UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Título: Migración semiautomática de sistemas Legacy hacia arquitecturas orientadas a servicios
Autores: Barzola, Mauro Ernesto
Director: Urbieta, Matías
Codirector: -
Asesor profesional: -
Carrera: Licenciatura en Sistemas

Resumen

Los sistemas legacy son sistemas informáticos que han quedado anticuados pero continúan siendo de vital importancia para dar soporte al negocio de numerosas empresas de software. Generalmente, este tipo de software no se puede reemplazar o actualizar de forma sencilla al buscar integrarse con nuevas tecnologías o escalar en funcionalidades.
El objetivo de este trabajo es definir un enfoque teórico-práctico que permita realizar una migración e integración de sistemas legacy hacia arquitecturas orientadas a servicios (SOA) de forma semiautomática. Por ejemplo, partiendo de tecnologías tales como Visual Basic o Delphi poder migrar hacia tecnologías modernas Web o Mobile apoyándose en un proceso de migración que luego podrá ser implementado sobre cualquier tecnología.
En este trabajo se desarrolló un enfoque teórico de migración el cual luego se implementó semiautomáticamente sobre tecnologías específicas realizando pruebas de conceptos sobre un sistema legacy real. Además, se implementó un prototipo de App Mobile con la finalidad de ejemplificar la facilidad de integración de la arquitectura SOA resultante.

Palabras Claves

Sistemas Legacy - Arquitecturas orientadas a servicios Migración semi-automática - Servicios Web - Refactoring - Delphi - SQL

Trabajos Realizados

Se comenzó analizando el estado de arte de migración de sistemas legacy. Luego, se desarrolló un modelo de solución sistemático teórico. Luego se instalaron y evaluaron distintas herramientas relacionadas a la automatización de cada etapa del proceso. Luego, se implementó el proceso propuesto sobre tecnologías específicas y se analizaron los resultados obtenidos. Por último, se desarrolló una App Mobile con la finalidad de ejemplificar la facilidad de integración con la arquitectura resultante.

Conclusiones

Migrar sistemas legacy no es un trabajo trivial, implementar una migración sobre mecanismos automatizables es un tanto ambicioso, pero no menos motivador. Disponer de un proceso de migración con un enfoque teórico-práctico sobre el cual guiar la implementación del caso de prueba arrojó resultados alentadores. La relación costo-tiempo entre la migración de un módulo o su reimplementación completa es muy marcada a favor de la migración. Con el proceso propuesto se logró un rápida migración hacia servicios.

Trabajos Futuros

El proceso propuesto toma como entrada orígenes de datos DBMS, pero se podría ampliar a otros tipos de orígenes de datos más antiguos como por ejemplo archivos o motores NoSQL. También se plantea la posibilidad de automatizar el refactor desde el código monolítico hacia SOA, por ejemplo definiendo reglas de derivación que permitan automatizarse. Por último, el proceso se podría profundizar definiendo pautas para implementar migración paralela sobre grandes equipos de trabajo.

Fecha de la presentación: Junio 2017

Migración semiautomática de sistemas Legacy hacia arquitecturas orientadas a servicios

Una Tesis Presentada Para Obtener El Título De
Licenciado en Sistemas
Universidad Nacional de La Plata

Mauro Ernesto Barzola.

Esta tesis está dedicada a mi familia, quienes me brindan todo su apoyo, cariño y compresión y han sido la fuente de inspiración a lo largo de todo el camino recorrido en búsqueda de mis objetivos.

Agradecimientos

En primer lugar, un sincero agradecimiento al Doctor Matías Urbieta quién aceptó dirigir esta tesis. Sus aportes nutrieron la presente tesis en base a su gran experiencia y conocimientos en el campo estudiado.

Y por supuesto, un gran agradecimiento a los integrantes de mi familia que fueron el pilar fundamental durante toda mi carrera.

Tabla de Contenidos

Índice de ilustraciones

Índice de tablas

Capítulo 1 - Introducción

1.1. Objetivos generales

"Los sistemas legacy constituyen un patrimonio muy importante en numerosas empresas de software. A menudo, los sistemas legacy son relativamente viejos, basados en mainframes que fueron optimizados para plataformas arcaicas de software y hardware" [Willem-Jan 2006]. Un sistema de este tipo ha quedado anticuado pero continúa siendo un activo muy importante para dar soporte al negocio y no se quiere o no se puede reemplazar o actualizar de forma sencilla.

La complejidad relacionada al reemplazo o actualización de un sistema legacy está dada principalmente por:

- Documentación nula, escasa o desactualizada.
- Miles o millones de líneas de código.
- Ausencia de código fuente.
- Pérdida de conocimientos del negocio.
- Procesos críticos de negocios que no se pueden detener.
- Personal sin conocimiento o desinteresado en tecnologías legacy.

Uno de los puntos claves a considerar al momento de plantear un reemplazo o actualización de un sistema legacy está relacionado al conocimiento de tecnologías por parte del los recursos humanos abocados al proyecto. Está claro que con el avance de las tecnologías, la disponibilidad de personal con conocimientos en tecnologías anticuadas es cada vez menor. A continuación, se presenta un resumen (Tabla 1) de estadísticas provistas por The Importance of Being Earnest [TIOBE 2017] acerca de los lenguajes de programación más utilizados actualmente donde se refleja claramente la poca disponibilidad de recursos con experiencia en sistemas legacy.
Las estadísticas muestran cómo las tecnologías actuales líderes como Java o C concentran la mayor parte de los recursos en relación a su utilización, y cómo decrece el uso de tecnologías que fueron estándares años atrás como es el caso de Visual Basic o Delphi.

Posición 2016	Posición 2015	Tecnología	Rating %
1	1	Java	20,84
2	2	C	13,90
3	3	C++	5,91
4	5	C#	3,79
5	2	Python	3,33
10	10	Visual Basic .Net	2,27
11	11	Delphi / Object Pascal	2,21
14	9	Visual Basic	1,60
21		COBOL	1,01
23		SAS	0,92

Tabla 1 : Estadísticas sobre uso lenguajes

Tras la necesidad de modernizar sistemas legacy buscando integrar con nuevas plataformas (web, móviles, etc.), escalando en funcionalidad o para satisfacer nuevos requerimientos, es común caer en las clásicas estrategias que se basan en remplazar total o gradualmente el sistema o planear una migración tecnológica [Rodríguez 2001]. Al remplazar el sistema es imposible reutilizar componentes que son efectivos y confiables tras años de desarrollos y pruebas constantes. El migrar hacia nuevas tecnologías es un desafío muy riesgoso que no siempre garantiza su éxito lo que podría llevar a grandes pérdidas a las compañías que no lo implementen satisfactoriamente.

Hoy, el avance en las plataformas y paradigmas tecnológicos nos permiten desarrollar aplicaciones que se integran fácilmente entre distintas implementaciones permitiendo la creación de sistemas de información altamente escalables. Por ejemplo, se podría destacar el desarrollo orientado a servicios, donde básicamente las funcionalidades se encapsulan en servicios que pueden ser accedidos desde distintas aplicaciones clientes, aplicaciones web, aplicaciones móviles, etc. Por tal motivo, si se lograse una actualización tecnológica del sistema legacy el mismo podría explotar tales características y así poder reutilizar funcionalidades existentes vitales del negocio.

En este contexto, y dado que los sistemas legacy poseen funcionalidades que serían convenientes reaprovechar (dado que son funcionalidades muy estables, probadas y utilizadas por mucho tiempo), se hace necesario contar con una guía sistemática que defina estrategias claras para reducir a la mínima expresión los catastróficos resultados de una migración que no cumpla con las expectativas.
Con base en tecnologías actuales con especificaciones de estándares es posible integrar nuevas plataformas con sistemas legacy [Hasselbring 2000].

El objetivo de esta tesis es definir un enfoque que permita realizar una migración e integración de sistemas legacy hacia arquitecturas orientadas a servicios (SOA). Por ejemplo, partiendo de tecnologías tales como Visual Basic o Delphi poder migrar hacia tecnologías modernas Web o Mobile de forma semi-automática apoyándose en un proceso de migración que luego podrá ser implementado con cualquier tecnología.

1.2. Objetivos específicos

- Estudiar el estado de arte de estrategias de migración de un sistema legacy a SOA.
- Analizar estrategias de refactoring sobre sistemas legacy.
- Analizar estrategias de migración de interfaces de usuario.
- Profundizar el análisis de migración para un sistema concreto desarrollado en Delphi con la finalidad de exponer sus funcionalidades hacia entornos Webs o Mobiles.
- Analizar el estado de arte de herramientas disponibles para migración e integración de sistemas.
- Estudiar las ventajas de migrar un sistema legacy a una arquitectura orientada a servicios.
- Proponer un proceso de migración teórico para luego ser implementado con tecnologías concretas.
- Proponer una guía práctica, sistemática y semiautomática para migración de sistemas legacy.

- Realizar una prueba de concepto sobre un sistema real que permita aplicar las contribuciones de esta tesis. Se propone implementar el proceso de migración resultante del estudio y luego validar la integración de la arquitectura resultante contra una app mobile.

1.3. Contexto del negocio

Hace más de una década nació Inmosoft [Inmosoft], un proyecto personal motivado por mi ingreso a la universidad y las ganas de llevar a la práctica lo que iba aprendiendo. Inmosoft es un software especializado en la gestión integral para inmobiliarias, abarcando desde simples circuitos hasta los más avanzados y productivos para el sector.

Con el correr de los años, Inmosoft fue creciendo y sumando tanto clientes como funcionalidades. Actualmente Inmosoft satisface más de cien clientes en el país, cientos de usuarios trabajan concurrentemente y mes a mes se incrementan estos índices.
Hoy ya finalizando la carrera de Licenciatura en Sistemas, Inmosoft es un producto estable y afianzado en el mercado pero con las limitaciones tecnológicas que implican a un software comenzando hace muchos años atrás.

Dado que Inmosoft está implementado en Delphi 6, una tecnología prácticamente obsoleta, y se requiere de ciertas funcionalidades disponibles tanto para aplicaciones móviles como para aplicaciones web la prueba de concepto que permitirá aplicar las contribuciones de esta tesis se realizará sobre el sistema Inmosoft.

1.4. Motivación

Los sistemas legacy fueron, son y probablemente seguirán siendo el corazón de muchas organizaciones que automatizaron sus procesos de negocios hace décadas bajo tecnologías que actualmente son anticuadas pero siguen siendo útiles para los usuarios y de gran valor para las empresas.

Hoy en día, el acelerado avance tecnológico, la evolución de paradigmas y plataformas hacen que la gran mayoría de negocios y organizaciones basen sus procesos en sistemas informáticos, lo que hace imprescindible que estos nuevos procesos convivan con los sistemas legacy.

Esta clase de sistemas al igual que las organizaciones deben evolucionar y adaptarse a nuevos requerimientos.

Según [Rodríguez 2001] las soluciones clásicas relacionadas son, el rediseño completo del sistema en una nueva tecnología, el wrapping del sistema legacy que provee una nueva interfaz para algún componente de éste, lo que permite mayor accesibilidad desde otras aplicaciones y la migración o transformación que mueve al sistema legacy a un nuevo ambiente o plataforma más flexible, reteniendo la funcionalidad y los datos del sistema original.
Una de las técnicas para migrar sistemas legacy hacia nuevas tecnologías, es realizando llamadas a las funcionalidades del sistema legacy existente mediante el uso

del Wrapper [Sneed 2004]. Actualmente, la necesidad de incorporar a Internet servicios proporcionados por sistemas legacy, conjuntamente con la mayor difusión de Corba, J2EE, COM y .NET, ha impulsado notoriamente esta idea.
Mediante wrapper, es posible adaptar al sistema legacy, para que sea parte de una nueva generación de sistemas, sin los riesgos inherentes a la reescritura y los altos costos de nuevos desarrollos. Es evidente, que para que esto no introduzca problemas a futuro, el estado de salud del sistema debe ser "sano" o debe ser factible pasarlo a dicho estado.
Otro de los aspectos importantes de esta técnica, es que permite adaptar partes de un sistema según las necesidades y las urgencias, manteniendo el resto inalterado.

Los estudios de [BACH 2006] indican que los sistemas legacy son considerados potencialmente problemáticos por numerosos ingenieros de software por diversos motivos. Muchas organizaciones no han logrado los resultados esperados tras migrar, re implementar o integrar sus sistemas legacy.

Muchos de los sistemas de información todavía corren sobre plataformas legacy pero el deseo de reemplazar estos sistemas se ve limitado por varias restricciones tales como [Len Erlikh 2003]:

- Grandes inversiones acumuladas en el sistema legacy en largo tiempo.
- Disminución del personal técnicamente habilitado.
- Pérdida significativa de conocimiento sobre las tareas internas que trabajan estos sistemas.

Entre los muchos ejemplos de sistema actuales que siguen funcionando bajo plataformas legacy, podemos mencionar:

- Aplicaciones gubernamentales.
- Sistemas contables.
- Sistemas bancarios.
- Aplicaciones Rapid Application Development (RAD) orientadas a interfaces gráficas.
- Sistemas empresariales basados en procesos de negocios.

Es demasiado el esfuerzo y el costo invertido como para arriesgarse a que un proyecto de migración o integración no cumpla con las expectativas, es por eso que bajo este contexto, esta tesis propone profundizar la problemática detallando de cada uno de los posibles escenarios, analizar las herramientas disponibles y proponer una guía actualizada teórico-práctica para lograr una migración o integración exitosa.

Un aspecto importante a tener en cuenta al momento de migrar sistemas legacy son las interfaces de usuario [Grechanik 2007]. Hace tiempo atrás era muy común el desarrollo RAD lo que generó el problema del acoplamiento entre la interfaz de usuario y la lógica de negocios. Es de interés para esta tesis estudiar el tema en detalle con el objetivo de simplificar migraciones que lidien con esta problemática.
Para llevar a delante el plan, se propone la migración e integración de una aplicación desktop a web/mobile de forma semi-automática. Es importante destacar, que se llevará

a cabo un gran cambio conceptual de arquitectura en los módulos que se migren, en concreto, se expondrán las funcionalidades migradas bajo una arquitectura orientada a servicios [Microsoft Web Services].

Esta tesis usará para su prueba de concepto módulos de una aplicación Desktop Borland Dephi [Delphi] con más de 10 años de constante desarrollo con el fin de exponer sus funcionalidades bajo una arquitectura orientada a servicios de Microsoft, en concreto Web Services .Net.

1.5. Organización de la tesis

El documento de tesis está organizado en ocho capítulos claramente identificables, el capítulo inicial nos introduce en la problemática a resolver, nos detalla los objetivos perseguidos y nos ofrece la motivación del estudio.

En los capítulos dos, tres y cuatro se realiza un estudio de los temas principales que componen el backgroung del tema en cuestión, se profundiza el estudio sobre sistemas legacy, migración de sistemas legacy y arquitecturas orientadas a servicios.

En el capitulo cinco, se propone un proceso de migración como solución teórica a la problemática estudiada.

En el capitulo seis, se plantea como caso de estudio el ejercicio de implementar el proceso de migración teórico sobre tecnologías concretas.

En el capítulo siete, se exponen y detallan los escenario e ideas que quedaron planteados como trabajo a futuro.

En el capítulo ocho, se mencionan trabajos relacionados al estudio y por último, el capítulo final nos resume las conclusiones del estudio aplicado.

Capítulo 2 - Sistemas legacy

2.1. Definición

Según [Willem-Jan 2006] los sistemas legacy son aplicaciones caracterizadas por ser antiguas debido al periodo de vida en que estos sistemas fueron puestos en producción. Son sistemas de tecnología obsoleta, es decir, que actualmente siguen funcionando bajo plataformas de hardware o software antiguas. Los sistemas legacy en general, no tienen técnicas de estructuración de sistemas, se los conoce como sistemas monolíticos al no disponer de arquitectura en capas. Otra característica a resaltar de este tipo de sistemas es la nula o escasa documentación dando como resultado una muy difícil modificación de sus componentes que en la mayoría de los casos dan soporte a funciones de misión crítica dentro de una organización.

Los sistemas legacy constituyen un activo clave para la organización que hace uso de este, pero como todo activo, debe ser objeto de cuidados, mantenimientos y adecuación continua a las necesidades cambiantes de los negocios y tecnologías.
Un ejemplo clásico de este tipo de sistemas son aquellos implementados hace décadas por grandes empresas (tales como entidades bancarias) que requerían gran soporte para el procesamiento por lotes y COBOL era una buena opción para ejecutar en sus mainframes.

En la literatura, el término "Sistema Legacy o Sistema Heredado" suele definirse como:

"*Los* sistemas legacy *son aplicaciones importantes, imprescindibles para el normal funcionamiento de una organización y que están en producción desde hace varios años*" [Willem-Jan 2006].

Con la evolución de las tecnologías y la aceptación de estándares, el hardware, los sistemas operativos, el software de base de datos y demás herramientas de base se han ido convirtiendo en commodities (piezas de fácil reemplazo) mientras que los sistemas legacy son difícilmente sustituibles. Esto se debe a distintos factores, entre los cuales podemos destacar: su alto costo de desarrollo (por ser software especifico propietario) o por haber requerido mucho tiempo de desarrollo definiendo reglas de negocios y procedimientos propios de las organización. De hecho, en general son sustituidos como consecuencia de una reingeniería de procesos de negocios y no por necesidades de adecuación tecnológica.

2.2. Problemática de los sistemas legacy

Los sistemas informáticos evolucionan a lo largo del tiempo conforme lo hacen tanto los modelos organizaciones como los requerimientos operativos. Y para ello, es necesario contar con herramientas que faciliten la adaptación en respuesta a sus cambios [Good 2002]. En ese sentido, como parte de la plataforma de sistemas se encuentran los sistemas legacy, los cuales en la mayoría de las organizaciones, dan soporte a funciones de misión crítica dentro de la organización.

Este tipo de sistemas, en general, componen el núcleo central de la infraestructura de tecnología de la información dentro de una organización. Según los estudios de [BACH 2006] para la integración de sistemas legacy hay que tomar en consideración principalmente los aspectos enumerados a continuación.

Distribución y heterogeneidad: Cada una de las aplicaciones que integran un sistema pueden estar en distintos equipos, pueden operar sobre diversas arquitecturas compuestas por hardware, sistemas operativos, lenguajes de programación, herramientas para administrar datos, soporte de comunicaciones, etc.

Reuso: En la medida en que las operaciones y servicios, pasen a ser reutilizables desde otros tipos de sistemas, nada impide que un servicio pueda ser utilizado desde diferentes tipos de operaciones, e incluso más de una sola vez dentro de la misma, mediante interfaces de programas bien definidos.

Documentación y estado del sistema: La falta de actualización de la documentación, afecta al estado del sistema, los cuales pueden ser fuentes de problemas y errores.

Impacto de los cambios realizados en los Sistemas Legacy: Hay que tener en cuenta un mecanismo de administración de versiones y notificación de nuevas funcionalidades.

Riesgos de alterar el funcionamiento del Sistema Legacy: La pertenencia de un Sistema Legacy no debe afectar el funcionamiento interno de cada tipo de sistema.

Derechos de uso y autenticación de usuarios: La administración de usuarios se torna algo muy complejo ya sea por la autenticación de los usuarios como también la asignación de los derechos para cada uno de los servicios a los que estos acceden.

Seguridad y privacidad de los datos: Existen ambientes operativos que requieren la utilización de transmisión de datos en entornos que pueden ser públicos, privados o mezclas de ambos.

Integridad transaccional: Se deben instrumentar todo tipo de procedimientos para deshacer todas las operaciones ejecutadas satisfactoriamente antes de la ocurrencia de una falla.

Adaptar los Sistemas Legacy: Para integrar Sistemas legacy a un entorno de aplicaciones es necesario adaptar aquellas piezas de software cuyas funcionalidades sean requeridas por la organización.

Administración y gestión de sistemas: Integrar sistemas es una tarea que requiere conformar equipos de trabajo, fijar metas comunes, acordar estándares, construir un equipo de dirección, instrumentar control de calidad a nivel de integración, administración de cambios y demás elementos propios del ciclo de vida de un sistema.

Gran tamaño: Un sistema puede tener un total de millones de líneas de código fuente.
Técnicas y lenguajes de programación obsoletos: Generalmente están codificadas en lenguajes Assembler, COBOL, C++, Visual Basic 6.0, y Power Builder.

Plataformas de hardware obsoletas: Dan soporte a estos sistemas, factor que dificulta el poder hacer mantenimiento a estas plataformas.

El mantenimiento: Es difícil y tienen un alto costo, debido principalmente a la falta de documentación y detalles de funcionamiento de estos tipos de sistemas.

Adición de nuevas funcionalidades: Es una tarea muy difícil, muchas veces imposible.

Falta de interfaces bien definidas: Para acceso de datos y funcionalidades de los sistemas, es uno de los principales obstáculos para la integración con otros tipos sistemas dentro de una organización.

Continuando con las definiciones de los problemas antes mencionados, los sistemas legacy son aplicaciones monolíticas que mezclan funciones de acceso a los datos, funciones de lógica de negocio y funciones de interfaces de usuario [Good 2002].

Capítulo 3 - Estado del arte de migración de sistemas legacy

3.1. Introducción

Como ya hemos mencionado en las secciones previas, los sistemas legacy son un gran activo para muchas organizaciones y el tiempo de vida de estos es muy variable e incluso en ocasiones pueden ser utilizados por décadas. El paso del tiempo y la evolución implica que estos sistemas tengan que ir modificando, agregando e incluso eliminando funcionalidades, desestabilizando el sistema inicial con una enorme cantidad de lógica y reglas de negocios. Según [Sommerville 2005] esto provoca malas prácticas en el mantenimiento, llegando al punto que haya funcionalidades que no se puedan utilizar en ningún otro punto de la organización más que en el propio código fuente legacy.

Los sistemas legacy suelen desempeñar papeles críticos en las tareas que desarrollan y el miedo a que dejen de hacerlo como hasta ahora prolonga al máximo en el tiempo cualquier decisión de reemplazarlos por sistemas modernos. No obstante, siempre se alcanza el momento en el que no son capaces de continuar ofreciendo los mismos servicios por diversas razones, pero principalmente por restricciones tecnológicas que impiden que las actualizaciones sobre esos sistemas funcionen. Llegados a este punto, se debe ofrecer una solución con garantías que nos proporcione todo lo que ya teníamos.
El hecho de mantener toda la lógica y reglas de negocios nos aleja de tomar la decisión de realizar desde cero el desarrollo de un nuevo sistema dado que estas tienen alta probabilidad de perderse o modificarse y nos invita a buscar otras alternativas [Bisbal 1997].
Una solución muy usada [Sneed 2004] para hacer frente a los problemas que trae consigo enfrentarse a los sistemas legacy es la reingeniería porque reduce los riesgos, los costos de desarrollo y hace que el software sea más fácilmente modificable.

3.2. Reingeniería

Según [Chikofsky 1990] la reingeniería consiste en la revisión, análisis y modificación de un sistema de software existente para reconstruirlo de una nueva forma, y la posterior aplicación de la nueva forma.

El proceso típicamente comprende una combinación de otros procesos tales como la ingeniería inversa, la reestructuración, la traducción y la ingeniería directa (ilustración 1).

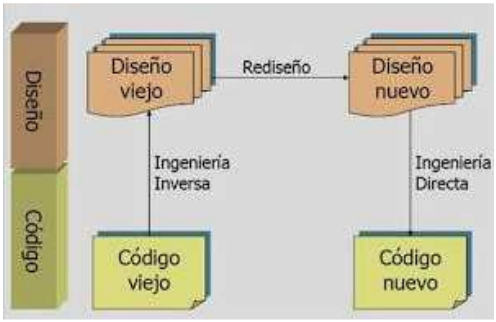


Ilustración 1: (Esquema de reingeniería)

El objetivo es entender el software existente (especificación, diseño, implementación) y luego volver a ponerlo en práctica para mejorar la funcionalidad, el rendimiento o la implementación del sistema.
Además, trata de mantener la funcionalidad existente y dejar todo preparado para las funcionalidades que se puedan añadir a futuro [Rosenberg 1996].

