben definir *brenchmarks* para medir la calidad de la migración y seleccionar una arquitectura para el nuevo sistema, indicando el hardware seleccionado.

La segunda etapa es de ingeniería reversa. La metodología *Butterfly* indica que es recomendable utilizar herramientas existentes para la ingeniería inversa y desarrollar nuevas herramientas sólo si es absolutamente necesario. El objetivo de esta etapa es identificar redundancias de funcionalidad, determinar la función de las nuevas interfaces del sistema, definir un nuevo modelo de datos y definir reglas de asociación para migrar los datos desde el sistema legado hacia el nuevo sistema.

La principal actividad de la tercer etapa implica seleccionar información de ejemplo para construir una herramienta de transformación de datos utilizada por *Butterfly*. Esta herramienta transforma los datos legados en almacenamientos temporales para el nuevo sistema. Esta información de ejemplo servirá luego para testear el nuevo sistema.

Durante la cuarta etapa, la información temporal obtenida en la etapa previa será utilizada para administrar el ciclo de vida del desarrollo del nuevo sistema. Según la metodología *Butterfly* este ciclo se basa en "diseño-desarrollo-testing" aplicado a cada componente migrado, lo cual constituye un proceso incremental para la migración de interfaces del sistema legado. Opcionalmente, en esta etapa se puede realizar una Validación de Requerimientos del Usuario.

La quinta etapa constituye el núcleo de la metodología *Butterfly*, donde se migran gradualmente los datos legados hacia el nuevo sistema. Para esta etapa, *Butterfly* comprende tres componentes: *TempStores*, *Data Access Allocator* (*DAA*) y *Chrysaliser* (transformador de datos). Un *TempStore* es un componente que representa un almacenamiento temporal utilizado por el nuevo sistema en proceso de *testing*. Luego, cada *TempStore* es migrado hacia el nuevo modelo de datos utilizando *Chrysaliser*, un componente para la transformación de los datos. Finalmente, el acceso a los datos legados es redireccionado hacia el nuevo modelo de datos, mediante el componente DAA, el cual genera un nuevo *TempStore*. Este proceso se realiza iterativamente hasta que los datos legados han sido migrados completamente. Durante esta etapa, el sistema legado aún continúa en completo funcionamiento. Una vez alcanzado el nuevo sistema, incluyendo la migración de los datos, el mismo se encuentra listo para pasar al ambiente de producción. Según la metodología *Butterfly*, el proceso de desconexión del sistema legado para dar lugar al nuevo sistema se denomina *Cut-Over* y representa la sexta etapa.

En resumen, este enfoque se centra en la migración de datos legados por sobre la migración de funcionalidad. En este sentido, debido a que el proceso de migración de datos –utilizando *TempStores*– es incremental, la metodología *Butterfly* introduce una "Condición de Terminación", un umbral ε para determinar cuándo la migración ha alcanzado el requisito para el *Cut-Over* (finalización del proceso de migración). Según $size(TS_n) \leq \varepsilon$, donde TS_n representa el *TempStore* creado en la iteración n, se determina si el tiempo necesario para crear TS_n es menor a ε –que representa el tiempo que demora la herramienta DAA en transformar los datos de la iteración n– para permitir una migración efectiva sin comprometer la lógica de negocios. Esto es, cuando el tiempo de transformación de datos para un determinado $TempStore_i$ es superior al tiempo de generación de $TempStore_{i+1}$, finaliza el proceso.

Cabe mencionar también que, a excepción de las herramientas que forman parte del enfoque, *Butterfly* no considera otros aspectos técnicos. Sin embargo, el objetivo del enfoque consiste en generar un nuevo sistema por lo que, a rasgos generales, *Butterfly* abarca la mayor parte de las inquietudes y problemas que implican un proceso de migración, desde la captura de requerimientos, pasando por detalles de diseño, hasta el reemplazo del sistema original

completo.