3.3. Ingeniería inversa

Esta técnica se denomina así dado que avanza en dirección opuesta a las tareas habituales de ingeniería, que consisten en utilizar datos técnicos para elaborar un producto determinado.

Es la primera de las tres etapas anteriormente mencionadas y se basa en obtener un conjunto de especificaciones a partir del sistema legacy que forme la base para construir una nueva implementación del sistema.

Los estudios de [Chikofsky 1990] indican que el resultado de este método es una nueva representación del sistema a un nivel de abstracción mayor, reduce la complejidad del sistema a cambio de cierta pérdida de información.

3.4. Reestructuración

En esta segunda etapa se toma como entrada las representaciones del sistema legacy obtenidas como resultado en la etapa anterior y se realizan trasformaciones al mismo nivel de abstracción para obtener nuevas representaciones que mejoran de algún modo las propiedades del sistema.
Estas transformaciones [Chikofsky 1990] deben mantener el comportamiento externo del sistema de origen, con ello se consiguen preservar la lógica y las reglas del negocio embebidas en el sistema.

3.5. Ingeniería directa

Esta es la tercera y última etapa, que al igual que la etapa anterior, toma como entrada el resultado de la etapa previa. En este caso, el objetivo es utilizar los modelos de

representación de la etapa previa para alterar o reconstruir el sistema con la finalidad de mejorar su calidad global.
En la mayoría de los casos [Chikofsky 1990] el software sometido a reingeniería vuelve a implementar la función del sistema existente y también añade nuevas funciones y mejoras.

3.6. Modelo en herradura

El conjunto de las etapas anteriormente mencionadas se representa en la reingeniería mediante el modelo de herradura (ilustración 2) [Kazman 1998].
La riqueza de este modelo está en sus niveles de abstracción. Este modelo refleja los niveles de abstracción de una manera muy sencilla junto con los distintos niveles de abstracción por los que vamos pasando en cada una de las etapas previamente explicadas. En un simple vistazo se puede confirmar que se alcanza el máximo nivel de abstracción en el término de la primera etapa, que en la segunda se mantiene y es en la tercera y última etapa donde disminuimos al máximo el nivel de abstracción.
A pesar de que este modelo se aplica desde hace bastante tiempo [Cuadrado 2012] tiene ciertos inconvenientes que aumentan el índice de fracasos al utilizarlo. Se puede mencionar la falta de formalización y estandarización del proceso completo de reingeniería o su falta de automatización, lo que plantea buscar alternativas que soluciones estos problemas, siendo el desarrollo dirigido por modelos una buena opción para solventar los problemas mencionados.

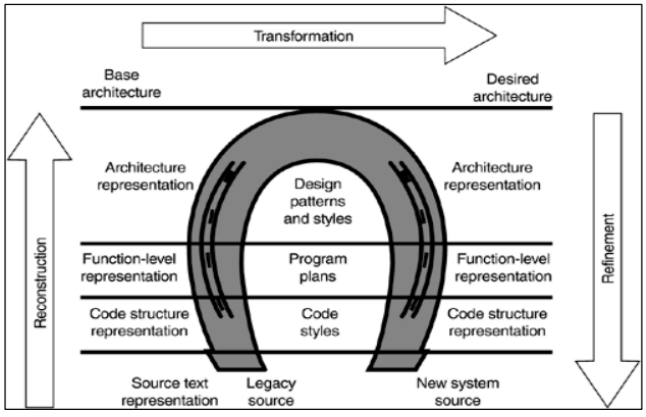


Ilustración 2: (Modelo de herradura)

3.7. Desarrollo dirigido por modelos

Según [Atkinson 2003], el desarrollo dirigido por modelos (Model Driven Development) nos ofrece una representación de sistemas de información mediante la generación de modelos que son más pequeños y más abstractos que los sistemas procesados.

Como ya predice su nombre, los modelos forman el pilar básico de este desarrollo. Una definición para este tipo de modelos sería la siguiente.

"*El diseño estructural de un sistema complejo*" [Atkinson 2003].

Recuperando la importancia de los modelos en este esquema, se debe mencionar que estos nunca superarán en complejidad ni dimensión el sistema de información al que representan, con lo que facilitarán su manipulación y comprensión.

Estos modelos serán el eje que permitirá las transformaciones necesarias para conseguir una implementación que implícitamente debe ser semiautomática en el desarrollo dirigido por modelos.

3.8. Arquitectura dirigida por modelos

El desarrollo dirigido por modelos necesita de una serie de reglas y están especificadas en la arquitectura dirigida por modelos (Model Driven Architecture) [Pastor 2007]. Estas reglas las define como estándar el Object Management Group con lo que proporciona un conjunto de guías para estructurar especificaciones expresadas con modelos.

Define tres tipos de modelos en función del nivel de abstracción (ilustración 3):

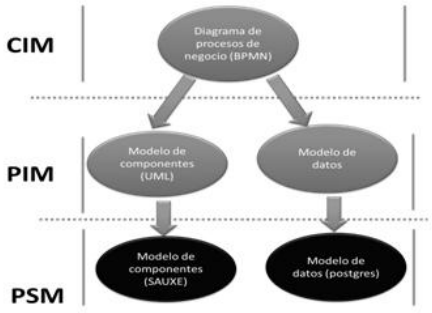


Ilustración 3: (Esquema de arquitectura)

CIM (Computation Independent Model): Proporciona una visión del sistema a un alto nivel de abstracción y es independiente del punto de vista de la computación. Se ocupa del modelado del negocio y de los requisitos.

PIM (Plataform Independent Model): Ofrece una visión del sistema a un nivel de abstracción intermedio. Se centra en el análisis y diseño, y es independiente de la plataforma tecnológica.

PSM (Pataform Specific Model): Muestra una visión del sistema desde un bajo nivel de abstracción. Trata el diseño dependiente de la plataforma tecnológica. El modelo será adaptado para una implementación tecnológica especifica.

3.9. Modernización dirigida por la arquitectura

La modernización dirigida por la arquitectura (Architecture Driven Modernization) es el nombre de la iniciativa de la Object Management Group (OMG) en relación con la construcción y la promoción de normas que pueden aplicarse para modernizar los sistemas legacy. El objetivo de esta iniciativa es ofrecer un conjunto de estándares para

la modernización de sistemas legacy y la definición de tareas como el análisis del código y la comprensión, y la transformación de los modelos que representan los sistemas legacy (OMG 2006).
Enlazando con el punto anterior, este modelo sigue el mismo enfoque mencionado previamente, más concretamente en el uso de las transformaciones entre distintos modelos según el nivel de abstracción para obtener una arquitectura deseada en función del sistema de información legacy.
Luego, ¿ se puede aplicar el mismo esquema de transformación anterior? No exactamente, sino que se realiza una modificación del mismo combinada con el modelo en herradura (ilustración 3).
Con esto se consigue que convivan las fases de ingeniería inversa, restructuración e ingeniería directa con los tres tipo de modelos de la arquitectura dirigida por modelos (PSM, PIM y CIM)

3.10. Metamodelo KDM

El metamodelo Knowledge Discovery Metamodel (KDM) es una especificación del Object Management Group diseñada para ser la base de la iniciativa de la modernización dirigida por la arquitectura del OMG [Castillo 2011].
KDM provee una representación común intermedia para los sistema de software existentes y sus entornos operativos.
Es una representación independiente de la plataforma y el lenguaje de programación.
KDM es un metamodelo Meta Object Facility (MOF), que define un formato de intercambio de archivos en formato XMI (XML Metadata Interchange) entre herramientas que trabajan con software existente, y define una API sobre la que poder construir herramientas de nueva generación de modernización y aseguramiento del software [Castillo 2011].

El propósito principal [Castillo 2011] por el que apareció este metamodelo era proporcionar una amplia vista de la estructura de la aplicación y los datos, pero sin llegar a representar ningún tipo de software. Las principales características que definen KDM son:

- Representa artefactos de software legacy como entidades, relacione y atributos.
- Incluye artefactos externos con los que interactúan artefactos de software.
- Soporte una gran variedad de plataformas y lenguajes.
- Describe la estructura física y lógica de los sistemas legacy.
- Puede agregar o modificar, por ejemplo, refactorizar la estructura física del sistema.

En cuanto a la organización que sigue KDM se pueden distinguir cuatro capas que a su vez se agrupan en distintos paquetes [Castillo 2011] y que, visualmente, se pueden distinguir en la ilustración 4.

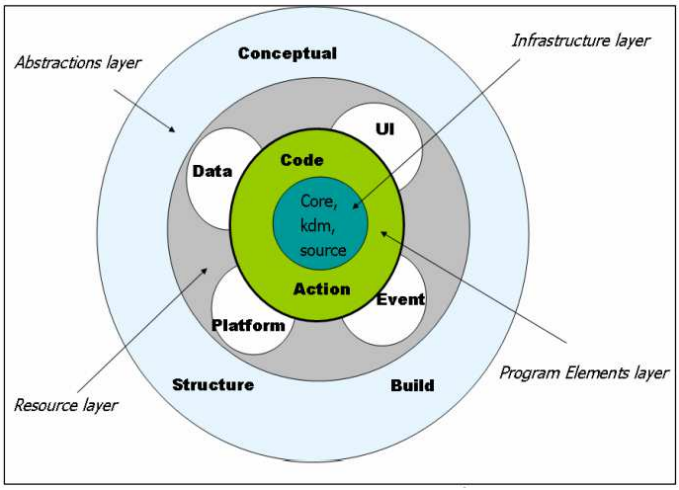


Ilustración 4: (Organización KDM)

Capa de infraestructura: La forman los paquetes Core, KDM y Source, que proveen un pequeño núcleo común para los demás paquetes, el inventario de los modelos de artefactos de los sistemas existentes con total trazabilidad entre los elementos del metamodelo en forma de enlaces al código fuente de los artefactos, así como el mecanismo uniforme de extensibilidad.

Capa de elementos de programa: Este capa está compuesta por los paquete Code y Action.
El paquete Code representa los elementos de programación determinados por los lenguajes de programación, por ejemplo los tipo de datos, procedimientos, clases, métodos, variables, etc. Este paquete provee un nivel más elevado de detalle, y se integra con las vistas arquitecturalmente significativas del sistema de software. La representación de tipo de datos en KDM está alineada
con el estándar ISO/IEC 11404.
El paquete Action captura los elementos de comportamiento de bajo nivel de las aplicaciones, incluyendo los flujos detallados de control y datos entre declaraciones.

Capa de recursos: Esta capa representa el entorno operacional de los sistema de software existentes.
Está relacionada con el área de la Enterprise Application Integration especificación de OMG (EAI) y la componen los paquetes de Platform, UI, Event y Data.
El paquete Platform representa el entorno operativo del software, relacionado con el sistema operante, el middleware, etc. Incluyendo el flujo de control entre componentes tal como son determinados por la plataforma runtime.
El paquete UI representa el conocimiento relacionado con las interfaces de usuario de los sistemas de software existentes.
El paquete Event representa el conocimiento relacionado con los eventos y comportamientos de las transiciones de estado de los sistemas de software existentes.
El paquete Data representa los artefactos relacionados con los datos persistentes, tales como ficheros indexados, bases de datos y otros tipos de almacenes de datos. Estos

activos son clave para el software empresarial, ya que representan los metadatos empresariales.

Capa de abstracciones: La capa de abstracciones representa las abstracciones de dominio y aplicación. La forman los paquetes Conceptual, Structure y Build.
El paquete Conceptual representa las reglas de negocio, en la medida en que esta información pueda ser extraída de aplicaciones existentes.
El paquete Structure describe los elementos del metamodelo para representar la organización lógica del sistema de software, capas y componentes.
El paquete Build representa los artefactos finales relativos al LIS.

Capítulo 4 - Arquitecturas orientada a servicios

4.1. Introducción a SOA

Los sistemas informáticos han evolucionado exponencialmente, lo cual se ha transmitido en un crecimiento de la complejidad del software de las empresas. Las tecnologías tradicionales han alcanzado su límite de capacidad y en las empresas cada vez se pide una respuesta más rápida a los requerimientos de negocio, reducciones de costes y a la integración de nuevos patrones y de nuevos clientes.
Las organizaciones de IT han buscado diferentes implementaciones antes de llegar a desarrollar SOA y los problemas básicos con lo que se han encontrado y con los que hemos de contar cuando queramos hacer implementaciones de SOA son [Holley 2004]:

Complejidad: Los entornos son complejos. Las obligaciones de cumplir los presupuestos y la eficiencia de operación hacen necesario una reutilización de los sistemas, más que un reemplazo de éstos. El fácil y barato acceso a Internet ha hecho posible la constante creación de nuevos modelos de negocio que se han de evaluar para seguir el ritmo de la competencia. EL crecimiento por fusión y adquisición están a la orden del día y por ello es necesario que las organizaciones IT, aplicaciones e infraestructuras se integren y absorban. A esta complejidad se le suman las soluciones punto a punto que lo hacen todavía más complicado, con lo que es necesario desarrollar un entorno heterogéneo en el que se utilicen todo tipo de hardware, sistemas operativos, lenguajes, tipos de datos, etc.

Programación redundante y no reutilizable: las aplicaciones igual que las empresas se van desarrollando a partir de fusiones y adquisiciones, lo cual hace que se tenga que trabajar con aplicaciones redundantes o aplicaciones con funciones que no son fácilmente reutilizables. Si cada unidad de negocio se desarrolla independientemente, a la hora de actualizar o de introducir los nuevos productos y servicios, la redundancia hace que aumenten los costes y el tiempo necesario para introducir los cambios en cada parte afectada.

Interfaces múltiples: Si tenemos que interconectar N sistemas existentes necesitamos N(N-1) interfaces para hacer posible esta realización, con lo que si queremos introducir un nuevo sistema deberá crear documentación, testear y mantener 2N nuevos interfaces. Hasta ahora se han desarrollado diferentes modelos de programación que han ido intentando perfeccionar las barreras comentadas.

En la actualidad, los Web Services han roto toda barrera para interconectar aplicaciones de una manera basada en un modelo orientado a objeto neutral. Los Web Services son otro de los pilares de SOA y se definen según [Holley 2004] como un conjunto de protocolos que se usan para que los servicios sean publicados, descubiertos y usados en una tecnología neutra y de una forma estándar. Podemos concluir que los modelos estándares de SOA se basan en J2EE y Web Services. SOA es un modelo de programación y de arquitectura a la vez. Nos permite diseñar sistemas software que proveen de servicios a otras aplicaciones a través de interfaces publicados y descubiertos. Los servicios pueden ser invocados sobre una red. Si se utilizan Web Services para implementar una SOA, se crea un camino para construir aplicaciones dentro de un modelo de programación más robusto y flexible. Se reduce el desarrollo, los costes y el riesgo de implementación.

4.2. Servicio SOA

El concepto más importante de SOA es el de SERVICIO. Según el Component Based Development and Integration (CBDI) Forum su definición sería: "*Vehículo a partir del cual las necesidades de los clientes son satisfechas de acuerdo con el contrato negociado (implícito o explícito), el cual incluye el acuerdo de servicio (Service Agreement), la función ofrecida etc.*".

Otra definición posible que podemos encontrar, es la que nos da el World Wide Web Consortium (W3C): "*Componente capaz de realizar una tarea*" [Wilkes 2004].

Se puede definir un servicio como un paquete funcional de software al cual se puede acceder a través de una infraestructura de red. Los servicios son autónomos, autocontenidos y uno no puede tener control, ni autoridad sobre ellos. Un ejemplo de servicio puede ser, dentro de un entorno de negocio, una transacción simple (como obtener la cuota de stock, conseguir las direcciones de los clientes, etc.), una transacción de negocio más compleja (calendario de entregas, cobertura de ventas, etc.) o servicios de sistema (autenticación de usuario, mensaje de conexión, etc.).

Las funciones de negocio desde el punto de vista de aplicación no son funciones de sistema, y son atómicas (aunque estén formadas por funciones más pequeñas, se considera como un paquete indivisible). Las transacciones de negocio tienen que implementarse como una composición de funciones de bajo nivel transparentes al que las llama. Por último los servicios de sistema son funciones genéricas que pueden ser abstraídas fuera de la plataforma particular, como Microsoft Windows o Linux.

La riqueza funcional de las aplicaciones nos la da el nivel de granularidad [Holley 2004]. La granularidad es la capacidad de descomponer las aplicaciones de negocio en servicios y tiene implicaciones prácticas, no es sólo un proceso de abstracción. Los servicios pueden ser de baja granularidad o funciones complejas de alta granularidad (fine-grained o coarse-grained respectivamente).

Generalmente, los servicios son funciones coarse-grained ya que suelen ser el resultado de ejecutar varias operaciones fine-grained; por ejemplo, abrir una cuenta está compuesto por identificar al usuario y crear una cuenta nueva etc. Esta es la

manera de desarrollar un entorno de aplicaciones basadas en componentes, donde los servicios se definen como un conjunto de componentes reusables que pueden ser usados para la construcción de nuevas aplicaciones, o para integrar los recursos de software ya existentes.

4.3. Orientación a objetos y CDB

Se pueden encontrar paralelismos entre la Orientación a Servicios (OS), la orientación a objetos (OO) y el desarrollo basado en componentes (CBD) [Wilkes 2004].
Los servicios también representan bloques de construcción naturales que nos permiten organizar capacidades de forma que nos sean familiares.
Como los objetos y los componentes, los servicios son un bloque fundamental que combina información y comportamiento, esconde el trabajo interno para que esté fuera de intrusos y presenta un interfaz simple para el resto del organismo.
Los objetos usan tipos de datos abstractos y datos abstractos. Los servicios tienen un nivel similar de adaptabilidad a través del aspecto o del contexto de orientación.
Se puede organizar también en clases y jerarquías de servicios heredando los comportamientos y pudiendo ser utilizados después de forma única o como jerarquías o colaboraciones.
Por otro lado, mientras los componentes son la mejor forma de implementar servicios, se debe entender que una aplicación correctamente basada en componentes, no necesariamente es una aplicación correctamente orientada a servicios. La clave para comprender esta diferencia radica en ver como una arquitectura orientada a servicios implica una capa adicional de arquitectura (una nueva abstracción) implementada con una granularidad más alta y ubicada más cerca del consumidor de la aplicación (ilustración 5) [Suarez 2006].

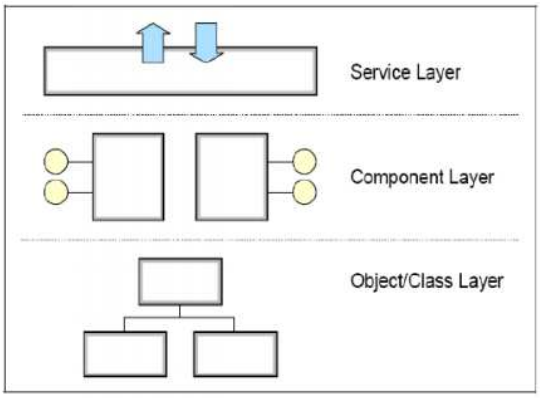


Ilustración 5: (Capas de arquitectura SOA)

Un servicio es una forma de exponer una visión externa de un sistema, con reutilización interna y una composición tradicional basada en el diseño de componentes. En una SOA, un servicio mapea una función identificada durante un proceso de análisis de negocio, dependiendo de la función de negocio de que se trate, la granularidad del mismo puede ser más o menos baja o alta. Los servicios no se diseñan en base a las

entidades de negocio; cada servicio es una unidad que maneja operaciones a través de un conjunto de entidades de negocio.
Como ya hemos destacado, un servicio es una unidad de procesamiento de granularidad gruesa, que consume y produce un conjunto de objetos pasados por valor, implementada sobre una colección de componentes que trabajan en colaboración para entregar la funcionalidad del negocio que el mismo representa; los componentes son de una granularidad más baja que la de los servicios. Mientras un servicio mapea una funcionalidad del negocio, un componente típicamente mapea las entidades del negocio y las reglas que las operan.

4.4. Arquitectura de servicio

Estructura y características de los Servicios

Los servicios son una forma de encapsular componentes/programas reusables (building blocks) para proveer funcionalidad a otros usuarios y a otros servicios. Cuando un servicio provee servicios a otro, al servicio que invoca lo llamaremos consumidor, para distinguirlo del usuario. Con los servicios se interactúa mediante el intercambio de mensajes. Un servicio consiste de 3 elementos [Suarez 2006]:

Contrato: El uso de la funcionalidad que provee un servicio es gobernada por un contrato. Especifica el propósito, la funcionalidad, las restricciones y el modo de uso del servicio. Es definido por el negocio, en términos del negocio.

Implementación: La funcionalidad en sí misma que provee el servicio, puede ser realizada utilizando cualquier tecnología.

Interfaces: Para acceder a la funcionalidad el consumidor necesita “interfacear” con el servicio. Proveen la forma de acceder a la funcionalidad de acuerdo al contrato. Un servicio puede ofrecer múltiples interfaces para permitir su consumo de diferentes maneras.

Las características funcionales de los servicios son:

- Invocación: sincrónica o asincrónica
- Intercambio: uni-direccional, bi-direccional
- Complejidad: referido a la granularidad

Las características no-funcionales:

- Requerimientos de Volúmenes
- Calidad del Servicio
- Tiempo de ejecución del Servicio

Categorización de los Componentes de Servicio

Según la función que cumplen pueden identificarse en primera instancia las siguientes categorías:

- Administración de datos
- Lógica de negocios básica
- Lógica de negocios compuesta
- Interacción con el usuario
- Componentes utilitarios comunes

Principios Comunes de la Orientación a Servicios

- Comparten un contrato formal
- Bajamente acoplados
- Abstraen la lógica que existen debajo (auto contenidos y modulares)
- Interoperables
- Componibles
- Reusables
- Autónomos
- Sin estado
- Descubribles (transparentes a la ubicación)

Pasos a seguir para definir los Servicios

1. Definir el propósito del servicio (orientado al negocio).
2. Determinar la información que debe de manejar el servicio (metadata y schemas).
3. Identificar los potenciales consumidores.
4. Definir los aspectos de niveles de servicio, seguridad y performance que brindará el servicio.
5. Determinar las funciones (métodos) encapsuladas dentro del servicio, es decir el comportamiento interno.
6. Definir las interfaces, los parámetros y el mapeo con las funciones o métodos internos.
7. Definir como deberá ser testeado el servicio (test information, service invocation, validez de los resultados, etc.).
8. Definir la documentación a incorporar.

4.5. Web Services

Es importante no confundir Web Services con SOA. Los estudios de [Holley 2004] resumen que Web Services es una colección de tecnologías, incluyendo XML, Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL) and Universal Description, Discover and Integration (UDDI); los cuales permiten construir soluciones de programación para mensajes específicos y para problemas de integración de aplicaciones.

SOA es una arquitectura, es más que un conjunto particular de tecnologías como los Web Services. SOA debe ser una arquitectura de aplicación dentro de la cual todas las funciones están definidas como servicios independientes con interfaces invocables bien definidos, los cuales pueden ser llamados en secuencias bien definidas para formar los procesos de negocio.
Que todas las funciones se definan como servicios incluye funciones de negocio, transacciones de negocio compuestas de funciones de bajo nivel y funciones de servicio de sistema.
Que todos los servicios sean independientes quiere decir que ellos actúan como cajas negras, las funciones no saben cómo realizan la tarea pero devuelven el resultado esperado.
Que los interfaces sean invocables quiere decir que a nivel de arquitectura es irrelevante si son locales o remotos, tanto da que sistema de conexión o que protocolo se utilice para la invocación, o que estructura de componentes son requeridos para realizar la conexión.
En SOA la clave está en el interfaz, éste define los parámetros requeridos y la naturaleza del resultado. Esto significa que define la naturaleza del servicio y no la tecnología utilizada. Esta función permite realizar dos de los puntos críticos: los servicios son realmente independientes y pueden ser manejados. WS es el estándar apoyado por la industria (Microsoft, IBM, BEA, Oracle, Sun y otros), por empresas de distintos rubros, no tecnológicas (Ford, United Airlines, KPMG, Daimler-hrysler), agrupadas en un comité conocido como Web Services Interoperability, o WS-I. Este organismo tiene por principal objetivo asegurar que los grupos de trabajo que definen las especificaciones sobre WS utilizan estándares adecuados, a la vez que monitoriza el avance de sus trabajos; no define ni desarrolla estándares [Suarez 2006].
WS es un estándar construido a su vez en estándares como:

- WSDL (para describir contratos entre el consumidor y el proveedor de un servicio)
- UDDI (para descubrir servicios)
- SOAP/REST (para invocar servicios)
- XML / HTTP (para la capa de transporte)

4.6. Ventajas y desventajas SOA

Ventajas:

El concepto básico de SOA es el de proveer de un conjunto básico de servicios a los que cada aplicación puede acceder para realizar su función. En esencia, la idea se refiere a escribir el código una vez para después utilizarlo en cualquier sitio. En consecuencia, el resultado es menos código, menos costes y el incremento de estandarizaciones. El Software Quality Assurance (SQA) total decrece desde que hay menos código para documentar y la documentación de empresa ya existe para los servicios SOA. Además, se deben compilar y debuggear menos líneas de código lo cual ahorra tiempo de programación.
Podemos resumir las ventajas más importantes de SOA en [Pérez 2007]:

- Reutilización
- Interoperabilidad
- Escalabilidad
- Flexibilidad
- Eficiencia de coste, al reutilizar infraestructuras ya existentes.

Desventajas:

El primer inconveniente con el que nos encontramos al desplegar SOA es que las arquitecturas existentes no están diseñadas para soportar SOA [Howerton 2007]. Es por ello que una migración hacia SOA necesita una gran planificación, reprogramación y adquisiciones. Muchos negocios no tienen presupuesto suficiente para hacer un cambio de este tipo teniendo una arquitectura funcional.

Otro inconveniente es que SOA no es tan escalable o tan fácil de conseguir como otras soluciones. SOA depende de XML y comunicaciones de red que requieren análisis y serialización continua, lo cual provoca que incremente el tiempo de transacción y que se haya de invertir en hardware adicional para llevar a cabo estas acciones. Por último, las empresas son reacias a "jugárselo todo a una carta", ya que si el servicio falla, todas aplicaciones que lo utilicen fallarán también.

Capítulo 5 - Proceso de migración legacy hacia SOA

5.1 Introducción

Hoy en día, el acelerado avance tecnológico, la evolución de paradigmas y plataformas hacen que la gran mayoría de negocios y organizaciones basen sus procesos en sistemas informáticos, lo que hace imprescindible que estos nuevos procesos convivan con los sistemas legacy.
Esta clase de sistemas al igual que las organizaciones deben evolucionar y adaptarse a nuevos requerimientos. Las soluciones clásicas relacionadas son [Rodríguez 2001], el rediseño completo del sistema en una nueva tecnología, el wrapping del sistema legacy que provee una nueva interfaz para algún componente de éste, lo que permite mayor accesibilidad desde otras aplicaciones y la migración o transformación que mueve al sistema legacy a un nuevo ambiente o plataforma más flexible, reteniendo la funcionalidad y los datos del sistema original.

El objetivo de esta sección es proponer un proceso que nos permita definir un enfoque para implementar una migración de un sistema legacy monolítico (ilustración 6) hacia una arquitectura orientada a servicios (ilustración 7). Este proceso, tendrá como premisa mantener el activo legacy del sistema y tendiendo a mecanismos automatizables disponibilizarlos en arquitecturas orientadas a servicios.

Arquitectura origen: Sistema legacy monolítico clásico donde no se encuentra bien diferenciado la interfaz de usuario, la lógica de negocios y el acceso a datos.

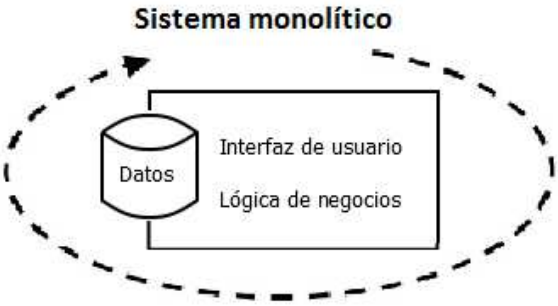


Ilustración 6: (Arquitectura monolítica clásica)

Arquitectura destino: Arquitectura orientada a servicios donde se diferencian bien las responsabilidades de cada capa permitiendo una simple integración con aplicaciones clientes.

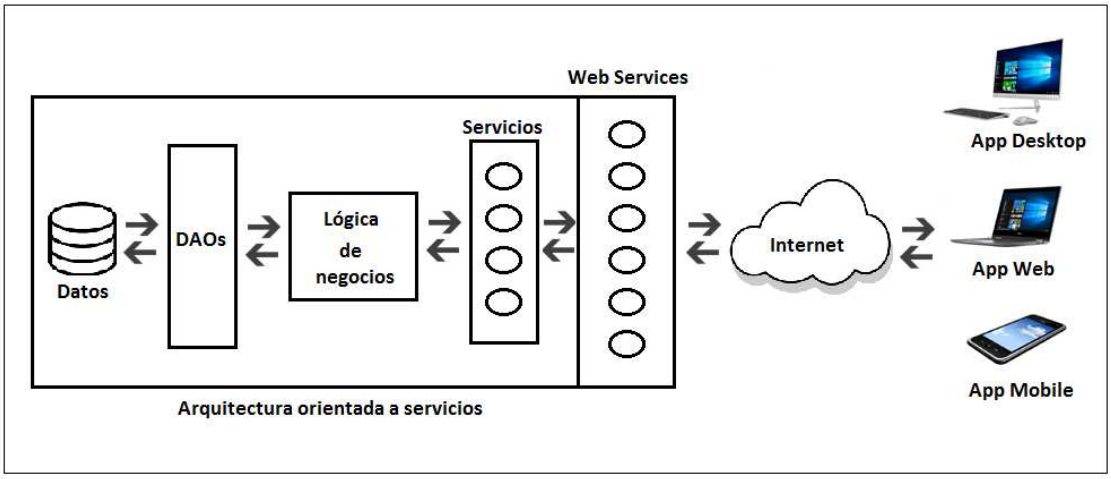


Ilustración 7: (Arquitectura SOA)

Es importante destacar que el proceso perseguirá mecanismos automatizables en cada etapa que sea posible, donde la idea principal gira en torno a inferir una arquitectura SOA base automáticamente del sistema legacy e ir derivando funcionalidades especificas a dicha arquitectura.

El siguiente gráfico (ilustración 8), nos anticipa el modelo sobre el cual el proceso se guiará.

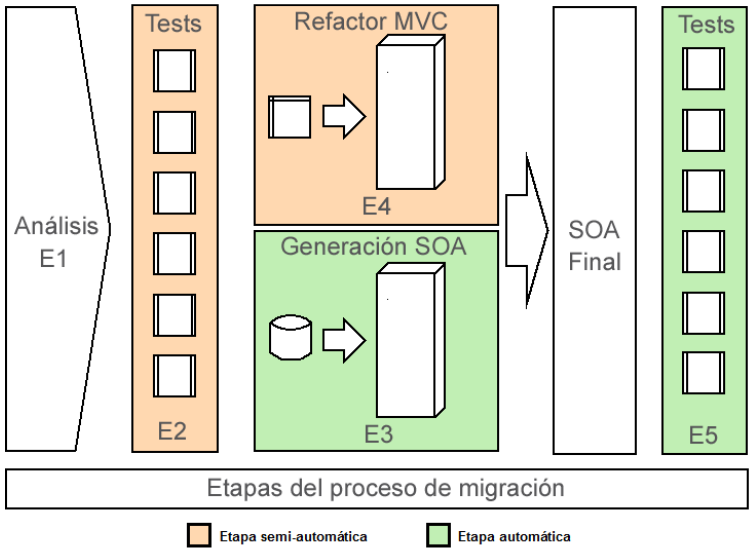


Ilustración 8: (Modelo del proceso de migración)

5.2 Descripción del proceso de migración

El proceso de migración estará compuesto por una serie de etapas claramente identificables. Comenzaremos el proceso realizando un diagnóstico acerca de la factibilidad de la migración del sistema legacy propuesto, una vez confirmado su potencial migratorio es necesario definir una serie de casos de prueba pre migratorios los cuales utilizaremos una vez concluida la migración para validar el proceso. El siguiente paso, es trabajar sobre los orígenes de datos que generalmente son bases de datos relacionales, pero podrían ser otros. Luego, ya con la base de datos preparada para migrar, nos concentraremos en la generación de arquitecturas orientadas a servicios mediante mecanismos automáticos de generación de código. Una vez disponibilizada la arquitectura base SOA, es el momento de trabajar sobre el código fuente legacy con el objetivo de integrar dichos fuentes con la arquitectura previamente generada. Por último, es muy importante contar con mecanismos automatizados para validar si la migración cumple las expectativas, para esto validaremos en base a los casos de uso previamente mencionados.
A continuación, se detalla cada etapa del proceso de migración.

5.3 Análisis inicial sobre la factibilidad de migración del sistema legacy

El primer y fundamental paso en cualquier migración de sistemas legacy es realizar un análisis minucioso acerca de cuan posible será llevar a cabo una migración exitosa [Good 2002], para eso es necesario tener presente los aspectos analizados a continuación.
Se debe tener bien claro qué es lo que se quiere conseguir al migrar el sistema legacy, para ello, se necesita definir las metas y objetivos del negocio y la forma en que la migración logrará alcanzarlos.

Es importante justificar el proyecto, las metas, los objetivos, el alcance y las limitaciones de la migración.
Se requiere tener conocimiento del punto de partida, esto es analizar las características del sistema legacy a migrar. Según los estudios de [Good 2002] principalmente, se debe saber si es un sistema de caja negra (no dispone de código fuente), caja blanca, mixto, cuáles son sus orígenes de datos y tecnologías relacionadas así como también no se deben descuidar cuales son las funciones principales las que luego del proceso se deben validar y mantener correctamente.
Es importante también, realizar un análisis relacionado a la deuda técnica del sistema legacy [Seaman 2011], en la sección 5.12 se profundiza acerca de este concepto.
Otro aspecto fundamental que debemos analizar está referido a qué se pretende exactamente migrar, se migrará el origen de datos, un conjunto de componentes, una serie de algoritmos, una capa de la aplicación o incluso se migrará el sistema legacy integralmente.
Finalmente, analizado todos los puntos antes descriptos es necesario definir también cuáles serán las tecnologías destino sobre las cuales apoyaremos el proceso de migración específico.

5.4 ¿Qué tipo de migración puedo llevar a cabo?

En este punto, se deben revisar las posibles estrategias de migración dada la naturaleza del sistema legacy. Los principales factores de decisión son la calidad del sistema legacy y la disponibilidad de componentes de reemplazo. Para evaluar la calidad del sistema, podemos focalizarnos en los siguientes parámetros [Good 2002]:

- ¿Cuán efectivo es el sistema legacy? Funciona correctamente, es correcto, qué volumen de errores posee, analizar soluciones alternativas y nivel de soporte necesario.
- ¿ Las reglas del negocio son estables o tienden a cambiar frecuentemente? Si las reglas del negocio tienen a cambiar constantemente, se debe poner en duda el proceso de transformación y analizar su reemplazo.

En resumen, la evaluación de calidad nos debe focalizar en saber cuan apta es el sistema legacy en términos comerciales y técnicos.
En concreto los tipos de migración que podemos llevar a delante son [Rodríguez 2001]:

- **Transformar o migrar:** Llevar a cabo un proceso de migración integral para permitir luego agregar funcionalidad, integración y alcance empresarial.
- **Reutilizar:** Es posible aplicar técnicas que nos permitan reutilizar solo ciertas funcionalidades del sistema legacy, en esos casos se utilizan wrappers específicos.
- **Re-escribir:** El activo clave aquí son las reglas de negocio y las estructuras de datos, el problema es la aplicación. Para iniciar la re-escritura, se requiere de la minería de la aplicación, el análisis de lógica del código y estructuras de datos.
- **Reemplazar:** En este caso, se decide abandonar el sistema legacy y adquirir sistemas totalmente nuevos contratados externamente.

Por supuesto, el objetivo que persigue este documento es realizar un proceso de migración integral sobre el sistema legacy.

5.5 Diseño de casos de prueba pre migratorios

Con el objetivo de garantizar el correcto funcionamiento del sistema luego del aplicar el proceso de migración se plantea la necesidad de inferir la sanidad del sistema legacy mediante la recolección de test para su posterior automatización. Si bien existen diferentes tipos de tests [Bertolino 2003], que se pueden representar mediante la pirámide de la automatización del test (Test de Unidad, Test de Servicios y Test de Interfaz de Usuario) (ilustración 9) el proceso se focalizará en la automatización de Test de Interfaces de usuario:

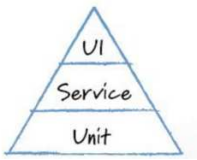


Ilustración 9: (Pirámide de test)

Dado que luego de aplicar el proceso de migración, tanto la funcionalidad como la representación del sistema legacy existente no debe ser alterada por los cambios generados, se plantea la necesidad de diseñar un plan de pruebas en base a la documentación sobre el proyecto y la documentación sobre el software a probar. Este tipo de pruebas se las conoce como test de regresión [Bertolino 2003]. A partir de dicho plan, se entra en detalle diseñando pruebas específicas basándose en la documentación del software a probar. Una vez detalladas las pruebas (especificaciones de casos y de procedimientos), se toma la configuración del software (revisada, para confirmar que se trata de la versión apropiada del programa) que se va a probar para ejecutar sobre ella los casos. A partir de los resultados de salida, se pasa a su evaluación mediante comparación con la salida esperada. A partir de ésta, se pueden realizar dos actividades, la depuración (localización y corrección de defectos) y el análisis de la estadística de errores.
Es intención del proceso de migración automatizar el plan diseñado sobre tests de interfaz de usuario mediante herramientas especializadas.

Las pruebas automatizadas que controlan la aplicación a través de la interfaz de usuario se conocen como pruebas de UI codificadas (CUIT). Estas pruebas incluyen una comprobación funcional de los controles de la interfaz de usuario. Permiten comprobar si toda la aplicación, incluida la interfaz de usuario, funciona correctamente. Las pruebas de UI codificadas son especialmente útiles donde haya una validación o cualquier otra lógica en la interfaz de usuario. También se suelen usar para automatizar una prueba manual existente.

Como se muestra en la ilustración 10, una experiencia típica de desarrollo podría ser aquella donde, inicialmente, el usuario se limite a compilar la aplicación y a hacer clic en los controles de la interfaz de usuario a fin de comprobar que todo funciona correctamente. Después, puede decidir crear una prueba codificada de forma que no sea necesario seguir probando la aplicación manualmente. Dependiendo de la

funcionalidad concreta que se prueba en la aplicación, puede escribir código para una prueba funcional o bien una prueba de integración que puede que incluya o no la realización de pruebas en el nivel de interfaz de usuario.

Si simplemente se desea tener acceso directamente a alguna lógica de negocios, se puede codificar una prueba unitaria. Sin embargo, en algunas circunstancias, puede ser beneficioso incluir pruebas de los diversos controles de UI en la aplicación.

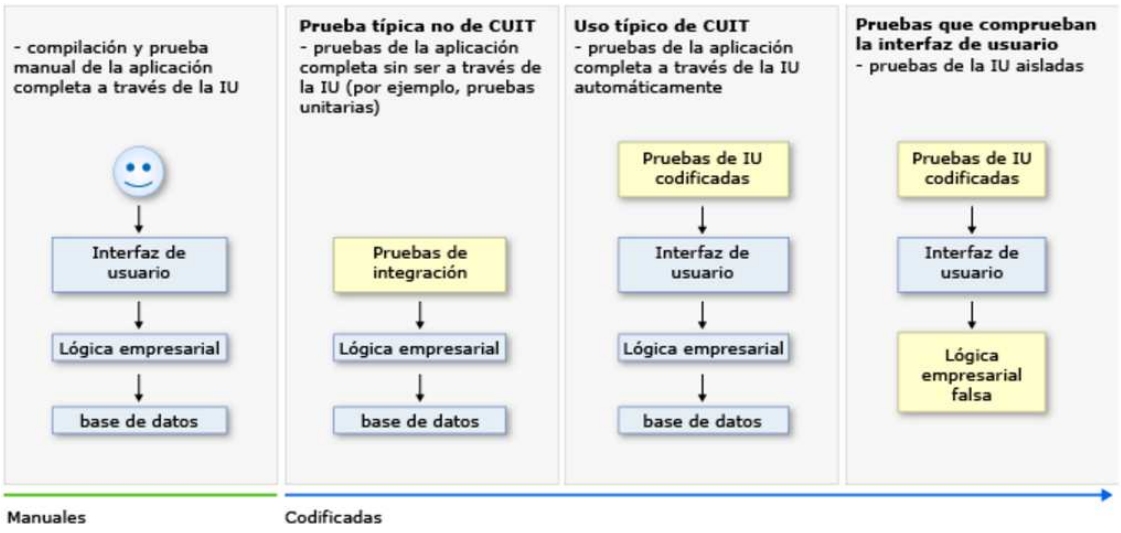


Ilustración 10: (Test manuales vs test codificados de UI)

5.6 Orígenes de datos

Dado que el propósito de este proceso es proveer un mecanismo de migración integral de sistemas legacy, el comienzo concreto del proceso se da trabajando sobre el o los orígenes de datos específicos con los que cuente el sistema legacy. Si bien, un sistema legacy podría tener sus datos almacenados en distinto tipos de tecnologías el alcance de este documento contemplará los orígenes de datos almacenados en bases de datos relacionales (DBMS). Como se menciona anteriormente, el enfoque de este documento es hacia entornos RAD que generalmente utilizan SQL como orígenes de datos. Se propone como trabajo a futuro el desarrollo de un subproceso que permita migrar tecnologías más antiguas (ej archivos) hacia DBMS o nuevas tecnologías no relacionales (NoSQL) [TF-1].

Es importante realizar chequeos sobre el estado actual del DBMS, ya que el siguiente paso en el proceso de migración será necesario partir de un DBMS sano y performante. Es común llevar a delante un plan de mantenimiento del servidor de datos en el cual se deben realizar principalmente los siguientes chequeos [Steve Rees 2013]:

- Comprobar integridad de la base de datos
- Corroborar claves en todas las tablas
- Reorganizar índices

- Reducir el tamaño de la base de datos
- Análisis de logs de errores
- Validar usuarios y roles
- Chequeos lógicos y físicos
- Definir estrategias de Backups

En general, cada tecnología viene acompañada de herramientas que permiten realizar estos y muchos más chequeos sobre el origen de datos a migrar.

5.7 Análisis de requerimientos para SOA

Si bien, como punto de partida en la generación automática de arquitectura SOA se trabajará íntegramente desde los orígenes de datos, es importante realizar un análisis de requerimientos sobre el sistema legacy a migrar para lograr identificar aquellos servicios que no sean triviales y no sean provistos automáticamente.

En caso de que el sistema legacy sea una “caja negra” (no haya acceso al código fuente), se deben analizar las entradas, salidas, y las respuestas obtenidas del sistema.
Por otro lado, la ingeniería inversa es útil para sistemas de “caja blanca” (con visibilidad del código fuente) y permite descomponer el sistema en funciones y datos. El análisis final de los diferentes componentes del sistema legacy permitirá dividir las partes de la aplicación según el tipo de funciones y alinear estas funciones a las metas del negocio, es decir mapear las funcionalidades a los requerimientos del negocio y así poder detectar y definir los servicios candidatos que no se generen automáticamente. También, permitirá descubrir aquellas funcionalidades importantes, que deben conservarse en el sistema migrado.

5.8 Generación automática de arquitectura SOA base desde modelo existente

Hoy en día, gracias al avance en las herramientas de automatización y generación de código contamos con la posibilidad de inferir desde un DBMS distintas alternativas de código autogenerado.
Uno de los objetivos importantes que debe satisfacer este proceso de migración, es ofrecer como resultado una arquitectura backend robusta totalmente orientada a servicios. El primer paso para lograr dicho objetivo es inferir desde la base de datos todo lo posible y autogenerar código de manera automática.
Como se menciona anteriormente, existen diversas herramientas que nos ayudan en la automatización del proceso, con la gran ventaja que no solo nos generan el código necesario en tecnologías modernas y líderes sino que también nos autogeneran arquitecturas n-capas donde podemos distinguir principalmente las siguientes capas:

- Capa de acceso a datos
- Capa de objetos de negocios
- Capa de servicios

Como resultado de finalizar esta etapa del proceso de migración tenemos disponibilizada una arquitectura totalmente orientada a servicios con una

implementación clara, donde se separan las responsabilidades de cada capa y todo de manera automática, prolija y escalable.
En este punto, nuestro sistema legacy ya es capaz de exponer sus datos mediante servicios web implementados automáticamente con tecnologías líderes. El siguiente paso en este camino, es concentrarse en la aplicación ejecutable legacy y definir las pautas y mecanismos de integración con los servicios previamente generados.

5.9 Migración del sistema legacy ejecutable

En esta etapa del proceso trabajaremos sobre el código legacy propiamente dicho. Una de las principales problemáticas de los códigos legacy es que están desarrollados sobre esquemas monolíticos donde sus artefactos están muy fuertemente acoplados. El alcance de este documento se focaliza en las plataformas legacy generadas bajo entornos RAD, muy aceptados y expandidos en los años noventa. Este tipo de entornos de desarrollo rápido no escapaba al problema de los sistemas monolíticos, e incluso los potenciaba (por ejemplo con la incorporación de manejo de eventos). En realidad no se trataba de un problema propio de la tecnología [Mohan 2000], sino del diseño del software pero el entorno orientado a eventos tentaba a simplificar esfuerzos a coste de un diseño no tan efectivo.
El objetivo de esta etapa del proceso de migración es trabajar sobre el código legacy con la idea de aplicar un refactor que nos permita llevarlo al esquema del clásico patrón modelo-vista-controlador (MVC). Logrando dicho objetivo, nuestro código legacy refactorizado estará en condiciones de integrarse al los servicios provistos en el apartado anterior.
Como introducción, el patrón MVC es una propuesta de diseño de software para implementar sistemas donde se requiere el uso de interfaces de usuario. Esta idea, surge hace décadas ante la necesidad de crear software más robusto con un ciclo de vida más adecuado, donde se potencie la facilidad de mantenimiento, reutilización del código y la separación de conceptos. Pero recién en los últimos años es donde su expansión fue más notoria [Burbeck 1992].

En resumen, el objetivo es extraer de las vistas cualquier lógica de negocio y dejar simplemente lógica inherente a la administración de las ventanas, widgets, etc. El controlador, gestionará el flujo de navegación y visualizara la información brindada de la capa SOA (datos, errores, excepciones, etc.).

Enumeración de los pasos del proceso:

- Refactoring hacia MVC
    - Desacoplar interfaces de lógica o acceso a datos.
    - Encapsular lógica de negocios.
    - Encapsular acceso a datos.
- Una vez refactorizado el código legacy integrar con servicios SOA
    - Reemplazar acceso a datos por invocación a servicios: En este punto del proceso ya dispondremos de dos recursos muy importantes, el acceso a datos accesible a través de servicios y el código legacy legible y organizado de modo de poder reemplazar el acceso a datos legacy hardcodeado por las correspondientes llamadas a servicios.

- Trasladar la lógica de negocios al servicio: Por último, y no menos importante en esta etapa es necesario extraer la lógica legacy e insertarla en la lógica de los servicios.