En relación a *Cold Turkey* o *Chicken Little*, se pueden notar dos grandes diferencias. Primero, la metodología *Butterfly* no utiliza puertas de enlace (*Gateways*) durante el proceso de migración y los datos necesarios son almacenados sobre la base de datos legada. Segundo, con *Butterfly* ambos sistemas (legado y nuevo) no funcionan en paralelo. Durante el proceso de migración, se utiliza la base de datos legada que luego es migrada hacia el nuevo sistema. Cuando se alcanza el *Cut-Over*, el nuevo sistema entra en funcionamiento y el sistema legado es desconectado completamente.

IV. Comparación

A fin de resumir los distintos métodos y herramientas expuestos en el presente trabajo, se realizará una tabla comparativa por cada tipo de enfoque de migración. Cada tabla consta de cuatro columnas. La primera columna indica el nombre del enfoque. La segunda columna indica el principal objetivo que persigue el enfoque propuesto. Finalmente, en la tercera y cuarta columna se observan las principales ventajas y desventajas de cada enfoque, respectivamente.

La Tabla I compara los enfoques de reemplazo. Si bien los tres enfoques permiten generar un sistema completamente moderno e independiente del sistema original, el enfoque *Cold Turkey* implica un riesgo mayor que los enfoques *Chicken Little* y *Butterfly*.

TABLA I

Nombre	Ventajas	Desventajas
Cold Turkey	Independencia del sistema original	Complejo y riesgoso, alta probabilidad de fracaso
Chicken Little	Enfoque iterativo y gradual; disminución del riesgo	Degradación de *performance* porque los *gateways* agregan indirección
Butterfly	El sistema legado sólo es desconectado por cortos períodos de tiempo; soporte para migración de datos legados; independencia del sistema original	Requiere sincronizar ambas bases de datos; requiere recursos para mantener ambos sistemas en simultáneo

La Tabla II presenta una comparación entre los enfoques de migración directa. Para los diversos enfoques existen diferentes objetivos a ser resueltos durante la migración. Por ejemplo, *Screen Scraping* se centra en la modernización de la interfaz visual del sistema legado. Por otro lado, los enfoques de Sneed, Canfora et al., Sneed & Sneed, COB2WEB, Alahmari et al. y Salvatierra et al. se centran en la identificación de Servicios Web candidatos. Finalmente, CelLEST se centra en la migración de módulos legados en base a las interacciones entre el usuario y el sistema.

TABLA II

Nombre	Objetivo	Ventajas	Desventajas
Screen Scraping	Generación de GUI moderna	Resultados a corto plazo	Sólo se moderniza la interfaz visual; dependencia con el sistema original
Sneed	Identificación y exposición de Servicios Web	Considera los problemas técnicos; resultados a corto plazo	Requiere interpretación por parte de personal experto; baja performance sobre sistemas grandes
Canfora et al.	Identificación y exposición de Servicios Web	Resultados a corto plazo	Requiere mucho trabajo manual para interpretar el análisis; dependencia con el sistema original
CelLEST	Migración de módulos legados en base a interacción entre el sistema y el usuario	No requiere contratación de personal experto; no requiere entendimiento del código legado; independencia de la funcionalidad legada	Requiere personal familiarizado con el sistema original; dificultoso si el número de interacciones entre el usuario y el sistema legado es muy grande
Sneed & Sneed	Identificación y exposición de Servicios Web	Permite desacoplar un sistema altamente acoplado; resultados a corto plazo	Requiere etapa de pre-procesamiento por parte de expertos; dependencia con el sistema original
COB2WEB	Identificación e implementación de Servicios Web candidatos	Modernización automática de la funcionalidad legada	Requiere interpretación de los resultados por parte de expertos
Alahmari et al.	Identificación de Servicios Web candidatos	Enfoque simple; permite generar un conjunto de interfaces de Servicios Web con granularidad óptima; independiente de las tecnologías de implementación de los servicios	Sólo se enfoca en identificar servicios; no hay soporte para implementación de los servicios candidatos; no provee información sobre la relación con el sistema legado luego de la migración
Salvatierra et al.	Identificación de Servicios Web candidatos y diseño de documentos WSDL	No requiere comprensión de la funcionalidad legada; puede ser aplicado por el mismo personal que mantiene el sistema legado; los WSDLs resultantes pueden ser reutilizados en migración indirecta	La generación de documentos WSDL debe ser realizada manualmente; requiere conocimiento sobre buenas prácticas en diseño de WSDLs; dependencia con el sistema original