Entre las alternativas para trasladar la lógica de negocios al servicio podemos destacar:

- Migración manual: Dependiendo de la complejidad del código legacy la primer alternativa es migrar manualmente el código hacia la arquitectura SOA.
- Migración asistida: Si el código legacy tuviese muchas líneas de código se podría utilizar herramientas de conversión tecnología origen a tecnología destino para asistir la migración.
- Migración semi-automática: Si la lógica legacy se apoyara mayormente en consultas SQLs es posible extraer el SQL hardcodeado como vistas, funciones o stores procedures y luego generar el servicio automático inferido de estas.
- Migración mixta: En este caso, se emplearían algunas o todas las técnicas juntas para implementar la migración de la lógica presonalizada.

5.10 Validación de casos de prueba post migratorios

La etapa final del proceso de migración será la encargada de validar mediante herramientas automatizadas que ningún circuito del sistema legacy migrado haya sido alterado. Para llevar a cabo esta tarea se debe hacer uso del plan de pruebas definido en el inicio del proceso, allí fue dónde se definió qué probar y cuáles deben ser los resultados esperados. Como se definió desde un principio el alcance del proceso se limitará a trabajar sobre los test de interfaces de usuario, para ello se utilizarán herramientas automatizadas para tal fin.

5.11 Escenarios de automatización

Tal como lo describe el proceso de migración, es necesario ir avanzando etapa por etapa de modo claro y consistente para poder llegar a una migración exitosa. Afortunadamente, en la actualidad existen diversas herramientas que nos brindan un gran soporte de ayuda y automatización de sub procesos. A continuación se enumeran algunos de los escenarios en los cuales podemos utilizar este tipo de herramientas:

- Herramientas de análisis de complejidad de aplicaciones existentes.
- Herramientas de generación de código automático.
- Herramientas de traducción de código.
- Herramientas de ayuda en refactors.
- Productos para diseñar arquitecturas orientadas a servicios.
- Estándares para facilitar integración.

Apoyar nuestro proceso migratorio en este tipo de herramientas, nos ofrecerá las siguientes ventajas:

- Ahorrar tiempos y costos, reduciendo la intervención manual.
- Reducir riesgos al utilizar herramientas probadas y garantizadas.
- Mantener la integridad de la aplicación legacy por medio de criterios de aceptación basados en casos existentes de prueba.
- Preservar el valor de las aplicaciones legacy transponiendo el código existente a una nueva plataforma y asegurando que la conversión no agregue nuevos errores de programa.

Además, el correcto uso de este tipo de herramientas comprueban que la solución podrá responder a las necesidades futuras dirigiéndonos hacia plataformas abiertas y mejorando la calidad del código y estructuras de manera que la aplicación pueda adaptarse a los cambiantes requerimientos de las empresas.

5.12 Problemas y enfoques inherentes al proceso

Por supuesto, no todos los escenarios del proceso de migración pueden ser resueltos de manera automática, si bien las herramientas disponibles son de gran ayuda, hay situaciones las cuales es importante tener en cuenta y preparar planes de soluciones.

Problemáticas a tener en cuenta:

- La aplicación legacy es monolítica. La capa de presentación, el almacenamiento persistente y la lógica de negocios están entrelazadas.
    - Enfoque: La generación automática de la arquitectura orientada a servicios inferida desde los orígenes de datos nos proveen una clara separación de responsabilidades, pero quedan dos puntos importantes para completar la arquitectura. Uno es la necesidad de extraer y encapsular la lógica de negocios del sistema legacy y la otra es la propia integración del sistema legacy hacia la arquitectura orientada a servicios. Es aquí donde se plantea la necesidad de aplicar un refactor en búsqueda de satisfacer los requerimientos antes descriptos, al desacoplar interfaces de usuario de lógica de negocios y acceso a datos sobre el sistema legacy este ofrecerá mecanismos de integración con la arquitectura orientada a servicios resultante de la generación automática.

- La migración implica la transformación de varias aplicaciones que poseen interfaces de comunicación e integración rígidas e incompatibles entre sí, y posiblemente escritas en diferentes lenguajes, en diferentes momentos, y por personas que no se comunicaron entre ellas.
    - Enfoque: Una de las grandes ventajas de las arquitecturas modernas es la habilidad de integrar diversidad aplicaciones una vez que la migración inicial fue llevada a cabo. Según [Good 2002], el enfoque de desarrollo evolutivo adoptado por la mayoría de las empresas quiere decir que la

migración se dará en conjunto con aplicaciones, datos e infraestructura existentes. Las aplicaciones migradas tendrán que coexistir e operar conjuntamente con esas aplicaciones.

- El código es el único lugar en que las reglas de la empresa están documentadas, las reglas están dispersas en el código y no hay nadie que comprenda la aplicación.
    - Enfoque: Realizar una minería en la aplicación podría separar código dependiendo de su responsabilidad (al igual que la solución de aplicación monolítica) una vez encapsulados (aplicando el refactor con el objeto de separar las responsabilidades por capas de aplicación) los fragmentos de código que representan lógica de negocios es mucho más simple interpretar y dar paso a la traducción o migración a otras arquitecturas.

- Deuda técnica: "*La deuda técnica es una metáfora de los artefactos inmaduros, incompletos o inadecuados en el ciclo de vida del desarrollo de software que causan mayores costos y menor calidad a largo plazo*" [Seaman 2011].

    - Enfoque: Actualmente, existen diversas herramientas de inspección automatizadas que permiten evaluar segmentos de código en búsqueda de errores conceptuales graves que conforman la deuda técnica del sistema. En el apéndice de este documento [Herramienta de análisis de deuda técnica] se presenta un ejemplo sobre la herramienta SonarQube. En casos extremos, puede llegar a ser necesario dejar la migración de lado y solo extraer las reglas del negocio a fines de reescribir el sistema por completo. En el apéndice

Capítulo 6 - Caso de estudio

Implementación del proceso de migración mediante tecnologías específicas

6.1 Introducción

El objetivo de este capítulo es llevar a cabo la implementación del proceso de migración propuesto en el capítulo anterior mediante tecnologías específicas. Si bien, el proceso de migración podría ser aplicado a cualquier tecnología origen y destino, en este caso aplicaremos el proceso desde tecnologías RAD Delphi (App desktop) y SQL Server (Base de datos) hacia Web Servicies SOA Microsoft .Net.

Arquitectura origen (ilustración 11): Sistema legacy monolítico implementado en Delphi 6 conectado directamente a la base de datos vía el motor Borland Database Engine (BDE).

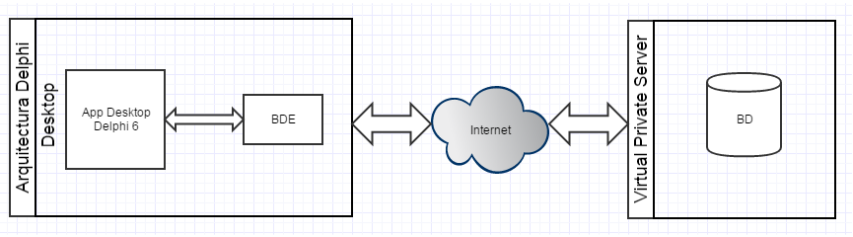


Ilustración 11: (Arquitectura monolítica a migrar)

Arquitectura destino (ilustración 12): Arquitectura orientada a servicios a implementar con Microsoft .Net (Backend) y Bordland Delphi XE7 (App frontend Inmosoft).

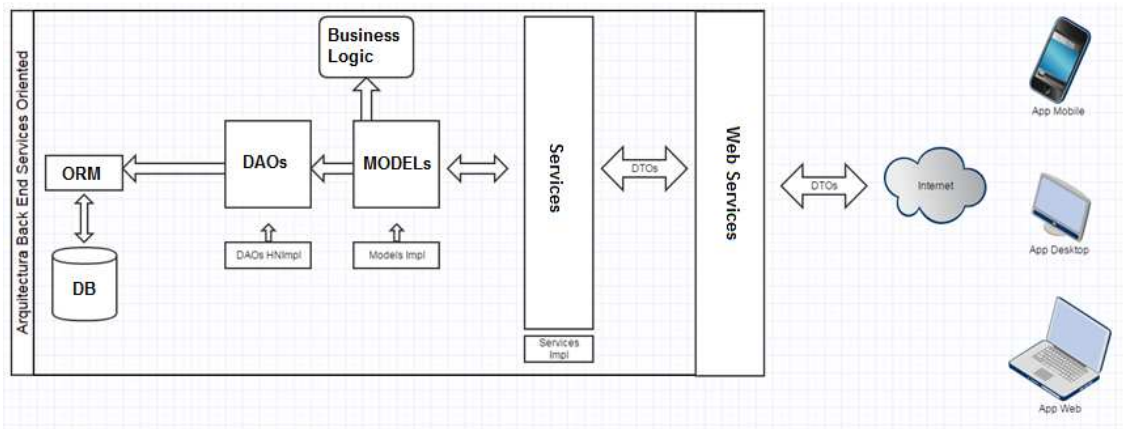


Ilustración 12: (Arquitectura destino post migración)

Como expresa parte de la motivación y objetivos de la presente tesis, se aplicará el proceso de migración a modo de prueba de concepto sobre Inmosoft. Inmosoft es un sistema legacy implementado bajo tecnologías RAD y SQL, en concreto Borland Delphi 6 y SQL Server 2000 [Inmosoft], es intención de este capítulo aplicar el proceso de migración sobre un módulo acotado del sistema Inmosoft para ejemplificar los aportes del proceso (ilustración 8).

6.2 E1 - Análisis del sistema legacy

Si bien a modo didáctico para la presente tesis se trabajará sobre un módulo acotado de Inmosoft, la idea es aplicar el proceso a modo incremental e iterativo procurando avanzar en la migración de un modo simple, claro y reduciendo al mínimo el margen de error. Para llevar a delante dicho plan, se sugiere trabajar sobre un mecanismo de releases periódicos que nos permita concentrarnos en un módulo acotado, migrarlo, testarlo, liberarlo y una vez aprobada la migración avanzar en los siguientes módulos del sistema.
Este tipo de proceso migratorio, podría además implementarse en paralelo, dependiendo del tamaño del equipo abocado a la migración del sistema legacy. Queda fuera del alcance de esta tesis el estudio de las posibles técnicas aplicables para

optimizar la migración paralela sobre grandes equipos de trabajo, se propone dicho estudio como trabajo a futuro relacionado [TF-4].

¿Por qué migrar el sistema legacy?

Es importante tener bien claro el por qué es necesario implementar el proceso migratorio sobre el sistema legacy en cuestión, en este caso como se expresa en parte de la motivación del presente trabajo la finalidad principal es permitir la integración de otras aplicaciones así como también extender funcionalidades que son imposible o muy difíciles de alcanzar con tecnología obsoletas. Si se lograse una migración exitosa hacia tecnologías modernas orientadas a servicios el sistema quedaría en condiciones de integrarse con aplicaciones modernas (app móviles o web) pero manteniendo el activo legacy encapsulado en capas de negocios. Por otro lado, es también importante considerar los recursos humanos relacionados al proyecto, con el tiempo, los conocimientos e interés sobre tecnologías legacy se van perdiendo y es un riesgo que no se puede dejar de considerar al optar por migrar hacia nuevas tecnologías.

Comenzando concretamente con la implementación del proceso migratorio (ilustración 13), el primer y fundamental paso es analizar el potencial migratorio del sistema legacy a migrar. En este caso (Inmosoft) nuestro punto de partida es un sistema de caja blanca del cual disponemos de todos los recursos necesarios para implementar ingeniería inversa directamente sobre el código legacy. Los recursos básicos y fundamentales disponibles son el código fuente (Borland Delphi 6), la base de datos (SQL Server 2000) y un gran conocimiento del dominio de la aplicación.

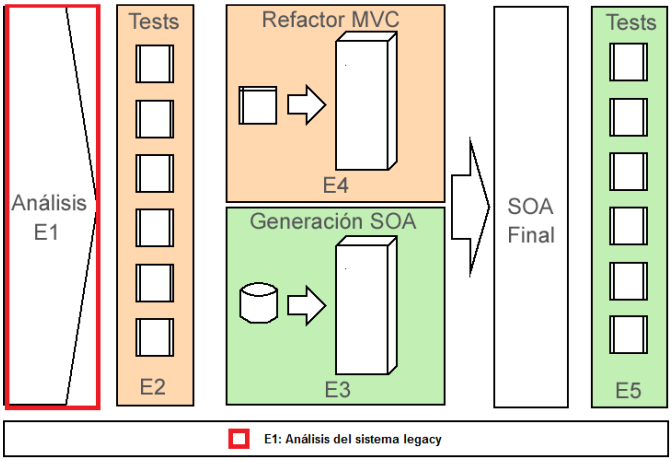


Ilustración 13: (E1: Etapa de análisis)

Basándonos en el manual de usuario de la aplicación, el aportes de expertos, el análisis de casos de usos, el análisis de código fuente u otros el punto de partida es identificar los servicios candidatos que requiere el módulo a migrar. Una vez identificados dichos servicios, en necesario clasificarlos dependiendo de si es un servicio genérico o uno específico personalizado (Etapa 3-a detallada a continuación).

Tecnología origen y destino

Como se menciona en el proceso teórico de migración, en este punto debemos definir cuáles serán las tecnologías destino sobre las cuales apoyaremos el proceso migratorio. En las siguientes tablas, resumimos las tecnologías origen (Tabla 2) y destino (Tabla 3) para nuestro proceso de migración.

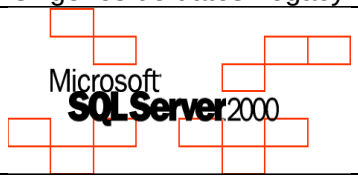
Orígenes de datos Legacy	Entorno de desarrollo Legacy
	
SQL Server 2000	Delphi 6 (2001)

Tabla 2 : Tecnologías pre migración

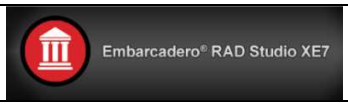
Orígenes de datos destino	Entorno de desarrollo destino	Entorno de desarrollo SOA
		
SQL Server 2012	Delphi XE7 (2014)	Visual Studio 2012

Tabla 3 : Tecnologías post migración

En la elección de las tecnologías destino antes descriptas se priorizó:

- Compatibilidad hacia atrás con código legacy y tecnologías relacionadas.
- Compatibilidad con estándares orientados a servicios.
- Compatibilidad con herramientas de automatización utilizadas en el proceso.
- Robustez y escalabilidad de los entornos de desarrollo.
- Experiencia y conocimientos del desarrollador.
- Minimizar incompatibilidades entre plataformas.

6.3 E2 - Diseño de casos de prueba

Siguiendo el modelo del proceso de migración, se implementará la segunda etapa (ilustración 14) sobre el caso de prueba propuesto.

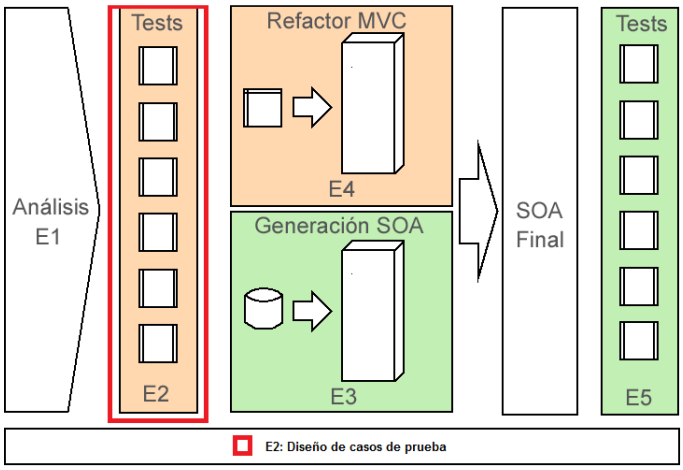


Ilustración 14: (E2: Diseño de casos de prueba)

Ejemplificaremos la creación de los casos de prueba sobre interfaz de usuario con el fin de automatizar los escenarios de tests. Actualmente, existen diversas herramientas que nos permiten automatizar test UI, entre las cuales podemos destacar las siguientes:

- **Ascentialtest:** Proporciona entornos para la planificación de pruebas, gestión de datos de prueba, desarrollo de pruebas manuales y automatizadas, ejecución de pruebas, seguimiento de defectos e informes.
  Web Site: www.zeenyx.com
  Ultimo acceso: Junio 2017.

- **AutoIt:** Lenguaje de scripting diseñado para automatizar la GUI de Windows y scripting general.
  Web Site: www.autoitscript.com
  Ultimo acceso: Junio 2017.

- **Oracle Application Testing Suite:** Con Application Testing Suite, se puede implementar aplicaciones web y servicios web en menos tiempo mientras maximiza la eficiencia de su equipo de pruebas.
  Web Site: http://www.oracle.com/technetwork/oem/app-test/index.html
  Ultimo acceso: Junio 2017.

- **Robot Framework:** Robot Framework es un framework genérico de automatización de pruebas para las pruebas de aceptación y el desarrollo basado en pruebas de aceptación (ATDD).
  Web Site: www.robotframework.org
  Ultimo acceso: Junio 2017.

- **Selenium:** Selenium es un entorno de pruebas de software para aplicaciones basadas en la web.
  Web Site: www.seleniumhq.org
  Ultimo acceso: Junio 2017.

- **Visual Studio:** Microsoft Visual Studio es un entorno de desarrollo multipropósito integrado (IDE) para sistemas operativos Windows. Visual Studio provee un entorno de automatización de pruebas de interfaz de usuario robusto y confiable.
  Web Site: www.visualstudio.com
  Ultimo acceso: Junio 2017.

En nuestro caso, trabajaremos con Visual Studio que nos provee un entorno muy eficiente para llevar a cabo esta etapa del proceso de migración.

A modo didáctico para la presente tesis, realizaremos unos test de UI básicos sobre el circuito de Inmosoft del módulo de personas con la finalidad de que una vez implementada la migración sobre el módulo podamos ejecutar tests de UI automatizados para garantizar que no se alteró la funcionalidad del sistema legacy.

Apoyándonos en la documentación del sistema, expertos del dominio, usuarios avanzados o directamente el código legacy es necesario inferir las funcionalidades existente que se deben respetar mediante el proceso de migración.

La interfaz gráfica de usuario que se utilizará como ejemplo para aplicar el proceso de pruebas basadas en automatización UI es la vista simple mostrada en (ilustración 15).

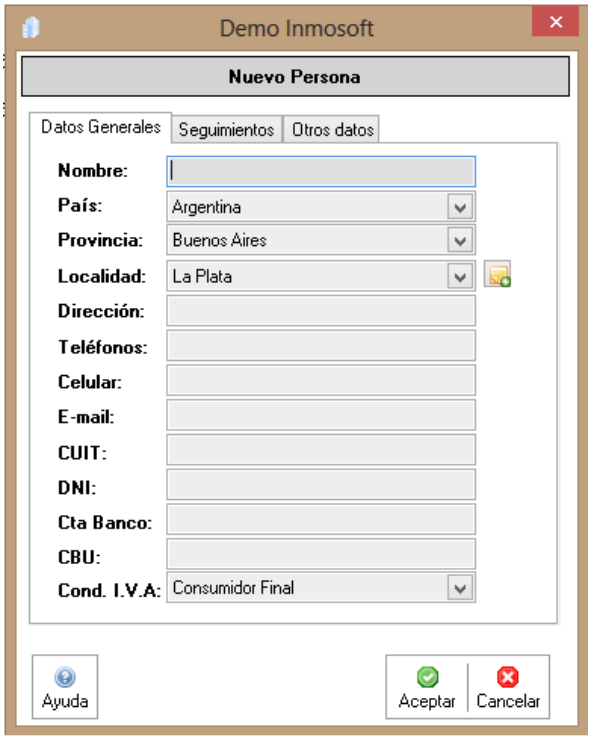


Ilustración 15: (Interfaz de prueba)

Esta vista es un formulario para el ingreso de los siguientes datos de una persona (Tabla 4):

Atributo	Descripción
Nombre	Texto Libre que representa el nombre de la persona
País	Valor seleccionable de un combo que representa el país de la persona
Provincia	Valor seleccionable de un combo que representa la provincia de la persona
Localidad	Valor seleccionable de un combo que representa la localidad de la persona
Dirección	Texto Libre que representa la dirección de la persona
Teléfonos	Texto Libre que representa los teléfonos de la persona
Celular	Texto Libre que representa el celular de la persona
E-mail	Texto con formato de email que representa el email de la persona
CUIT	Texto con formato de CUIT que representa el CUIT de la persona
DNI	Texto Libre que representa el DNI de la persona
Cta Banco	Texto Libre que representa el Cta Banco de la persona
CBU	Texto Libre que representa el CBU de la persona
Cond. IVA	Valor seleccionable de un combo que representa el IVA de la persona

Tabla 4 : Atributos de entidad persona

Para diseñar el modelo de pruebas de la UI, lo primero que se definieron son los requerimientos o validaciones que debe implementar la interfaz. A partir de esto se abstraen los aspectos que se moldearán de la interfaz, de modo que se comporte de acuerdo a los requerimientos.

Las validaciones que se tuvieron en cuenta a la hora de probar una interfaz gráfica de usuario son:

- Validación de campos obligatorios.
- Validación de la longitud/rango apropiada de los campos.
- Validación del formato de los campos de entrada de la interfaz.
- Validación personalizada de datos.
- Validación del despliegue de ventanas, diálogos y mensajes.
- Validación de la apropiada activación de controles de la interfaz.
- Validación de invocación de operaciones ante una acción específica de usuario.

Para el caso de la vista (ilustración 15), las validaciones definidas para los principales elementos gráficos o controles que componen esta interfaz son mostradas en la siguiente tabla (Tabla 5). El tipo de validación indica que pertenece a algún grupo descrito anteriormente.

Elemento	Tipo de validación	Descripción
Campo Nombre	Validación de campo obligatorio	El nombre no puede ser vacío
	Validación de longitud	La longitud del nombre no debe ser mayor a 50 caracteres
Campo País	Validación de campo obligatorio	El país no puede ser vacío, debe seleccionarse del combo
Campo Provincia	Validación de campo obligatorio	La provincia no puede ser vacío, debe seleccionarse del combo
Campo Localidad	Validación de campo obligatorio	La localidad no puede ser vacío, debe seleccionarse del combo
Campo Email	Validación del formato	El campo email debe respetar el formato genérico de emails.
Campo CUIT	Validación del formato	El cuit debe respetar el formato CUIT.
Campo IVA	Validación de campo obligatorio	El IVA no puede ser vacío, debe seleccionarse del combo

Tabla 5 : Validaciones

Una vez definido el caso de prueba, se procede a automatizarlo para luego validar post migración. Para crear pruebas de UI codificadas con Visual Studio basta con realizar la prueba manualmente mientras el generador de pruebas CUIT se ejecuta en segundo plano. También es posible especificar qué valores deben aparecer en campos concretos. El generador de pruebas CUIT registra las acciones y genera código a partir de estas. Después de crear la prueba, también es posible editarla en un editor especializado que permita modificar la secuencia de acciones [UI Automation VS].

6.4 E3 - Generación automática de arquitectura SOA

La tercera etapa de implementación del proceso de migración (ilustración 16) está destinada a generar una arquitectura SOA base inferida automáticamente desde la base de datos disponible.

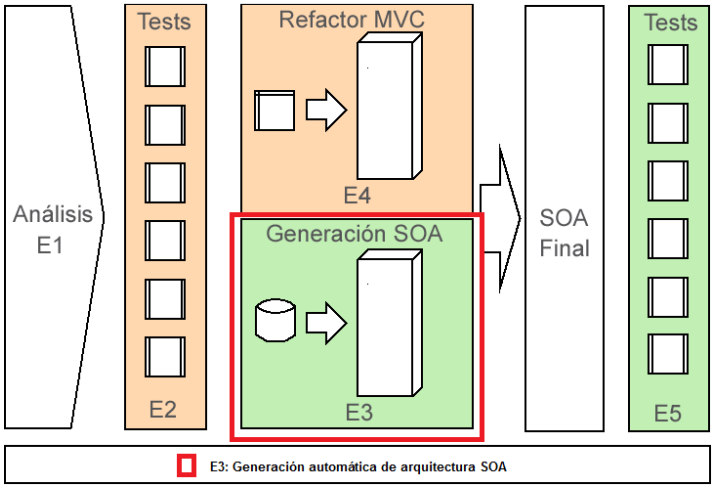


Ilustración 16: (E3: Generación SOA)

La ilustración 17 nos grafica los elementos que intervienen en esta etapa del proceso. El objetivo de esta etapa es generar los artefactos que integrarán la arquitectura SOA inferidos automáticamente desde la base de datos.

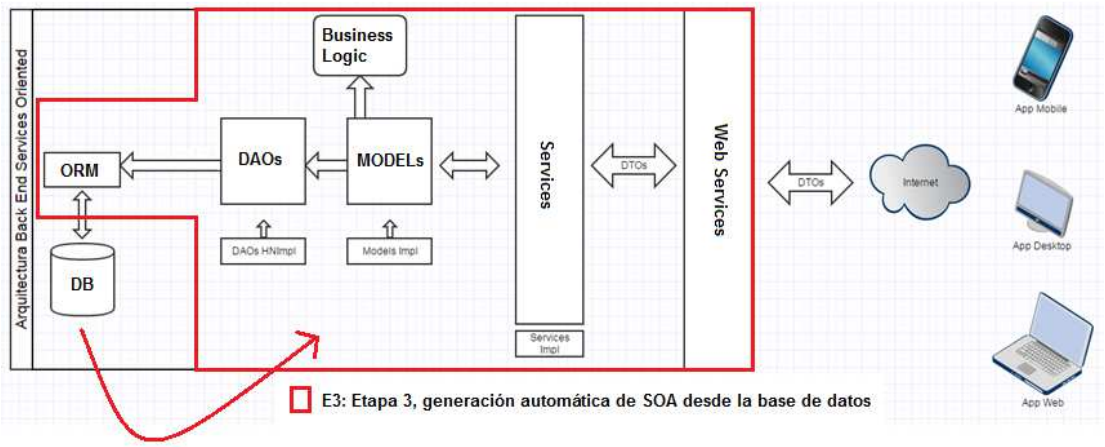


Ilustración 17: (Generación automática de SOA)

Luego de disponibilizar la arquitectura SOA, el proceso avanza a la siguiente etapa donde se busca integrar con el sistema legacy migrado. A continuación, se realiza un análisis e implementación de la etapa del proceso sobre un caso de prueba simplificado para fines didácticos.

6.4.1 E3 a - Análisis de requerimientos SOA

Como analizamos previamente, en esta etapa del proceso de migración es necesario identificar y clasificar los servicios candidatos sobre los cuales apoyaremos la siguiente etapa de refactor del sistema legacy así como también quedarán expuestos para ser consumidos desde otras aplicaciones (ver ejemplo app mobile).

En este punto, el objetivo es identificar y clasificar los servicios que debe exponer nuestra arquitectura SOA resultante. Una vez identificados dichos servicios, es necesario clasificarlos dependiendo de si es un servicio genérico o uno específico personalizado.

Por servicio genérico entendemos todo aquel dato o estructura que puede ser inferido automáticamente desde su origen de datos (en este caso base de datos SQL Server) y expuesto como tal proveyendo de sus operaciones automáticas tales como alta, baja, modificación y búsqueda (Etapa 3-b detallada a continuación).

Por servicios específico entendemos todo aquel dato o estructura resultado de una operación especifica propia del dominio de la aplicación (Etapa refactor - 4 detallada a continuación).

Resumen de servicios candidatos del módulo de personas (Tabla 6).

Servicios genéricos automáticos	Servicios específicos
• Listado de Personas • Listado de Paises • Listado de Provincias • Listado de Localidades • Listado de Condiciones de IVA • Alta, Baja, Modificación y Búsqueda de todas las entidades que intervienen en el módulo (Personas, Paises, Provincias, Localidades, CondIVA)	• Validación lógica de alta de persona. • Validación lógica de Modificación de persona. • Validación de CUIT. • Total de dinero en cuenta corriente
Estos servicios serán generados automáticamente en la siguiente etapa del proceso (E3 b)	Estos servicios serán migrados manualmente en la siguiente etapa del proceso (E4)

Tabla 6 : Servicios candidatos

6.4.2 E3 b - Generación automática de arquitectura SOA desde modelo de datos

Actualmente es posible generar código automático partiendo de la definición de una base de datos, para el caso de tecnologías Microsoft .Net existen diversas herramientas que pueden ayudarnos en la tarea. A continuación, repasaremos algunas de las herramientas más utilizadas:

- **Hibertante:** Hibernate Tools son un conjunto de herramientas para Hibernate que nos permite entre otras cosas generar código de clases a partir del modelo de base de datos.
  Web Site: www.hibernate.org
  Ultimo acceso: Junio 2017.

- **Data TierGenerator:** Herramienta para generar clases C# desde procedimientos almacenados y SQLs de SQL Server.
  Web Site: https://datatiergenerator.codeplex.com
  Ultimo acceso: Junio 2017.

- **CodeBhagat:** Herramienta para generar clases, interfaces y servicios inferidas de bases de datos.
  Web Site: www.codebhagat.com
  Ultimo acceso: Junio 2017.

- **TierDeveloper, LayerCakeGenerator .NET, CodeFluentEntities:** ORMs y generadores de código que nos permite generar arquitectura en capas expuestas como servicios.
  Web Site: www.alachisoft.com/tdev
  Web Site: http://www.layercake-generator.net/
  Web Site: https://www.softfluent.com/product/codefluent-entities
  Ultimo acceso: Junio 2017.

- **CodeTrigger:** Plug-in de Visual Studio que ofrece ORMs y generadores de código que nos permite generar arquitectura en capas expuestas como servicios.
  Web Site: www.codetrigger.com
  Ultimo acceso: Junio 2017.

En nuestro caso, nos inclinamos por la herramienta Code Trigger ya que nos brinda en forma gratuita las siguientes características como plugin de Visual Studio:

- Genera componentes en C# .NET.
- Genera capas de acceso a datos, dominio y web services de calidad aceptable.
- Genera arquitectura de servicios WCF de Microsoft.
- Permite personalizar el código generado.
- Está integrado con Visual Studio.
- Mapea objetos desde SQL.
- Permite generar interfaces GUI.

Prueba de Concepto (PoC)

En esta prueba de concepto nos enfocaremos en todo lo necesario relacionado a esta fase para generar las altas, bajas y modificaciones de personas del sistema legacy Inmosoft (ABM de Personas) con la finalidad de ejemplificar los aportes del proceso.

Como punto de partida, actualizaremos el entorno de desarrollo legacy procurando actualizar hacia entornos modernos que sean compatibles tanto hacia atrás con nuestro código legacy como hacia delante en relación a arquitecturas SOA. En nuestro caso, la tabla 7 detalla las tecnologías origen y destinos involucradas.

Base origen	Base destino	IDE Origen	IDE Destino
SQL 2000	SQL 2012	Delphi 6	Delphi XE7

Tabla 7 : Tecnologías involucradas

Partiendo de un subconjunto de tablas y relaciones almacenadas en un motor de Base de datos SQL Server 2012 (ilustración 18) generaremos todos los artefactos relacionados con la herramienta CodeTrigger para exponerlas como Web Services .Net de manera automática.

Sub conjunto de tablas (Dominio acotado para PoC)

Inmosoft está compuesto por un conjuntos de **Empresas** que representan a cada una de las inmobiliarias del dominio. Una empresa (Inmobiliaria) tiene una colección de sus **Personas** relacionadas. Cada persona registrada pertenece a una única inmobiliaria. Al momento de cargar una persona se debe indicar el **País**, la **Provincia**, la **Localidad** y la **Condición de IVA** (entre otros atributos básicos). Para todas las empresas, existe una colección de países, provincias relacionadas por países y localidades relacionadas por provincia. La Tabla 8, detalla las entidades del dominio acotado. La ilustración 18, detalla los atributos y entidades relacionadas.

Tabla	Descripción
Paises	Tabla que almacena los países del dominio.
Provincias	Tabla que almacena las provincias del dominio.
Localidades	Tabla que almacena las localidades del dominio.
CondicionesIva	Tabla que almacena las posibles condiciones de Iva para una persona.
Personas	Tabla que almacena las personas del dominio.
Empresas	Tabla que almacena las empresas del dominio.

Tabla 8 : Dominio limitado PoC

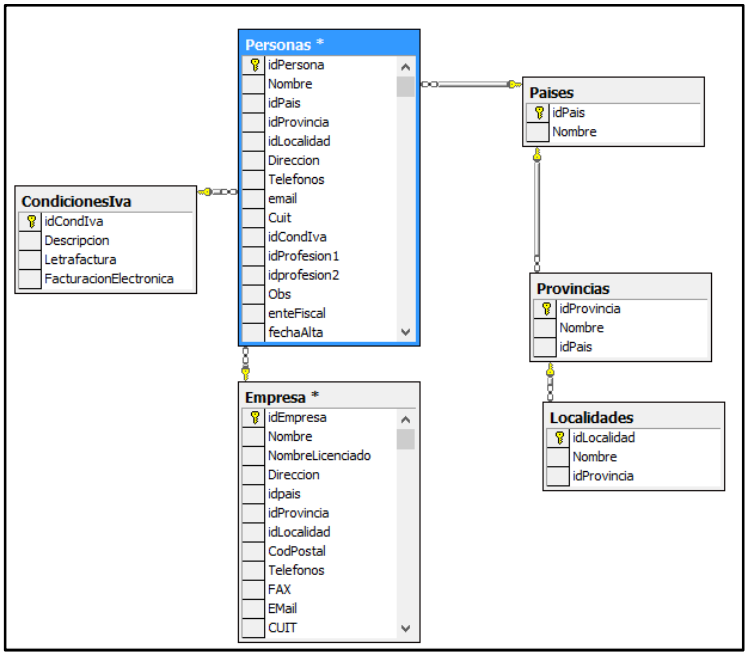


Ilustración 18: (Modelo acotado para prueba de concepto)

A continuación, se detalla paso a paso la generación automática de la arquitectura SOA en base al modelo propuesto.

Paso 1: Importar el modelo SQL con la herramienta CodeTrigger (ilustración 19)

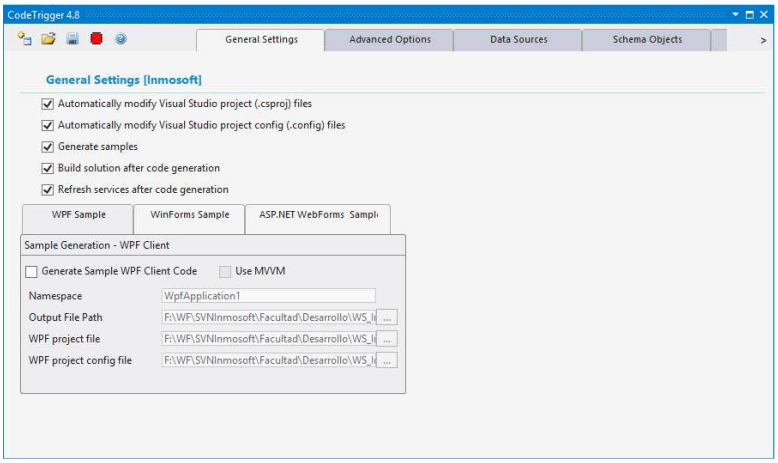


Ilustración 19: (Herramienta CodeTrigger)

Paso 2: Importar la solución generada por CodeTrigger a Visual Studio, compilar y ejecutar Web Services (ilustración 20)

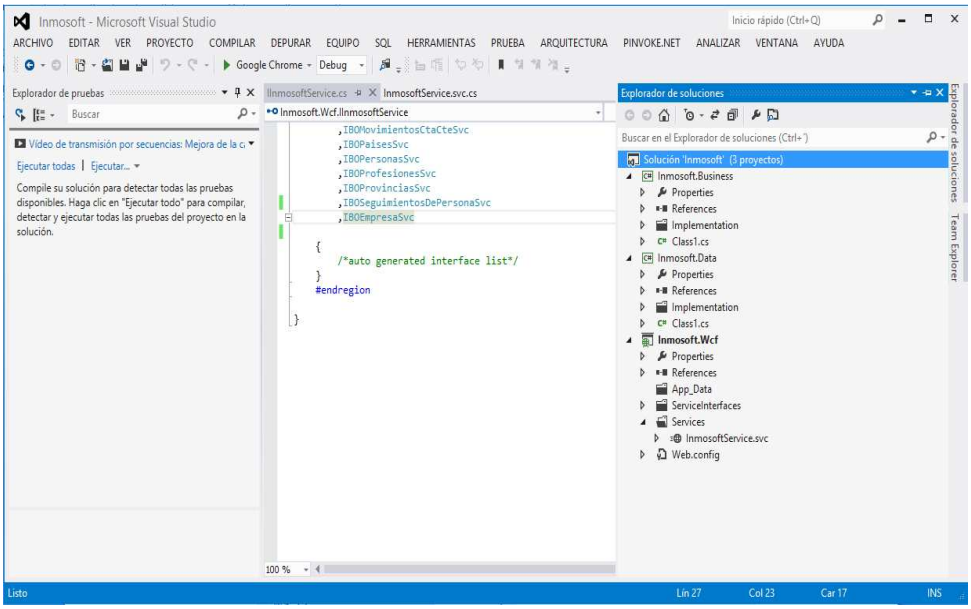


Ilustración 20: (Código generado Visual Studio)

Paso 3: Disponibilizar Web Services generados. La ilustración 21 muestra los servicios web corriendo en servidor local. La ilustración 22 muestra los resultados de una invocación a un servicio.

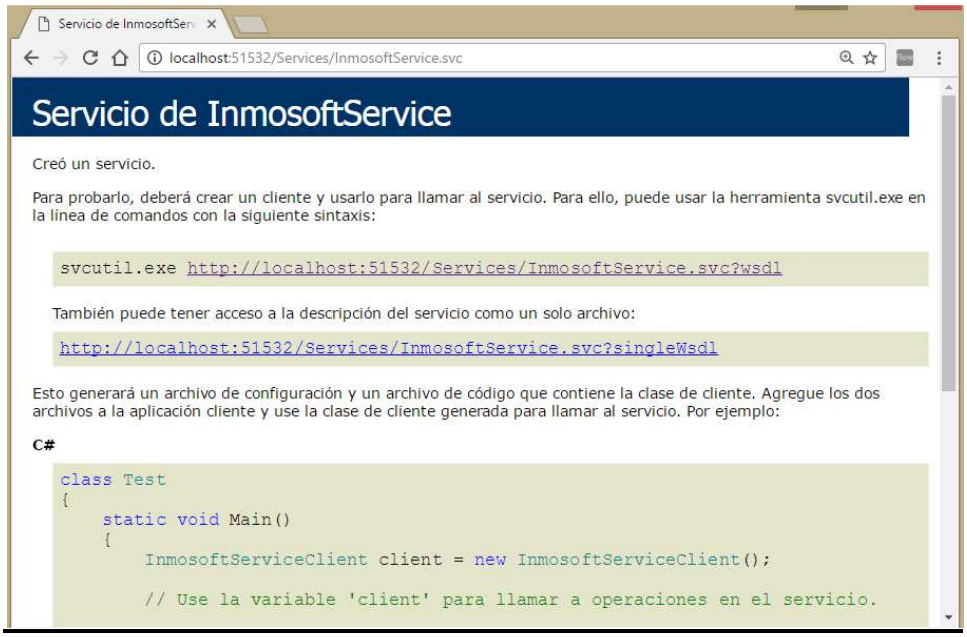


Ilustración 21: (Web Services generados WCF)

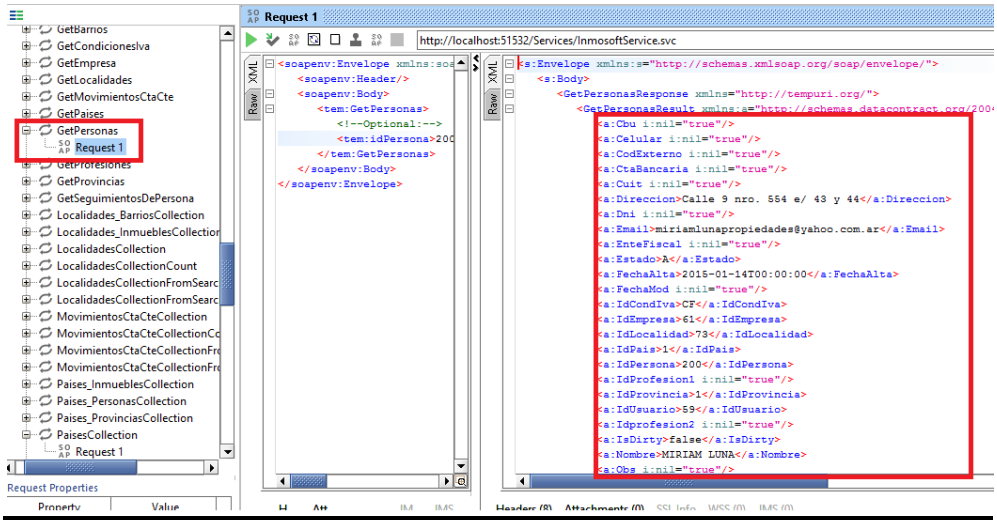


Ilustración 22: (Invocando servicio generado desde cliente SOAP)

Una vez disponibles los Web Services para el dominio acotado presentado, finaliza la etapa 3 del proceso (E3).

6.5 E4 - Refactor Delphi monolítico a MVC

La cuarta etapa del proceso de migración (ilustración 23) está destinada a separar manualmente aspectos entremezclados de aplicaciones Legacy monolíticas. Se busca

separar interfaz de usuario, lógica de negocios y acceso a datos consumiendo los Web Services provistos en la etapa anterior. Se propone una guía sistemática de apoyo en el refactor.

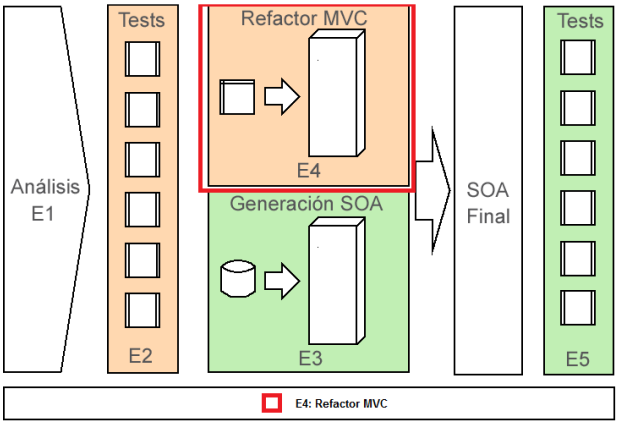


Ilustración 23: (E4: Refactor MVC)

Las aplicaciones monolíticas presentan la problemática de tener su código entre mezclado. Las aplicaciones Delphi desarrolladas bajo la tecnología RAD no escapan a dichos problemas, tanto el código relacionado a las interfaces de usuario, lógica de negocios y acceso a datos se encuentra mezclado entre sí generando un acoplamiento extremo el cual atenta contra la escalabilidad del sistema.
El objetivo de esta etapa del proceso es lograr una clara separación entre cada aspecto del sistema; en concreto, separar cuestiones propias de las interfaces de usuario, cuestiones propias del negocio y de acceso a datos (ilustración 24).

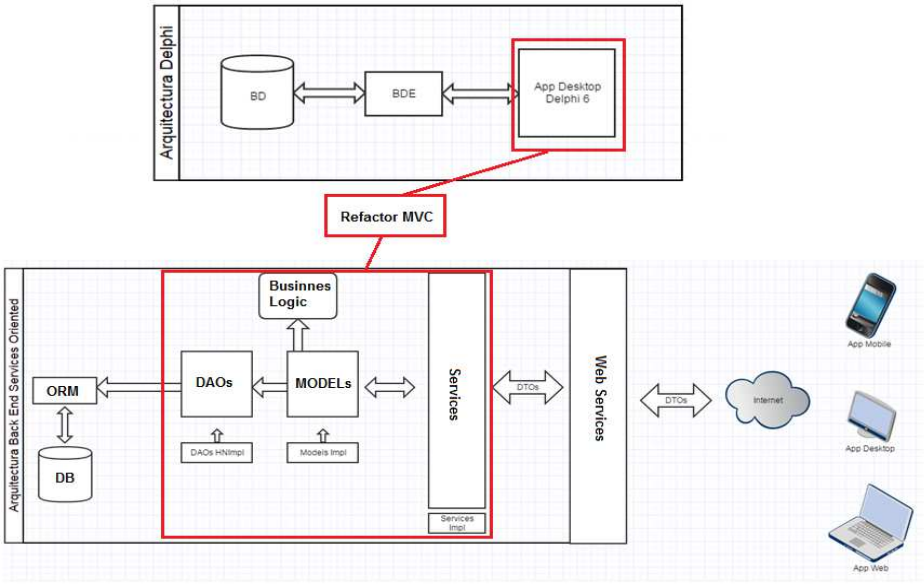


Ilustración 24: (Monolito a MVC en integración SOA)

A continuación, se presenta una guía compuesta por cinco etapas de refactor con una alternativa de solución la cual persigue las bondades del conocido patrón MVC con la idea de hacer ajustes simples pero eficientes sobre el sistema legacy actual.

6.5.1 Refactor etapa 0 (Introducción)

Tomaremos como PoC el ABM de Personas del sistema legacy el cual presenta el formulario de la ilustración 25 en vista diseño y de la ilustración 26 en vista ejecución.

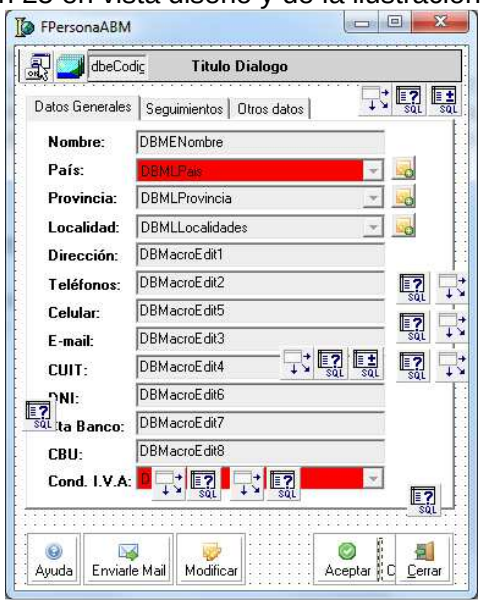


Ilustración 25: (Vista diseño)

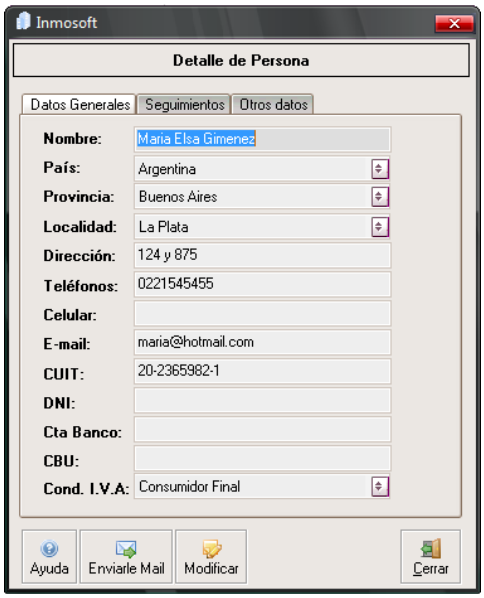


Ilustración 26: (Vista en ejecución)

Principales Problemáticas

Como se comenta anteriormente, este tipo de soluciones presenta un fuerte acoplamiento en todos los aspectos, a continuación se ejemplifica.