En la Tabla III se observa una comparación entre los enfoques de migración indirecta. Estos esfuerzos intentan resolver varios aspectos importantes en la modernización de un sistema legado. Así por ejemplo, SoSR propone una modernización completa hacia SOA. Distante et al. se centra en la generación de una aplicación Web. Cheng et al. intenta identificar e implementar Servicios Web candidatos. Cuadrado et al. aplica una estrategia enfocada en refactorizar la arquitectura del sistema legado. Finalmente, Van Deursen & Kuipers intenta resolver problemas inherentes al análisis y entendimiento de un sistema legado a fin de identificar módulos reutilizables. Como se puede apreciar en el Cuadro

III todos estos enfoques intentan modernizar un sistema legado fuertemente, generando una independencia completa del sistema original.

TABLA III

Nombre	Objetivo	Ventajas	Desventajas
SoSR	Modernización completa a SOA	Permite desacoplar sistemas con módulos legados altamente acoplados; independencia del sistema original	Requiere personal altamente especializado en arquitecturas debido al uso de diversas vistas; enfoque en general complejo
Distante et al.	Generación de aplicación Web	Independencia del sistema original conservando las restricciones de negocios originales	Requiere mucho tiempo para alcanzar el nuevo sistema; necesidad de contratar personal experto
Cheng et al.	Identificación e implementación de Servicios Web candidatos	Independencia del sistema original	Demasiado complejo para sistemas con grandes volúmenes de datos
Cuadrado et al.	Recuperación de arquitectura	Permite obtener un reporte detallado de la funcionalidad del sistema legado; independencia del sistema original	Sólo aplicable a sistemas de poco porte
Van Deursen & Kuipers	Análisis, entendimiento e identificación de módulos legados reutilizables	Informe detallado de la funcionalidad legada; se puede adaptar de forma sencilla a las particularidades de cada sistema	Se invierte demasiado tiempo para reducir el tamaño del código legado a analizar; presenta limitaciones durante el análisis de sentencias; asume la existencia de base de datos; sensible a la convención de nombres de variables

Finalmente, la Tabla IV compara los enfoques de migración directa e indirecta híbridos. Ambos enfoques utilizan técnicas de ingeniería inversa y *wrapping* para obtener un nuevo sistema SOA. Asimismo, O'Brien et al. centra su enfoque en la reestructuración de la arquitectura legada concentrándose también en aspectos técnicos y de diseño. Por otro lado, SMART es un enfoque más general que utiliza un conjunto de guías de migración y considera aspectos de ingeniería de software tales como la captura de requerimientos y priorización de atributos de calidad, sin tener en cuenta aspectos técnicos.

TABLA IV

Nombre	Objetivo	Ventajas	Desventajas
O'Brien et al.	Identificación e implementación de Servicios Web candidatos	Permite reestructuración completa de la arquitectura; independencia del sistema original	Requiere entendimiento previo del sistema legado; pueden surgir inconsistencias heredadas del sistema original
SMART	Generación de nuevo sistema SOA	Enfoque de alto nivel centrado en requerimientos; flexibilidad para combinar estrategias para las diferentes etapas del enfoque; independencia del sistema original	No tiene en cuenta aspectos técnicos; puede resultar complejo de aplicar

Con respecto a los enfoques de reemplazo, se concluye que son complejos de aplicar y poseen alto riesgo de fallos. En [14] se identifican dos riesgos en los enfoques de reemplazo: mantener el nuevo sistema será completamente distinto a mantener el sistema legado; asegurar que el nuevo sistema cumplirá con las reglas de negocios del sistema original.

En cuanto a los enfoques de modernización, se puede concluir que los enfoques de migración directa implican un riesgo menor que los enfoques de migración indirecta. No obstante, estos últimos permiten independencia del sistema original y, en general, mejor calidad en el sistema resultante.