- **Problema 1:** El Acceso a datos está fuertemente acoplado a la interfaz de usuario.

Los controles no visuales como el TQuery SQL  contienen el acceso a datos mediante SQL harcodeado (ilustración 27).

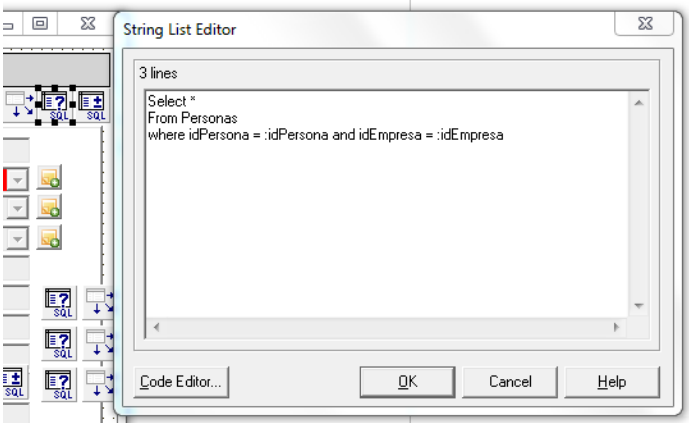


Ilustración 27: (Ejemplo de SQL harcodeado en controles)

- **Problema 2:** La implementación del botón aceptar realiza validaciones de negocio, setea datos y aplica los cambios (ilustración 28).

```
procedure TFPersonaADM.AAceptarExecute(Sender: TObject);
{ACEPTAR}
begin

  if (not AlcanzoLimiteLicencia ) then
  begin
    if (trim(isNull(DBMENombre.Text,''))<> '') and (qPersona['idpais'] <> null) and (qPersona['idCondIva']<>null) then
    begin
      DM.DBMacro.StartTransaction;
        try
          SetCamposTracking(qPersona,_oper);
          qPersona.Post;
          qPersona.ApplyUpdates;
          qPersona.CommitUpdates;
          _idGenerado:=MaxIdDeTablaByUsuario('idPersona','Personas');
          GrabarSeguimientos;
          DM.DBMacro.Commit;
          modalresult:=mrOk;
        except
        on E:Exception do
        begin
          DM.DBMacro.Rollback;
          MessageDlg('No se pudo realizar la operación.'+#13+E.Message,mtWarning,[mbOk],0);

        end;
      end;
    end
    else
      Information(__COMPLETAR_CAMPOS_OBLIGATORIOS__);
  end
  else
  begin
    Information('ATENCION! Usted alcanzó el límite de personas para su licencia '+__EMPRESA.LicenciaActiva.getCuenta().GetDescripcion+'
                . Por favor actualice su licencia para poder cargar más personas.');
    FDetallesLicencia:=TFDetallesLicencia.create(self);
    FDetallesLicencia.ShowModal;
    FDetallesLicencia.Free;
  end;

end;
```

Ilustración 28: (Ejemplo de implementación monolítica)

Enfoque de solución

Una posible solución a este tipo de problemas es realizar un refactor intentando desacoplar todo lo posible las responsabilidades de cada capa de implementación. A continuación se resume cómo deberían conformarse cada capa.

Formularios (Interfaz de usuario)

Los formularios se deben limitar a la representación de los datos (Vista) e interacción con el usuario, ante la necesidad de algún dato debe interactuar con su controlador (archivo .pas).

Archivos .pas (Controlador)

El archivo .pas tomará el rol del controlador en el esquema MVC simplificado, será el responsable de orquestar las necesidades de la vista e invocar al modelo (Módulo de datos) cuando requiera lógica de negocios o acceso a datos.

Módulos de datos (Modelo)

Delphi provee estos archivos con la finalidad de ser un repositorio de controles de acceso a datos, en nuestro caso tomarán el rol de Modelo en el esquema MVC simplificado y serán los encargados de comunicarse con los Web Services provistos en la etapa anterior (E3).

6.5.2 Refactor etapa 1 (Preparar estructura)

La primera etapa del refactor consiste en preparar la estructura de los archivos necesarios para aplicar la separación de responsabilidades, los pasos son:

1. Crear Data Module propio para la entidad en cuestión (ej: PersonaModelDM).
2. Pasar al Data Module todos los controles de acceso a datos existentes en el formulario.(luego serán reemplazados por consumidores de Web Services)
3. Relacionar el Data Module con el formulario.

La ilustración 29 muestra la estructura resultante.

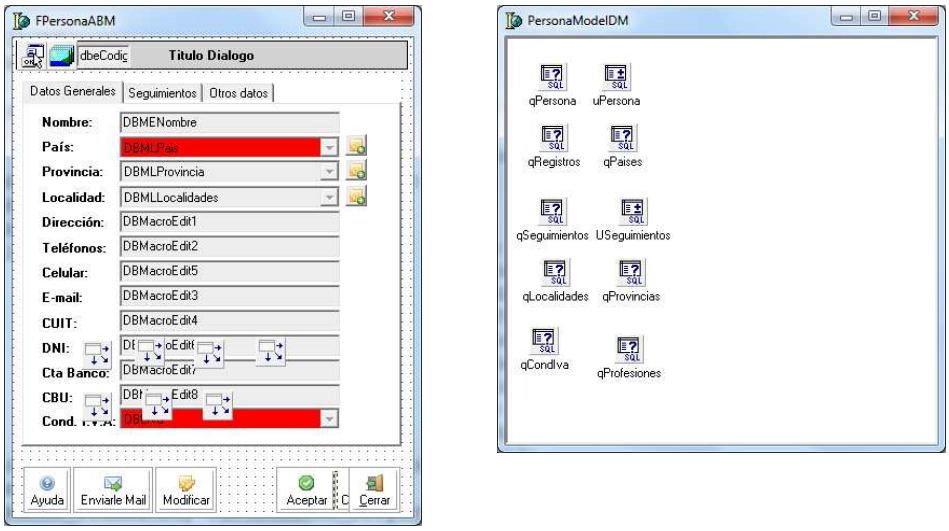


Ilustración 29: (Estructura resultante)

6.5.3 Refactor etapa 2 (Separar Acceso a Datos)

La etapa 2 del refactor está destinada desacoplar el acceso a datos donde corresponda, para ello se debe realizar un barrido por cada método existente en la implementación del formulario, si se detecta código de acceso a datos este debe ser retirado y colocado en el módulo de datos previamente creado.

Ejemplo

En el código legacy actual, al configurar el formulario está accediendo a los datos mediante el control TQuery (qPersona) (ilustración 30)

```
procedure TFPersonaABM.configurar(oper: char; idPersona: integer;titulo:string);
{CONFIGURAR}
begin
  _oper:=oper;
  _idPersona:=idPersona;
  qPersona.Close;
  qPersona.ParamByName('idPersona').Value:=idPersona;
  SetParametrosVisibilidad(qPersona);
  qPersona.Open;
```

Ilustración 30: (Ejemplo de acceso a datos monolítico)

Refactor

Lo correcto sería definir un método en el módulo de datos y que el controlador le solicite los datos. Entonces, cuando vista pide configurarse, el controlador invoca al modelo y el modelo es el encargado de retornar la información.
La ilustración 31 ejemplifica cómo queda el controlador luego de aplicar el refactor.

```
procedure TFPersonaABM.configurar(oper: char; idPersona: integer;titulo:string);
{CONFIGURAR}
begin
  _oper:=oper;
  _idPersona:=idPersona;

  PersonaModelDM.GetByID(idPersona);
```

Ilustración 31: (Ejemplo de acceso a datos post refactor)

La ilustración 32 ejemplifica cómo queda el modelo luego de aplicar el refactor.

```
procedure TPersonaModelDM.GetByID(id: Integer);
begin
  qPersona.Close;
  qPersona.ParamByName('idPersona').Value:=Id;
  SetParametrosVisibilidad(qPersona);
  qPersona.Open;
end;
```

Ilustración 32: (Ejemplo de acceso a datos post refactor)

De esta forma, eliminamos de la vista o el controlador el acceso a datos existente.

6.5.4 Refactor etapa 3 (Acceso a Web Services)

En este punto debemos ser capaces de consumir los Web Services provistos por la etapa anterior (E3), para este caso el servicio en cuestión se encuentra publicado en un servidor local Internet Information Services (IIS).

Publicación IIS Local:
http://localhost:51532/Services/InmosoftService.svc?singleWsdl

Paso 1

El primer paso es realizar la importación del Web Service sobre Delphi, la ilustración 33 ejemplifica la importación.

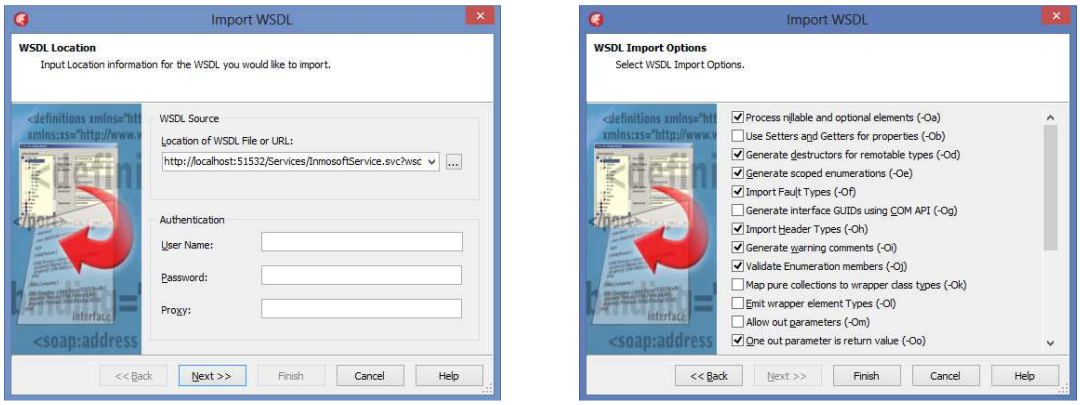


Ilustración 33: (Importación de WS)

Como resultado de la importación, tenemos una nueva unidad Delphi “InmosoftService” la cual utilizaremos para acceder a los servicios.

Paso 2

La Tabla 9 indica los controles que es necesario incorporar al módulo de datos (PersonaModelDM).

HTTPRIOPersona	HTTPRIO: Este control nos permitirá enlazar con los Web Services SOAP y suscribirnos opcionalmente a sus eventos.
CDSPersonas	ClientDataSet: Este control reemplazará el TQuery y será el encargado de manejar la estructura de los datos y sus correspondientes tipos.
DSPersona	DataSource: Este control nos permitirá enlazar los datos con controles visuales.

Tabla 9 : Controles refactor

Este conjunto de controles nos permitirá eliminar los SQLsHarcodeados, el acoplamiento entre capas y la total abstracción del Back End que será provista por los Web Services.

Paso 3

- Copiar la definición de campos del control SQL y pegarlos en el CDSPersonas (ilustración 34)

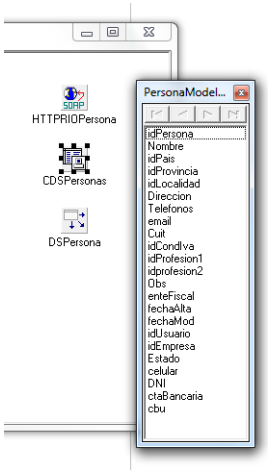


Ilustración 34: (Ejemplo de atributos entidad)

En este punto, ya podemos eliminar el control indeseado TQuery

- Crear los siguientes métodos genéricos en el PersonaModelDM (ilustración 35)

```
private
  { Private declarations }
  _entity: BOPersonas2;
  procedure MappingEntityToCDS();
  procedure MappingCDSToEntity(oper:char);
  function GetEntity():BOPersonas2;
  procedure SetEntity(entity:BOPersonas2);
public
  { Public declarations }
  procedure GetByID(id:Integer);
  procedure UpdateEntity();
  Procedure AddEntity();
  function SaveEntity(oper:char):boolean;
  function ValidateEntity():boolean;
```

Ilustración 35: (Métodos de acceso a datos)

- la ilustración 36 muestra la implementación del método GetByID(id:integer) consumiendo en Web Service relacionado.

```
procedure TPersonaModelDM.GetByID(id: Integer);
begin
  try
    SetEntity(GetIInmosoftService().GetPersonas(id));
    MappingEntityToCDS;
  except
    on E:Exception do
      begin
        MessageDlg('No se pudo realizar la operación.'+#13+E.Message,mtInformation,[mbOk],0);
      end;
  end;
end;
```

Ilustración 36: (Ejemplo Implementación GetByID)

- La ilustraciones 37 y 38 muestran la estructura final post refactor.

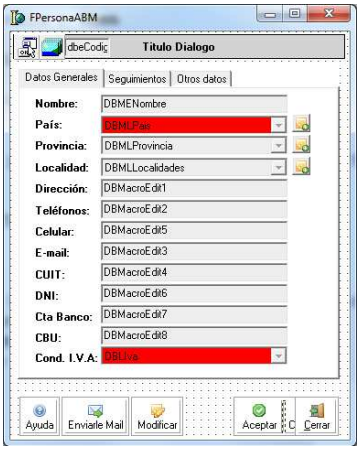


Ilustración 37: (Vista final - Formulario Delphi)

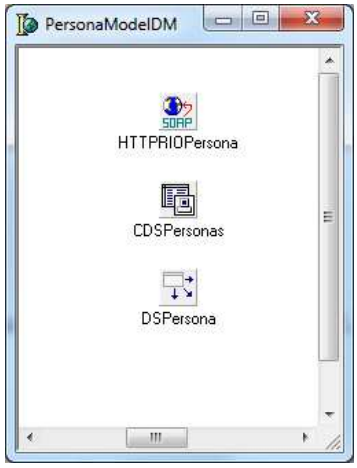


Ilustración 38: (Modelo - Data Module)

La ilustración 39 muestra cómo queda el controlador luego de aplicar el refactor.

Controlador (.pas ej Método configurar)

```
procedure TFPersonaABM.configurar(oper: char; idPersona: integer;titulo:string);
{CONFIGURAR}
begin
  _oper:=oper;
  _idPersona:=idPersona;

  Case oper of
  'A':begin
        Panel3.Caption:='Nuevo '+titulo;
        TBSendMail.Visible:=False;
        PersonaModelDM.AddEntity;
      end;
  'M':begin
        Panel3.Caption:='Modificación de '+titulo;
        PersonaModelDM.GetByID(idPersona);
        PersonaModelDM.UpdateEntity;
      end;
  'D':begin
        Panel3.Caption:='Detalle de '+titulo;
        PersonaModelDM.GetByID(idPersona);
        ConfigurarComoDetalle();
      end;
  end;
end;
```

Ilustración 39: (Ejemplo controlador)

Notas y consideraciones

Dado que este punto del proceso es fundamental para un correcto avance de la migración se propone como trabajo a futuro profundizar el mecanismo de refactor para lograr una mayor automatización del mismo. Para el avance teórico-práctico de la tesis presentada este punto del proceso lo aplicaremos de un modo manual con el objeto de poder avanzar en las siguiente etapas y concluir con la prueba de concepto dando una visión integral de la solución [TF-2].
Un enfoque que podría ayudar en la automatización del proceso es focalizarse en reglas de derivación, es decir, los pasos antes descriptos enumerarlos como reglas con el objetivo de que en base a estas, automatizar distintos aspectos. Como por ejemplo la generación automática de las estructuras y métodos necesarios para el refactor [TF-3].

6.5.5 Refactor etapa 4 (Separar lógica de negocios)

La cuarta etapa del refactor consiste en realizar una inspección manual por cada método existente en la implementación del formulario, si se detecta código de lógica de negocios este debe ser retirado y colocado en el módulo de datos previamente creado.

Durante la etapa previa de análisis de servicios candidatos, se detectó que el módulo de personas (Vista Personas) ofrece el saldo de la misma en su cuenta corriente (ilustración 40):

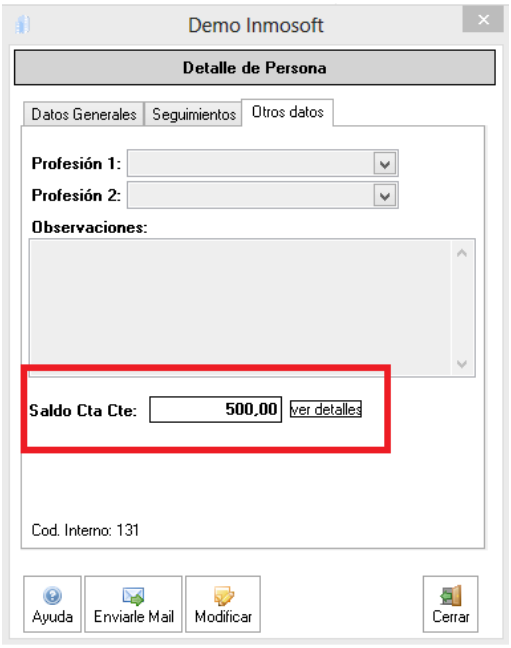


Ilustración 40: (Saldo en Cta Cte de la persona)

Analizando esta funcionalidad, se tomó como servicio candidato específico a un servicio que dado la identificación de una persona nos retorne el saldo en su cuenta corriente.

Tomaremos este escenario para ejemplificar la migración de una lógica personalizada desde el sistema legacy hacia la nueva arquitectura SOA.

Siguiendo con el mecanismo previamente descripto, el punto de partida es aplicar el refactor MVC buscando separar las responsabilidades de la implementación legacy e integrar luego con el servicio correspondiente.

Paso 1 : Refactor MVC sobre sistema legacy.

Analizando el código legacy, nuevamente nos encontramos en la situación que la implementación del retorno del saldo en cuenta corriente del la persona se encuentra mezclada con la interfaz gráfica harcodeando SQLs (ilustración 41)

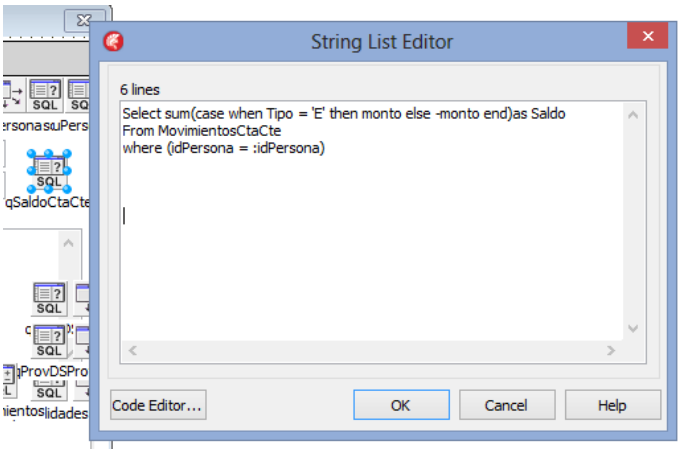


Ilustración 41: (SQL harcodeado en vista)

El método GetSaldoCtaCte (implementado en la interfaz) realiza la consulta SQL y retorna el saldo (ilustración 42).

```
function TFPersonaABM.GetSaldoCtaCte: currency;
begin
  qSaldoCtaCte.Close;
  qSaldoCtaCte.ParamByName('idPersona').Value:=_idPersona;
  qSaldoCtaCte.Open;
  Result:=IsNull(qSaldoCtaCte['saldo'],0);
end;
```

Ilustración 42: (Implementación monolítica)

Siguiendo el mecanismo de refactor explicado anteriormente, se debe disponibilizar el método como parte del modelo (módulo de datos) y eliminar controles y responsabilidades de la vista relacionado a la lógica migrada.

Paso 2: Disponibilizar lógica como servicio personalizado.

Dado que esta lógica no se infiere automáticamente (sección 6.4 E3) durante la generación automática inicial de arquitectura SOA es necesario disponibilizar manualmente el servicio personalizado en nuestra arquitectura resultante.

Para el caso de prueba, dado que la lógica relacionada es el resultado de una consulta SQL optaremos por una migración semi-autómatica. En ese caso, solo es necesario trasladar el SQL hardcodeado en Delphi hacia un store procedure SQL server que luego será inferido automáticamente con la herramienta CODETRIGGER y disponibilizado como servicio.

Creación del Store Procedure SQL Server

Este paso consiste simplemente en generar un store procedure con el SQL extraído de la implementación legacy monolítica (ilustración 43).

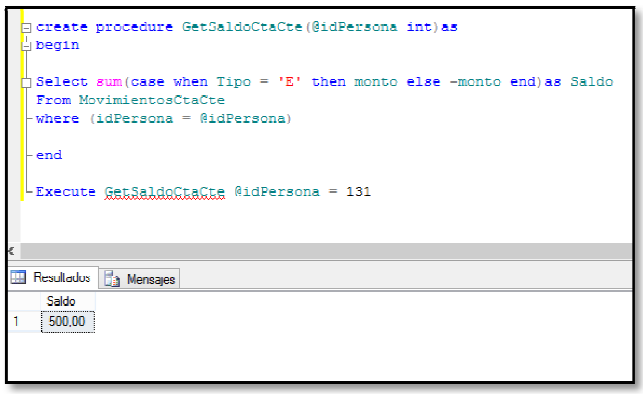


Ilustración 43: (Store procedure SQL)

Servicio inferido y disponibilizado en arquitectura SOA

En este punto, ya se dispone del servicio inferido del store procedure (ilustración 44).

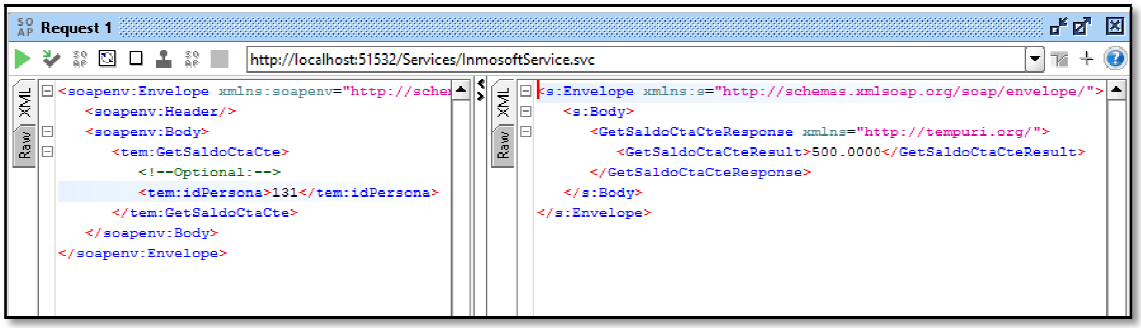


Ilustración 44: (Servicio inferido del store procedure)

Paso 3 : Por último, resta integrar el servicio antes descripto al modelo del refactor MVC para cerrar el circuito (ilustración 45 y 46).

```
function TFPersonaABM.GetSaldoCtaCte: currency;
begin
  Result:=PersonaModelDM.GetSaldoCtaCte(_idPersona);
end;
```

Ilustración 45: (La vista solo invoca al modelo)

```
function TPersonaModelDM.GetSaldoCtaCte(idPersona:Integer): currency;
begin
  try

    Result:= FormatoCurreny((GetIInmosoftService().GetSaldoCtaCte(idPersona)).AsBcd);

  except
    on E:Exception do
      begin
        MessageDlg('No se pudo realizar la operación.'+#13+E.Message,mtInformation,[mbOk],0);
      end;
  end;
end;
```

Ilustración 46: (El modelo invoca al servicio personalizado)

6.5.6 Refactor etapa 5 (Revisión)

Como en todo refactor, es necesario garantizar que las funcionalidades no fueron alteradas durante el proceso. Si bien, en este proceso se propone un refactor con la finalidad de simplificar el proceso y evitar errores críticos, los posibles errores no son ajenos y deben ser tratados en el caso de ser detectados.

Como se describe en el proceso teórico, las pruebas para garantizar que el sistema legacy siga funcionando correctamente estarán focalizadas en pruebas de interfaz de usuario. Sin embargo, es importante también realizar un chequeo en relación al refactor técnicamente aplicado. En concreto, debemos garantizar que se respetaron al menos los siguientes conceptos:

- El ABM debe funcionar igual a como funcionaba antes del refactor.
- La vista (Formulario) no debe conocer nada de lógica ni acceso a datos.
- El controlador SOLO orquesta atendiendo eventos de la vista e invocando al Modelo.
- El modelo (Data Module) tendrá solo las siguiente responsabilidades:
  - Encapsular Lógica de Negocios.
  - Encapsular Acceso a Datos consumiendo los web services de la fase 1.

Una vez garantizados estos conceptos, se procede a ejecutar los test automáticos de UI para comenzar la etapa de revisión de errores.

En la siguiente etapa, se expondrán los resultados de la validación automatizada en base a los test de UI previamente definidos.

6.6 E5 - Validación de casos de prueba (Automatizado)

Siguiendo el proceso de migración propuesto, la etapa final está abocada a realizar una batería de pruebas UI automatizadas sobre la implementación migrada. La ilustración 47 remarca la etapa final en a la que nos abocaremos.

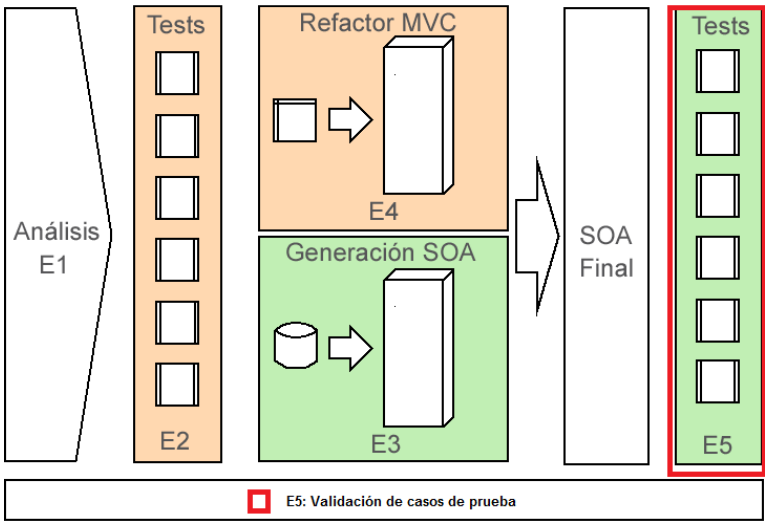


Ilustración 47: (E5: Validación de casos de prueba)

Concretamente, se procede a ejecutar los test UI que previamente se hayan definido (etapa 2) para el módulo migrado en cuestión.
En este caso, como ejemplo de la automatización se trabaja sobre dos escenarios automatizados:

- Agregar una persona validando que se indique el nombre.
- Modificar una persona validando que se indique el nombre.

Como se menciona previamente (etapa 2), la herramienta utilizada para automatizar los test de UI es Visual Studio, la ilustración 48 presenta el explorador de test UI donde se visualizan los dos métodos de ejemplo automatizados.

En este caso, el método *AgregarPersona* fue ejecutado y superó la prueba automatizada.

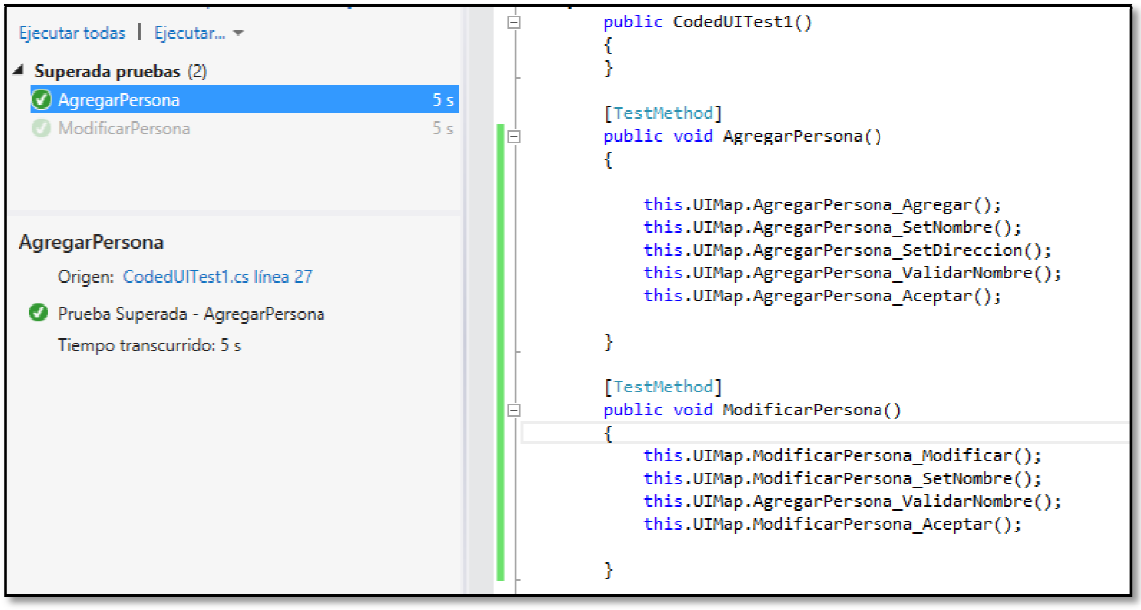


Ilustración 48: (Explorador de test de UI)

La ilustración 49 nos indica el modo en que se ejecutan las pruebas, en este caso se ejecutará el método *ModificarPersona* el cual también es superado con éxito tal cual lo indica la ilustración 50.

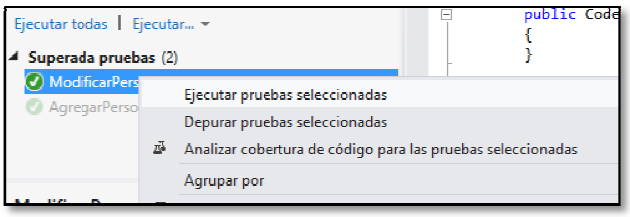


Ilustración 49: (Test UI Modificar Persona)

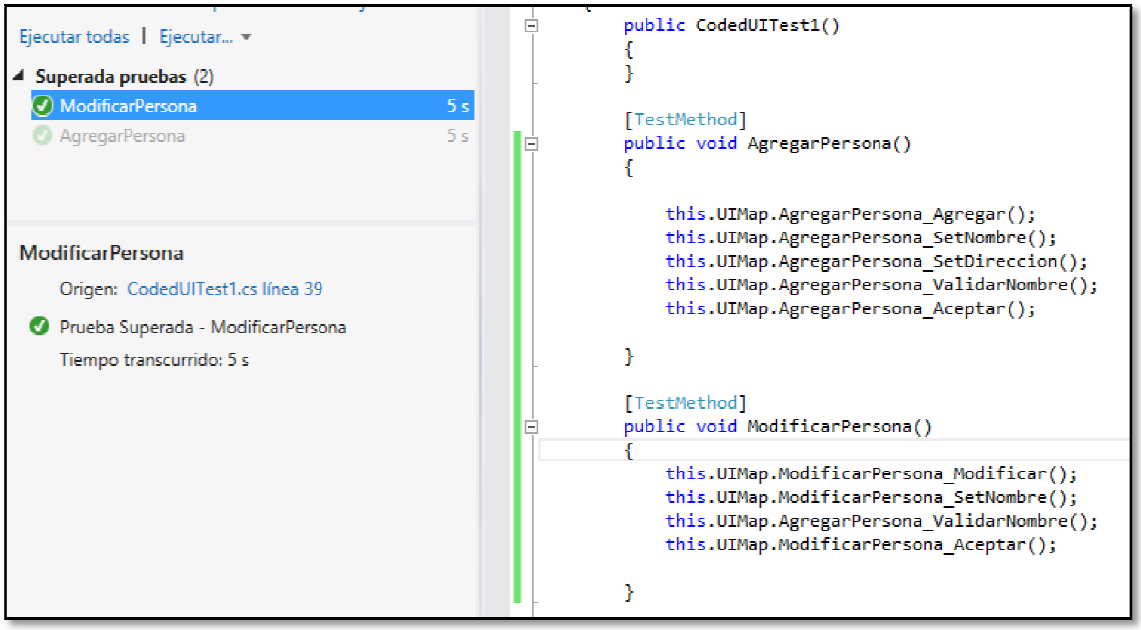


Ilustración 50: (Test UI Modificar Persona superado)

Por supuesto, un test UI automatizado puede fallar como nos indica la ilustración 51. En este caso, se forzó el error comentando la instrucción que setea el nombre, por lo tanto no pasa la validación y arroja los siguiente errores:

Ilustración 51: (Test UI Agregar Persona NO superado)

Si bien, el objeto de esta sección es presentar el mecanismo básico para automatizar test UI la herramienta se puede explotar realizando test tan completos como amerite la

migración. Queda fuera del alcance de este estudio profundizar en las funcionalidades de este tipo de herramientas.

6.7 PoC - Consumiendo servicios desde app mobile

En esta última sección del estudio sobre el caso de prueba, realizaremos una prueba de concepto con la idea de consumir algunos de los servicios provistos por la arquitectura SOA resultante de la migración desde una app mobile especialmente desarrollada para este fin (ilustración 52).

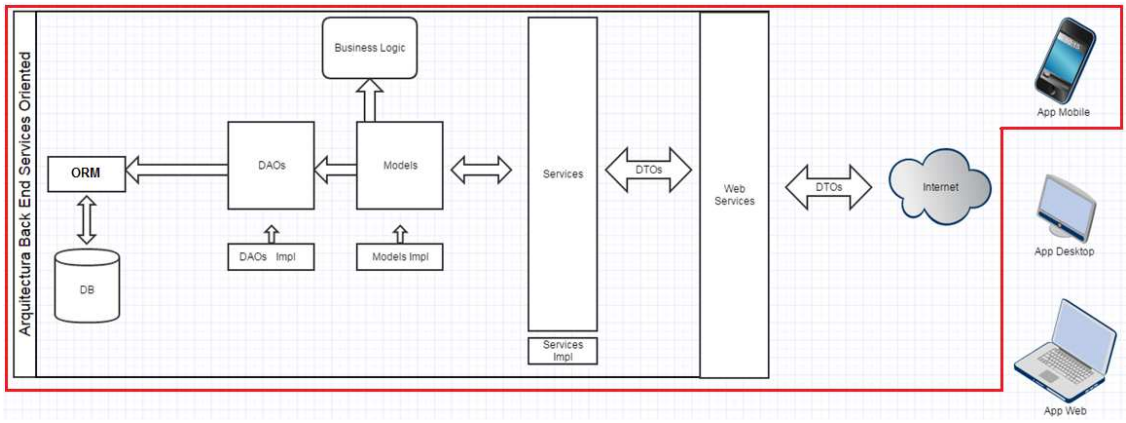


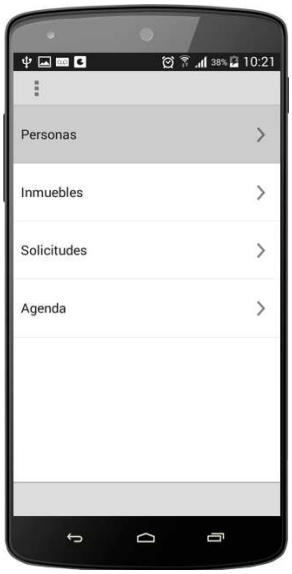
Ilustración 52: (Arquitectura SOA)

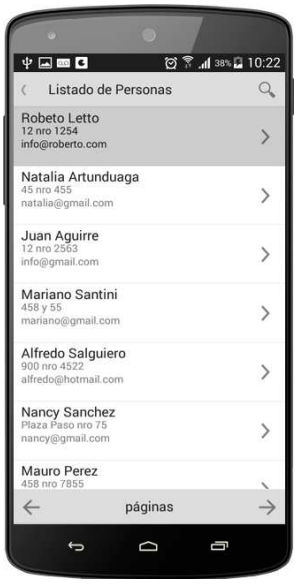
Nos focalizaremos en la construcción de una simple app android que utilice los siguientes servicios previamente disponibilizados durante el proceso de migración:

- Servicio que nos retorna el listado de personas (inferido automáticamente).
- Servicio que nos permite realizar búsquedas de personas (inferido automáticamente).
- Servicio que nos retorna el saldo en cuenta corriente de una persona (inferido semi - automáticamente).

A continuación, se presentan las pantallas que forman parte de la app propuesta.

Login de la app  *Ilustración 53: (Login App)*	Descripción: La primer pantalla (ilustración 53) que visualizamos al ingresar a la app es la de acceso a la misma, si bien a modo de ejemplo es posible ingresar las credenciales, internamente no realiza el login y hardcodea dichos datos para simplificar la demostración ya que existen datos previamente cargados en la base de datos para trabajar con la empresa DEMO.

Menú  *Ilustración 54: (Menú App)*	Descripción: Una vez que ingresamos a la app, esta nos presenta un menú simplificado con distintas opciones (ilustración 54). La opción sobre la cual trabajaremos es "Personas".

Listado de personas  *Ilustración 55: (Listado App)*	Descripción: Ingresando al menú "Personas" accedemos directamente al listado de las personas de la empresa en cuestión (ilustración 55). En este punto, se está haciendo uso del servicio de personas provisto mediante la migración. Desde esta sección, es posible realizar búsquedas y ver los detalles de para una persona dada. Dichas opciones se detallan a continuación.

Buscador  *Ilustración 56: (Buscador App)*	Descripción: Uno de los servicios provistos automáticamente mediante el proceso de migración es la búsqueda sobre una entidad dada, en este caso utilizaremos la búsqueda de una persona (ilustración 56).

Resultado buscador  *Ilustración 57: (Resultado App)*	Descripción: La ilustración 57 nos presenta la pantalla del resultado de la búsqueda realizada. Tocando el elemento (persona) podemos acceder a visualizar el detalle de la misma.

Detalle de la persona  *Ilustración 58: (Detalle App)*	Descripción: Por último, combinando los servicios de búsqueda, listado de personas y saldo en cta cte podemos visualizar el detalle de los datos de la persona seleccionada (ilustración 58)

Capítulo 7 - Trabajo futuro

Más allá del esfuerzo invertido en la realización de esta tesis, la misma aún puede ser mejorada en varios aspectos, a continuación se enumeran algunos puntos importantes como semillas para futuros trabajos relacionados.

[TF-1] Migración de orígenes de datos antiguos:

Uno de los puntos claves del proceso de migración propuesto en esta tesis tiene que ver con el estado y el tipo de los orígenes de datos. Si bien, la presente tesis se limita a trabajar sobre orígenes de datos de tipo base de datos DBMS, no sería raro encontrarse con sistemas legacy implementados sobre otros tipos de orígenes de datos. Se propone como trabajo a futuro el desarrollo de un subproceso que permita migrar tecnologías más antiguas (ej archivos) hacia DBMS o nuevas tecnologías no relacionales (NoSQL).

[TF-2] Herramienta de refactor monolítico a MVC SOA:

El inevitable proceso de refactor buscando desacoplar todo lo posible el monolítico sistema legacy nos presenta el desafío de automatizar todo lo posible este proceso, dado que este punto del proceso es fundamental para un correcto avance de la migración se propone como trabajo a futuro profundizar el mecanismo de refactor para lograr una mayor automatización del mismo.

[TF-3] Automatizar reglas de derivación (Caso de estudio):

Si bien, es posible automatizar varias de las etapas del proceso de migración existen apartados donde es necesario realizar trabajo manual, como por ejemplo al realizar el refactor monolítico hacia MVC. Sin embargo, una vez descriptos los pasos necesarios para realizar el refactor manual surge la idea como trabajo a futuro de formalizar la secuencia de pasos en reglas e intentar automatizar los aspectos que sean posibles. Como por ejemplo, la generación de estructuras y métodos necesarios para el refactor.

[TF-4] Optimización de migración paralela en grandes equipos de trabajo

Durante las etapas de análisis pre migratorio de sistemas legacy surge la cuestión de cómo llevar a delante la implementación de la migración en relación al equipo de trabajo. Si bien, el presente estudio no profundiza en la cuestión no es un tema menor cuando se dispone de un gran equipo el cual permita paralelizar tareas. En estos casos, sería muy importante analizar en detalle el modo óptimo de implementar migración paralela con grandes equipos de trabajo, se propone como trabajo a futuro profundizar dicho estudio.

Capítulo 8 - Trabajos relacionados

El tema de estudio abarcado en la presente tesis fue y probablemente seguirá siendo un campo muy estudiado y continuará evolucionando y mejorando las técnicas de

migración de sistemas legacy. A continuación, hacemos referencia a una serie de estudios que vale la pena considerarlo y analizarlos.

Los estudios de [Salvatierra 2012] son muy interesantes como complemento a la presente tesis, ya que el objetivo propuesto es mejorar la frontera SOA obtenida en una migración directa sin el costo de la migración indirecta.

El enfoque de [Salvatierra 2012] se basa en dos tipo de migraciones, la directa y la indirecta, donde:

- Migración directa: La migración es incremental manteniendo el sistema legacy (este sería el caso de estudio de la tesis). Ese tipo de migración, es rápida y económica pero con el costo de tener menos calidad en el SOA generado.
- Migración indirecta: En este caso, se da un reemplazo total del sistema legacy. Por supuesto, es más caro, lento pero probablemente con una calidad superior de SOA.

La idea propuesta es aplicar un proceso semi automático que permita mejorar la frontera SOA obtenida en una migración directa sin el costo de la migración indirecta.

Para evaluar el enfoque se utiliza:

- Frontera SOA indirecta.
- Frontera SOA directa.
- Frontera SOA directa + proceso propuesto.

Los resultados muestran que el enfoque produce una frontera SOA casi tan buena como la indirecta, pero a un costo similar al de la migración directa.

Herramientas utilizadas: Utiliza la técnica de buscar antipatrones WSDL para detectar problemas y ofrecer mejoras (herramienta COBOL to SOA).

Por otro lado, hacemos mención también al estudio de [Hunold 2008], relacionado al campo del refactoring de sistemas monolíticos a modulares. En este caso el enfoque se basa en extraer la lógica de negocios del sistema legacy en un metamodelo (esto requiere que el desarrollador separe manualmente la interfaz de usuario), transformar el modelo monolítico en modular (dispone de una herramienta "TransFormr") y generar el código en un lenguaje destino (Java).

El enfoque propuesto por [Hunold 2008] es interesante, pero como requiere una intervención importante por parte del usuario para generar el metamodelo. Este tipo de estrategias de refactor ha sido estudiada y es muy potente, pero requiere de grandes análisis e implementaciones de herramientas.

Capítulo 9 - Conclusiones

La presente tesis planteó un enfoque de migración de sistemas legacy hacia arquitecturas orientadas a servicios iterativa e integral apoyándose en mecanismos automatizables. Si bien, no es posible automatizar todos los aspectos estudiados, en cada etapa del proceso se plantearon alternativas en búsqueda de automatizar o semi automatizar cada etapa.

Se desarrollo un proceso de migración teórico detallando cada etapa del proceso para luego poder ser implementado sobre cualquier tecnología origen y destino.

Se propuso inferir los orígenes de datos y exponerlos como servicios, implementar un refactor MVC sobre el sistema legacy e integrar con los servicios expuestos automatizando los posibles escenarios.

En cada etapa del proceso se estudiaron y presentaron herramientas sobre las cuales se implementaron los conceptos sobre un caso de prueba real.

Se implementó una app mobile con el objeto de ofrecer una visión general de las posibilidades de integración basadas en servicios.

La migración de sistemas legacy no es un trabajo trivial, es un escenario donde abundan los desafíos complejos que ponen en riesgo una migración exitosa. Implementar una migración con mecanismos automatizable es un tanto ambicioso, pero no menos motivador. Partiendo de dicha realidad, el hecho de disponer de un proceso de migración modelado sobre un enfoque práctico sobre el cual guiar la implementación del caso de prueba arrojó resultados muy alentadores.

Si bien migrar un sistema legacy hacia arquitecturas orientadas a servicios nos ofrece un panorama muy amplio de integración con nuevas tecnologías, es importante en ciertos casos (como el caso de estudio aplicado) que las interfaces del sistema legacy se mantengan intactas ya que los usuarios finales podrían no estar interesados en ningún tipo de cambio de las aplicaciones actuales. Es aquí donde se requieren de mecanismos que nos garanticen que nuestra migración backend (acceso a datos y lógica de negocios) no altere tales interfaces. Este escenario es el que se logra satisfacer con las herramientas de automatización de pruebas de interfaces de usuarios. Los escenario de automatización de pruebas de interfaces de usuarios nos permiten además, obtener resultados de calidad aceptable.

La evolución de herramientas relacionadas a migraciones ofrecen grandes ventajas como es el caso de las mencionadas durante el caso de estudio. La relación costo-tiempo entre la migración de un módulo o su re implementación de cero es muy marcada a favor de la migración. Para el caso de estudio propuesto, se siguieron los pasos teóricos del proceso de migración y con un esfuerzo acotado se logró un rápida migración hacía servicios.

Migrar lógica de negocios y encapsular acceso a datos mediante servicios implementados con tecnologías modernas nos brindan múltiples ventajas, entre las que se podría destacar:

- Accesibilidad web/mobile.
- Módulos más productivos y flexibles.
- Evolución rápida y económica.
- Reducir el nivel de acoplamiento.
- Clara definición de roles de desarrollo.
- Definición de seguridad más clara.
- Facilidad de testeo.
- Favorece la reutilización.
- Favorece el desarrollo en paralelo.
- Permitir fácil escalabilidad.
- Permitir un mapeo directo entre los procesos y los sistemas.
- Permitir un monitoreo preciso.
- Mejorar productividad y motivación del personal.

Por último, integrar nuevas tecnologías como la app mobile propuesta en el caso de estudio apoyándose en la arquitectura SOA resultante de la migración resultó muy simple y escalable dado que la misma no es la encargada del acceso a datos ni implementar lógicas de negocios. Y por el contrario, si se llegase a modificar parte de la lógica (implementación de los servicios) las aplicaciones que las consumen seguirían funcionando sin problema alguno.

Bibliografía

Artículos científicos

Rodriguez Alfredo, Márquez Antonio and Toro Miguel. "Gestión de la evolución del software. El eterno problema de los legacy systems". Ingeniería Simple. Departamento de Lenguajes y Sistemas Informáticos Escuela Técnica Superior de Ingeniería informática Universidad de Sevilla. Web. 10 Mar. 2016.

Hasselbring, Wilhelm. "Information System Integration 2000". Research Gate. Department of Computer Science. Web. 20 Mar. 2016.

Sneed harry, Wien AneCon. "Wrapping Legacy Software for Reuse in a SOA". Research Gate. Department of Computer Science. Web. 9 May. 2016.

Len, Erlikh. "Integrating Legacy Systems Using Web Services". Enterprise systems media 2003. Web. 18 May. 2016.

Grechanik Mark, Conroy Kevin. "Composing Integrated Systems Using GUI-Based Applications And Web Services", Web Services (ICWS) 2007 IEEE International Conference on, pp., 2007. Web. 30 May. 2016.

Good, Declan. " Transformación de Aplicaciones Legacy ".Club de Investigación Tecnológica Agosto 2002. Web. 25 Abr. 2016.

Bisbal Jesus, Lawless Deirdre , Richardson' Ray. " An overview of legacy information system migration". Research Gate. Department of Computer Science 1997. Web. 11 Jul. 2016.

Chikofsky E.J., Cross J.H.."Reverse engineering and design recovery: a taxonomy", IEEE Software (Volume: 7, Issue: 1, Jan. 1990). Web. 12 Jul. 2016.