Otro factor de comparación importante es el tiempo para lograr una migración efectiva. Resulta evidente que reemplazar completamente un sistema legado, o modernizarlo mediante migración indirecta, implica mayor tiempo que la modernización mediante migración directa. Incluso, dependiendo el enfoque, una migración exitosa puede tardar años en completarse.

Adicionalmente, este trabajo describe algunos ejemplos de investigaciones donde los autores han decidido centrar sus enfoques en el nivel técnico mediante una herramienta o *framework* de migración. Así, se amplía la variedad de tácticas para migrar un sistema legado. Por ejemplo, la herramienta propuesta por [30] –COB2WEB– funciona como soporte para un enfoque de migración directa, debido a que el conjunto de herramientas propuesto permite automatizar la identificación e implementación de servicios candidatos. Asimismo, el *framework* propuesto por [1] se interpreta como un soporte para un enfoque de migración directa, automatizando la identificación de Servicios Web candidatos. Finalmente, la herramienta propuesta por [17] se enfoca en el análisis de la funcionalidad legada a fin de identificar módulos legados reutilizables con el objetivo de reescribir el código fuente en un enfoque de migración indirecta.

V. Conclusiones

Seleccionar una estrategia de migración implica tener en cuenta una amplia variedad de aspectos. Claramente, el aspecto más importante radica en las fortalezas y debilidades de cada enfoque. Los enfoques de reemplazo de sistemas legados pueden ser prácticamente inviables en la mayoría de los casos. Por ejemplo, los enfoques *Cold Turkey* o *Chicken Little* requieren de interoperabilidad entre el sistema legado y el nuevo sistema, incluso llegando a desconectar el sistema legado por largos períodos de tiempo, lo cual puede llegar a ser inaceptable para organizaciones que requieren funcionalidad 24x7. Considerando que estos sistemas pertenecen a grandes empresas o entidades gubernamentales,

este tipo de enfoque puede resultar altamente costoso en tiempos y dinero.

Algunos enfoques promueven el reemplazo completo de un sistema en etapas no iterativas. Esto significa apagar el sistema legado y encender el nuevo sistema a la vez, lo cual es altamente riesgoso debido a que el nuevo sistema se encuentra en producción, no pasando por un apropiado proceso de *testing*. Por otro lado, algunos enfoques se centran en mantener dos sistemas al mismo tiempo. Esto resulta costoso debido a que se requiere de mayor cantidad de recursos para soportar ambos ambientes y personal experto en las nuevas tecnologías. Adicionalmente, el riesgo se incrementa debido a la posible falta de consistencia entre ambos sistemas en aquellos dominios donde las reglas del negocio cambian constantemente, y la sincronización puede resultar compleja. En resumen, estos enfoques de migración omiten muchos detalles o son demasiado complejos para ser aplicados en la práctica. Específicamente, el enfoque *Chicken Little* es un enfoque maduro, pero la interoperabilidad requerida entre el sistema legado y el nuevo sistema agrega complejidad a la solución.

Cuando el camino a seguir es la modernización, el riesgo de fallos disminuye, así como también los tiempos y costos de migración. No obstante, estos enfoques requieren de una conexión entre el sistema legado y el nuevo sistema, incluso luego de finalizada la migración. Adicionalmente, hay que considerar las ventajas y desventajas de los dos tipos enfoques de modernización. Por ejemplo, el enfoque propuesto por [32] es más aplicable a programas pequeños ya que la identificación y exposición de lógica de negocios puede consumir demasiado tiempo. La técnica de encapsulamiento propuesta por [7], si bien está destinada a generar interfaces de servicios automáticamente, necesita bastante interacción manual que puede tornarse complicada debido al uso e interpretación de una Máquina de Estados Finitos. El proceso propuesto por [34] aporta la ventaja de la independencia de código pero su complejidad puede aumentar cuando el número de interacciones entre los usuarios y el sistema legado es muy grande. En cuanto a enfoques de migración indirecta, todas las investigaciones resumidas en el presente trabajo requieren de métodos complejos y mayor tiempo para alcanzar el objetivo de migrar un sistema legado. En todos los casos, se requieren herramientas especializadas para automatizar el proceso de migración, o partes del mismo.

En resumen, un enfoque de migración directa implica menos costos y tiempos de desarrollo para alcanzar el objetivo pero, como desventaja, no permite conseguir un nuevo sistema independiente del sistema legado original. Por el contrario, un enfoque de migración indirecta permite conseguir un sistema completamente modernizado, tentativamente independiente de los módulos legados del sistema original, pero con mayores costos.

Otro aspecto a tener en cuenta a la hora de seleccionar un enfoque de migración, es el tipo de organización que posee el software y quiénes lo utilizan. Esto surge en relación al *trade-off* entre los costos de la migración y la calidad del sistema resultante . Esto es imperioso para organizaciones que buscan resultados efectivos a corto plazo y al menor costo posible.

En conclusión, las investigaciones y herramientas reportadas recientemente se centran en la automatización de diferentes estrategias de migración, lo cual siempre requiere de personal experto tanto en el dominio del sistema legado como en los métodos y herramientas de migración utilizados. Este hecho hace que este tipo de modernización sea también costosa. Tal es así, que la mayoría de las organizaciones se encuentran actualmente en el escenario de migración directa de sus sistemas legados a SOA [2], [13], [28], [22] debido a que este enfoque es el menos costoso y requiere menos tiempo en completarse, lo cual permite a una organización mantener su sistema vigente al mismo tiempo que se incorpora una nueva capa de software a su alrededor para mejorar la integración con tecnologías modernas.

REFERENCIAS

[1] S. Alahmari, E. Zaluska, and D. D. Roure. A Service Identification Framework for Legacy System Migration into SOA. *IEEE International Conference on Services Computing*, pages 614–617, 2010.

[2] A. Ahmed, MA. Almonaies, J. Cordy, and T. Dean. Legacy System Evolution towards Service-Oriented Architecture. In *International Workshop on SOA Migration and Evolution*, pages 53–62, 2010.

[3] L. Aversano, G. Canfora, A. Cimitile, and A. de Lucia. Migrating Legacy Systems to the Web: an Experience Report. In *Fifth European Conference on Software Maintenance and Reengineering*, page 148, Washington, DC, USA, 2001. IEEE Computer Society.

[4] J. Bisbal, D. Lawless, and R. Richardson. A Survey of Research into Legacy System Migration. Technical report, Broadcom Research and Trinity College, 1997.

[5] M. Brodie and M. Stonebraker. DARWIN: On the Incremental Migration of Legacy Information Systems. Technical report, Berkeley, CA, USA, 1993.

[6] M. Brodie and M. Stonebraker. *Migrating legacy systems: gateways, interfaces & the incremental approach*. Morgan-Kaufmann Publishers Inc., San Francisco, CA, USA, 1995.

[7] G. Canfora, A. R. Fasolino, G. Frattolillo, and P. Tramontana. Migrating Interactive Legacy Systems to Web Services. In *Conference on Software Maintenance and Reengineering*, pages 24–36, Washington, DC, USA, 2006. IEEE Computer Society.

[8] G. Canfora, A. R. Fasolino, G. Frattolillo, and P. Tramontana. A wrapping approach for migrating legacy system interactive functionalities to Service Oriented Architectures. *Journal of Systems and Software*, 81(4):463–480, 2008.

[9] F. Chen, S. Li, and W. C.-C. Chu. Feature Analysis for Service-Oriented Reengineering. In *12th Asia-Pacific Software Engineering Conference*, pages 201–208, Washington, DC, USA, 2005. IEEE Computer Society.

[10] S. Chung, J. B. C. An, and S. Davalos. Service-Oriented Software Reengineering: SoSR. In *40th Annual Hawaii International Conference on System Sciences*, HICSS '07, pages 172c–, Washington, DC, USA, 2007. IEEE Computer Society.

[11] S. Chung, S. Davalos, J. B. C. An, and K. Iwahara. Legacy to web migration: service-oriented software reengineering methodology. *International Journal of Services Sciences*, 1:333 – 365, 2008.

[12] S. Chung, P. S. Young, and J. Nelson. Service-Oriented Software Reengineering: Bertie3 as Web Services. *Web Services, IEEE International Conference on*, 0:837–838, 2005.

[13] M. Colosimo, A. D. Lucia, G. Scanniello, and G. Tortora. Evaluating legacy system migration technologies through empirical studies. *Inf. Softw. Technol.*, 51:433–447, February 2009.

[14] S. Comella-Dorda, K. Wallnau, R. C. Seacord, and J. Robert. A Survey System Modernization Approaches. In *Seminal works and reference material created by SEI staff*, 2000. CMU/SEI-2000-TN-003.

[15] F. Cuadrado, B. García, J. C. Dueñas, and H. A. Parada. A Case Study on Software Evolution towards Service-Oriented Architecture. In *22nd International Conference on Advanced Information Networking and Applications - Workshops*, pages 1399–1404, Washington, DC, USA, 2008. IEEE Computer Society.

[16] A. De Lucia, R. Francese, G. Scanniello, and G. Tortora. Developing legacy system migration methods and tools for technology transfer. *Softw. Pract. Exper.*, 38:1333–1364, November 2008.

[17] A. V. Deursen and T. Kuipers. Rapid System Understanding: Two COBOL Case Studies. *International Conference on Program Comprehension*, 0:90, 1998.

[18] D. Distante, S. Tilley, and G. Canfora. Towards a Holistic Approach to Redesigning Legacy Applications for the Web with UWAT. In *Conference on Software Maintenance and Reengineering*, pages 295–299, Washington, DC, USA, 2006. IEEE Computer Society.
[19] A. Lakhotia and J.-C. Deprez. Restructuring functions with low cohesion. In *Sixth Working Conference on Reverse Engineering*, WCRE '99, pages 36–, Washington, DC, USA, 1999. IEEE Computer Society.
[20] G. Lewis, E. Morris, and D. Smith. Analyzing the Reuse Potential of Migrating Legacy Components to a Service-Oriented Architecture. *Software Maintenance and Reengineering, European Conference on*, 0:15–23, 2006.
[21] G. Lewis, E. Morris, D. Smith, and L. O'Brien. Service-Oriented Migration and Reuse Technique (SMART). In *13th IEEE International Workshop on Software Technology and Engineering Practice*, pages 222–229, Washington, DC, USA, 2005. IEEE Computer Society.
[22] S.-H. Li, S.-M. Huang, D. C. Yen, and C.-C. Chang. Migrating Legacy Information Systems to Web Services Architecture. *Journal of Database Management*, 18(4):1–25, 2007.
[23] C. Mateos, M. Crasso, J. M. Rodriguez, A. Zunino, and M. Campo. Measuring the impact of the approach to migration in the quality of web service interfaces. *Enterprise Information Systems*, 0:1–28, 2012.
[24] L. O'Brien, D. Smith, and G. Lewis. Supporting Migration to Services using Software Architecture Reconstruction. In *13th IEEE International Workshop on Software Technology and Engineering Practice*, pages 81–91, Washington, DC, USA, 2005. IEEE Computer Society.
[25] Oracle. Mainframe Rehosting with Oracle Tuxedo: Accelerating Cost Reduction and Application Modernization. Technical report, Oracle, 2009.
[26] M. Razavian and P. Lago. Understanding SOA Migration Using a Conceptual Framework. In *Journal of Systems Integration*, volume Vol. 3, pages 1–11, 2010.
[27] J. Rodriguez, M. Crasso, A. Zunino, and M. Campo. Automatically Detecting Opportunities for Web Service Descriptions Improvement. In W. Cellary and E. Estevez, editors, *Software Services for e-World*, volume 341 of *IFIP Advances in Information and Communication Technology*, pages 139–150. Springer Boston, 2010.
[28] G. Salvatierra, C. Mateos, M. Crasso, and A. Zunino. Towards a Computer Assisted Approach for Migrating legacy Systems to SOA. In *12th International Conference on Computational Science and Its Applications (ICCSA 2012)*, Lecture Notes in Computer Science, pages 484–497. Springer, 2012.
[29] D. Smith. Migration of Legacy Assets to Service-Oriented Architecture Environments. In *29th International Conference on Software Engineering*, ICSE COMPANION '07, pages 174– 175, Washington, DC, USA, 2007. IEEE Computer Society.
[30] Sneed and Vienna. COB2WEB A Toolset for migrating to Web Services. *Web Site Evolution, 2008. WSE 2008. 10th International Symposium on*, 2:19 – 25, 2008.
[31] H. M. Sneed. Integrating legacy Software into a Service Oriented Architecture. In *Conference on Software Maintenance and Reengineering*, pages 3–14, Washington, DC, USA, 2006. IEEE Computer Society.
[32] H. M. Sneed. Wrapping Legacy Software for Reuse in a SOA. In F. Lehner, H. Nösekabel, and P. Kleinschmidt, editors, *Multikonferenz Wirtschaftsinformatik 2006*, volume 2 of *XML4BPM Track*, pages 345–360. GITO-Verlag Berlin, 2006.
[33] H. M. Sneed and S. H. Sneed. Creating Web Services from Legacy Host Programs. In *WSE'03*, pages 59–65, 2003.
[34] E. Stroulia, M. El-Ramly, P. Sorenson, and R. Penner. Legacy systems migration in CelLEST. In *22nd International Conference on Software Engineering*, ICSE '00, pages 790–, New York, NY, USA, 2000. ACM.
[35] E. Stroulia, M. El-Ramly, P. Sorenson, and R. Penner. FromLegacy to Web through Interaction Modeling. In *International Conference on Software Maintenance*, pages 320–330, Washington, DC, USA, 2002. IEEE Computer Society.
[36] S. Tilley and D. Smith. Perspectives on legacy system reengineering, 1995.
[37] N. Weiderman, L. Northrop, D. Smith, and S. Tilley. The Impact of Distributed Object Technology on Reengineering. In *International Conference on Software Maintenance*, pages 300– 308, Washington, DC, USA, 1997. IEEE Computer Society.
[38] B. Wu, D. Lawless, J. Bisbal, and R. Richardson. The butterfly methodology: A gateway-free approach for migrating legacy information systems. In *3rd IEEE Conference on Engineering of Complex Computer Systems*, pages 200–205. IEEE Computer Society Press, 1997.
[39] Z. Zhang, R. Liu, and H. Yang. Service Identification and Packaging in Service Oriented Reengineering. In *7th International Conference on Software Engineering and Knowledge Engineering*, pages 241–249, 2005.

Gonzalo Salvatierra is currently working on his M.Sc. thesis about legacy COBOL system migration to service oriented platforms at UNICEN. He is working under the supervision of Marco Crasso and Cristian Mateos. He holds a Systems Engineer degree from the UNICEN.

Cristian Mateos received a Ph.D. degree in Computer Science from the UNICEN, in 2008, and M.Sc. in Systems Engineering in 2005. He is a full time Teacher Assistant at the UNICEN and member of the ISISTAN and the CONICET. He is interested in parallel/distributed programming, grid middlewares, and service-oriented computing

Marco Crasso received a Ph.D. degree in Computer Science from the UNICEN in 2010. He is member of the ISISTAN and the CONICET. His research interests include Web service discovery and programming models for SOA.

Alejandro Zunino received a Ph.D. degree in Computer Science from the UNICEN, in 2003, and his M.Sc. in Systems Engineering in 2000. He is a full Adjunt Professor at UNICEN and member of the ISISTAN and the CONICET. His research areas are Grid computing, service-oriented computing, Semantic Web services, and mobile agents.

Marcelo Campo received a Ph.D. degree in Computer Science from the Universidade Federal do Rio Grande do Sul, Brazil, in 1997. He is a full Associate Professor at the UNICEN, head of the ISISTAN, and member of the CONICET. His research interests include intelligent aided software engineering, software architecture, and frameworks.