Rosenberg, Linda. Software Re-engineering", Software Assurance Technology Center 1996. Web. 25 Jul. 2016.

Kazman Rick, Woods G. Steven, Carrière S. Jeromy. " Requirements for Integrating Software Architecture and Reengineering Models: CORUM II". Software Engineering Institute Carnegie Mellon University 1998. Web. 10 Ago. 2016.

Cuadrado Jesus, Avila-García Orlando. " Parametrización de las transformaciones horizontales en el modelo de herradura". Research Gate. Universidad Autónoma de Madrid 2012. Web. 20 Jul. 2016.

Atkinson Colin, Kühne Thomas. "Model-driven development: a metamodeling foundation", IEEE Software (Volume: 20, Issue: 5, Sept.-Oct. 2003). Web. 4 Sep. 2016.

Pastor Oscar, Molina Carlos. "Model-driven architecture in practice: A software production environment based on conceptual modeling". Research Gate. Universidad Politécnica de Valencia 2007. Web. 30 Jul. 2016.

Castillo Ricardo, Guzmán Ignacio. " Knowledge Discovery Metamodel-ISO/IEC 19506: A standard to modernize legacy systems". Research Gate. University of Castilla-La Mancha 2011. Web. 30 Jul. 2016.

Holley Kerrie, Channabasavaiah Kishore. "Migrating to a service-oriented architecture ". IBM Gobal Services. IBM Corporation Software Group 2004. Web. 5 Jul. 2016.

Wilkes Lawrence, Sprott David. " The architecture Journal:Understanding Service-Oriented Architecture ". CBDI Forum. CBDI Forum 2004. Web. 15 Jul. 2016.

Suárez Carlos, Suárez Jorge, Papaleo Mauricio. "ARQ-RFC-01-Pautas y recomendaciones para SOA (Service Oriented Architectures) Versión 0.91". Arquitectos de los Centros de Desarrollo 2006. Web. 20 Jul. 2016.

Pérez, Ramón. "Servicios Web: Orquestación y coreografías". Universidad de Oviedo 2007. Web. 29 Jul. 2016.

Howerton Jared. "Service-Oriented Architecture and Web 2.0", IT Professional (Volume: 9, Issue: 3, May-June 2007). Web. 4 Ago. 2016.

Bertolino, Antonia. " Software Testing Research and Practice". Research Gate. Lecture Notes in Computer Science · March 2003. Web. 20 Sep. 2016.

Rees Steve , Melnyk Roman B. " Best practices Tuning and monitoring database system performance ". IBM Gobal Services. IBM Corporation 2013. Web. 5 Oct. 2016.
Tanniru Mohan , Agarwal Ritu , Prasad Jayesh, Lynch John. " Risks of rapid application development". Research Gate. Communications of the ACM 43(11):1 · November 2000. Web. 28 Sep. 2016.

Burbeck Steve. " Applications Programming in Smalltalk-80 (TM): How to use Model-View-Controller (MVC)". ParcPlace Systems 1992. Web. 10 Nov. 2016.

Salvatierra Gonzalo, Mateos Cristian, Zunino Alejandro. "Towards a Computer Assisted Approach for Migrating Legacy Systems to SOA". Research Gate.
International conference on Computational Science and Its Applications 2012. Web. 10 May. 2017.

Hunold Sascha , Korch Matthias. " Transformation of Legacy Software into Client/Server Applications through Pattern-Based Rearchitecturing". Research Gate.
Computer Software and Applications, 2008. COMPSAC '08. 32nd Annual IEEE International. Web. 15 May. 2017.

Seaman, Carolyn. " Measuring and Monitoring Technical Debt". Research Gate.
Advances in Computers December 2011. Web. 20 Jun. 2017.

Libros

Heuvel, Willem-Jan Van den. *Aligning modern business processes and legacy systems: a component-based perspective*. Cambridge, MA: MIT Press, 2007. Web. 10 Abr 2016. ISBN-13: 978-0-262-22079-8.

Sommerville, Ian. *Software Engineering. International computer science series*. Harlow: Pearson/Addison-Wesley., 2005.. Web. 25 Jul 2016. ISBN-13: 978-0-13-703515-1.

Tesis

Bach, León. "Integración de sistema heredados utilizando web services". Universidad Ricardo Palma, 2006. Web. 10 Jun. 2016.

Otras referencias

"TIOBE estadísticas." The Importance of Being Earnest. Tiobe, 2016. Web. 10 May 2016. <http://www.tiobe.com/tiobe_index?page=index>.

"Inmosoft." Inmosoft Software para inmobiliarias. Inmosoft, 2017. Web. 20 May 2017. <http://www.inmosoft.com.ar>.

"Microsoft Web Services." Web Services. Microsoft, 2016. Web. 4 Jun 2016. <https://msdn.microsoft.com/en-us/library/ms950421.aspx>.

"Delphi." Embarcadero: Fast Cross-Platform App Development Software. Web. 19 Jun 2016. <https://www.embarcadero.com/es/products/delphi>.

"CodeTrigger." Herramienta de generación SOA. Web. 30 Jun 2016. <https://www.codetrigger.com>.

"IBM SOA." Integrate legacy systems into your SOA initiative. Web. 10 Dic 2016. <https://www.ibm.com/developerworks/library/ws-soa-legacyapps/>.

" Transformation of Legacy Systems." Software Reengineering at the Architectural Level: Transformation of Legacy Systems. Web. 2 Mar 2017. <https://www.researchgate.net/publication/228943033_Software_Reengineering_at_the_Architectural_Level_Transformation_of_Legacy_Systems >.

" Migration from Legacy Systems." Migration from Legacy Systems. Web. 2 Mar 2017. <https://www.unece.org/fileadmin/DAM/stats/documents/ece/ces/ge.50/2013/Topic_2_NZ_Legacy_Migration_Final.pdf >.

" SonarQube - Calero 2015. " Herramienta de análisis de deuda técnica . Web. 20 Jun 2017. <https://sonarqubehispano.org/pages/viewpage.action?pageId=4980855#DeudaTécnica-Cálculodeladeudatécnica >.

Apéndice

MVC

El patrón se basa en la separación del código en tres capas diferentes, acotadas por su responsabilidad, en lo que se llaman Modelos, Vistas y Controladores (ilustración 59).

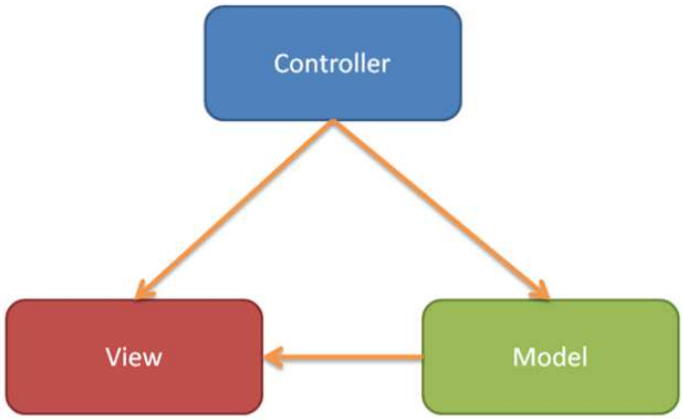


Ilustración 59: (Esquema MVC clásico)

- Modelo (Model)
  Es la capa donde se trabaja con los datos, por tanto contendrá mecanismos para acceder a la información y también para actualizar su estado. Los datos los tendremos habitualmente en una base de datos, por lo que en los modelos tendremos todas las funciones que accederán a las tablas y harán los correspondientes selects, updates, inserts, etc.
- Vista (View)
  Las vistas, como su nombre nos hace entender, contienen el código de nuestra aplicación que va a producir la visualización de las interfaces de usuario, o sea, el código que nos permitirá renderizar los estados de nuestra aplicación. En las vistas nada más tenemos los códigos que nos permite mostrar la salida.
- Controlador (Controller)
  Contiene el código necesario para responder a las acciones que se solicitan en la aplicación, como visualizar un elemento, realizar una compra, una búsqueda de información, etc.

UI Automation VS

Guía básica para crear pruebas de IU codificadas con Visual Studio 2012.

Paso 1

Crear un proyecto de prueba de IU codificada.

Las pruebas de IU codificadas deben incluirse en un proyecto de prueba de IU codificada. Si aún no tiene un proyecto de este tipo, crear uno. En el Explorador de soluciones**,** en el menú contextual de la solución**,** elija Agregar**,** Nuevo proyecto y seleccione Visual C#**.** A continuación**,** elija Probar**,** Prueba de IU codificada**.**

- No se ven las plantillas de proyecto Prueba de IU codificada**.**

Puede que esté usando una versión de Microsoft Visual Studio 2012 que no admite pruebas de IU codificadas. Para crear pruebas de IU codificadas, debe usar Visual Studio Enterprise.

Paso 2

Agregue un archivo de prueba de IU codificada.

Si acaba de crear un proyecto de IU codificada, el primer archivo de CUIT se agrega automáticamente. Para agregar otro archivo de prueba, abra el menú contextual del proyecto de prueba de IU codificada, elija Agregar y, a continuación, Prueba de IU codificada (ilustración 60).

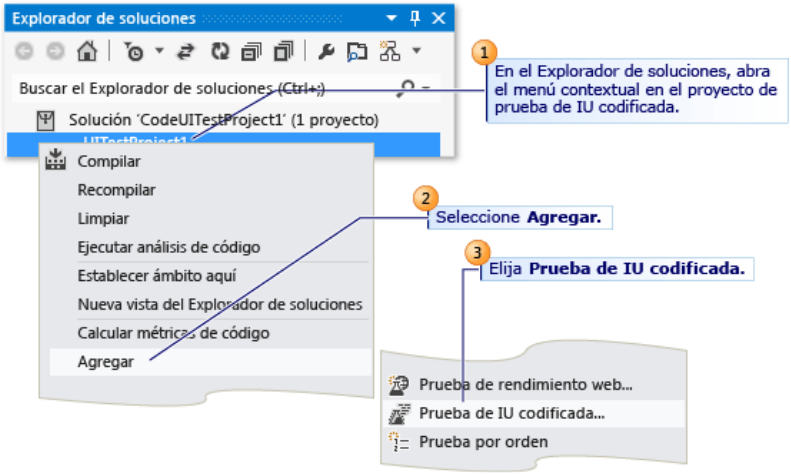


Ilustración 60: (Explorador de soluciones)

En el cuadro de diálogo Generar código para prueba de IU codificada, elija Grabar acciones, editar asignación de IU o agregar aserciones (ilustración 61).

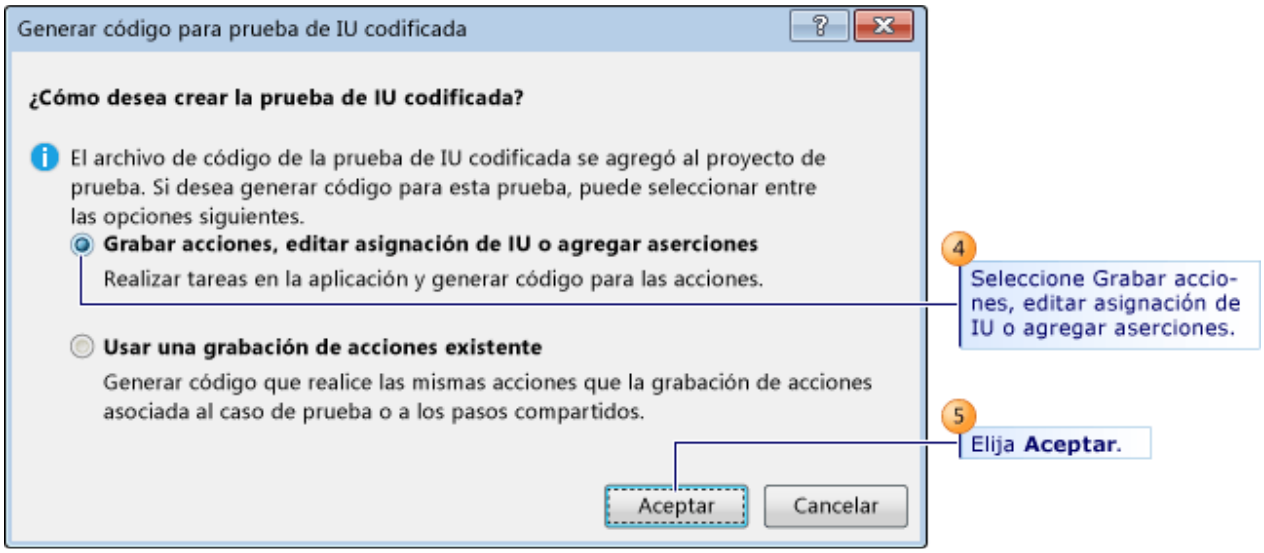


Ilustración 61: (Generador de código 1)

Aparece el generador de pruebas de IU codificadas y Visual Studio se minimiza (ilustración 62).

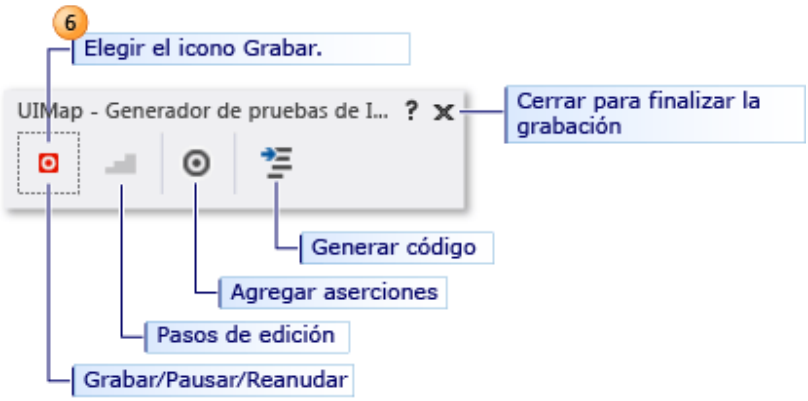


Ilustración 62: (Generador de código 2)

Paso 3

Grabe una secuencia de acciones.

Para iniciar la grabación, elija el icono Grabar. Realice las acciones que desea probar en la aplicación, incluido el inicio de la aplicación si es necesario. Por ejemplo, si está probando una aplicación web, puede iniciar un explorador, navegar al sitio web e iniciar sesión en la aplicación. Para pausar la grabación, por ejemplo, si tiene que encargarse del correo entrante, elija Pausar.

Para eliminar acciones grabadas por error, elija la opción para editar acciones.

Para generar código que replique las acciones, elija el icono Generar código y escriba un nombre y una descripción para el método de prueba de IU codificada.

Compruebe los valores de los campos de la interfaz de usuario, como cuadros de texto.

Elija Agregar aserciones en el generador de pruebas de IU codificadas y, a continuación, seleccione un control de IU en la aplicación en ejecución. En la lista de propiedades que aparece, seleccione una propiedad, por ejemplo, Text en un cuadro de texto. En el menú contextual, elija Agregar aserción. En el cuadro de diálogo, seleccione el operador de comparación, el valor de comparación y el mensaje de error.

Cierre la ventana de aserción y elija Generar código (ilustración 63).

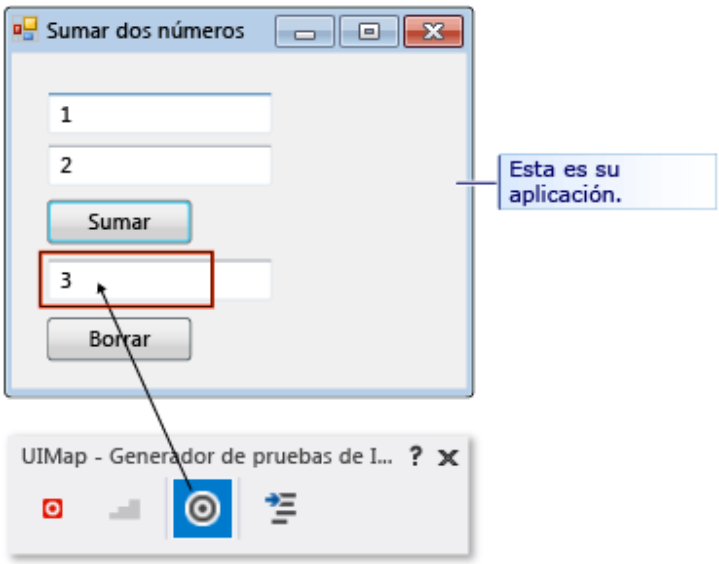


Ilustración 63: (Generador de código 3)

Herramienta de análisis de deuda técnica

A continuación, se resumen el artículo de [Calero 2015] donde se ejemplifica el estudio de la deuda técnica con la herramienta SonarQube.

El cálculo de la deuda técnica está basada en la metodología Software Quality Assessment based on Lifecycle Expectations (SQALE). SQALE es una metodología que fue desarrollada por Inspearit y fue cedida como código abierto.

La implementación de SQALE de SonarQube se basa únicamente en reglas y evidencias. Esto significa que si se desea gestionar toda la deuda técnica con SQALE,

primeramente se necesitará habilitar las reglas en el repositorio común de SonarQube que etiquetan:

- Los bloques duplicados.
- Las pruebas unitarias falladas.
- Ramas cubiertas por las pruebas unitarias insuficientes.
- Densidad de comentarios insuficientes.
- Cobertura de líneas cubierta por pruebas unitarias insuficientes.
- Pruebas unitarias omitidas.

Estas reglas están en el repositorio común de SonarQube porque son comunes a todos los lenguajes. Una vez habilitadas se puede realizar seguimiento de cada fallo como una evidencia y realizar seguimiento de la deuda técnica, que el modelo SQALE mide en días.

Cálculo de la deuda técnica

En el cuadro de mando por defecto (ilustración 64), se puede ver un resumen de la deuda técnica del proyecto con el widget "Evidencias y Deuda Técnica":

Ilustración 64: (Resumen deuda técnica)

Esa medida en días se realiza sumando la deuda técnica asociada a cada evidencia, que puede verse en el bloque de evidencias cuando se abre un fichero de código (ilustración 65).

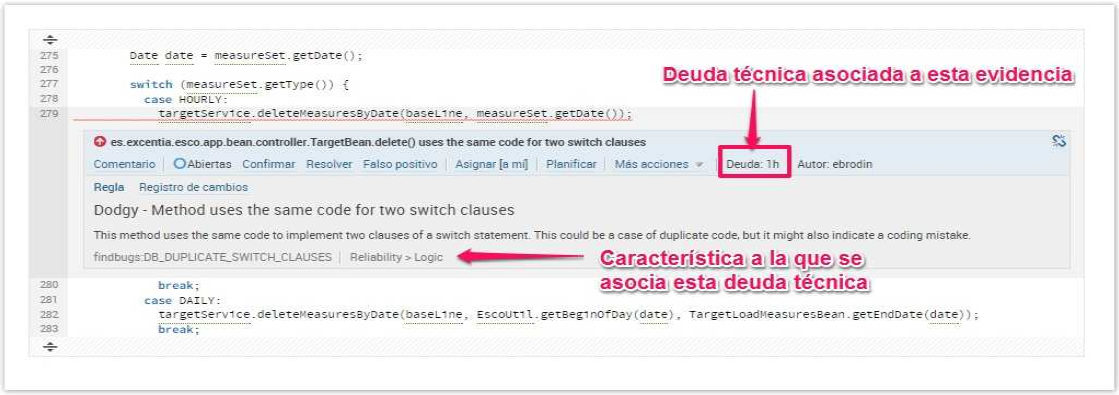


Ilustración 65: (Detalles deuda técnica)

La deuda técnica de cada evidencia se establece a nivel de la regla. Si se tiene el plugin comercial SQALE, se puede ajustar la estimación de cada regla (al hacer esto, se está editando el Modelo de Análisis SQALE - el coste asociado a la resolución de cada regla). Sin embargo, estas estimaciones están realizadas por profesionales expertos de forma que no se requiera modificarlas.

Inmosoft

Hace más de una década nació Inmosoft, un proyecto personal motivado por mi ingreso a la universidad y las ganas de llevar a la práctica lo que iba aprendiendo. Inmosoft es un software especializado en la gestión integral para inmobiliarias, abarcando desde simples circuitos hasta los más avanzados y productivos para el sector inmobiliario.
Con el correr de los años, Inmosoft fue creciendo y sumando tanto clientes como funcionalidades. Hoy ya finalizando la carrera de Licenciatura en Sistemas, Inmosoft es un producto estable y afianzado en el mercado pero con las limitaciones tecnológicas que implican a un software comenzando hace muchos años atrás.

Dado que Inmosoft está implementado en Delphi 6, una tecnología prácticamente obsoleta, y se requiere de ciertas funcionalidades disponibles tanto para aplicaciones móviles como para aplicaciones web la prueba de concepto que permitirá aplicar las contribuciones de esta tesis se realizará sobre el sistema Inmosoft.

Detalles funcionales

Inmosoft es un sistema informático de gestión integral para inmobiliarias.
Inmosoft simplifica cada uno de los aspectos relacionados a la gestión de agenda de personas, cartera de inmuebles, contratos de alquiler y liquidaciones a propietarios haciendo que el trabajo diario del corredor inmobiliario sea 100% productivo cada día.
Olvídese de los papeles, fichas, confecciones a mano de recibos, difícil acceso a su información y todos los problemas relacionados a esto.
Olvídese de calcular importes, saldos, punitorios y deudas a mano. Inmosoft realiza todo de manera automática en segundos.
Cambie su imagen frente a sus clientes al informatizar sus servicios y agilice todas sus labores diarias gracias a nuestro sistema.

Capturas

Las ilustraciones 66, 67, y 68 presentan capturas del sistema Inmosoft en ejecución.

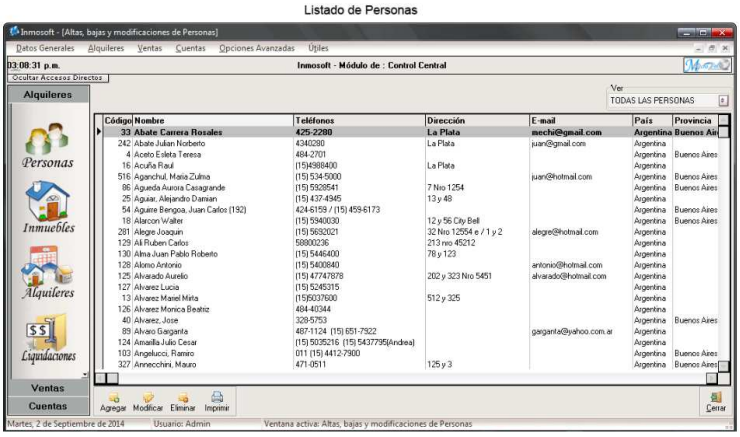


Ilustración 66: (Listado Personas Inmosoft)

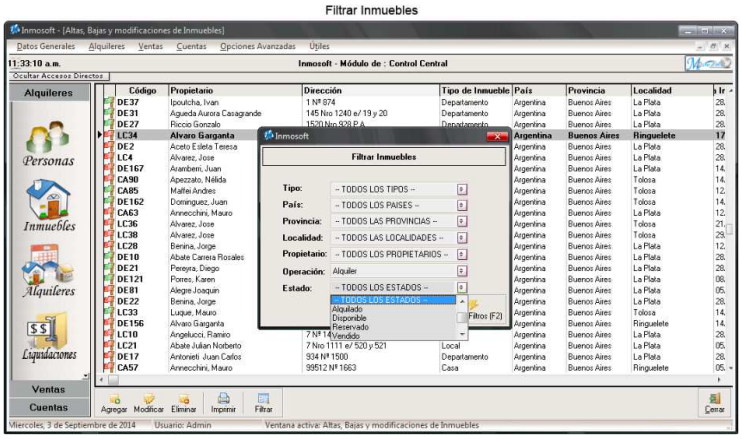


Ilustración 67: (Filtrador Inmuebles Inmosoft)

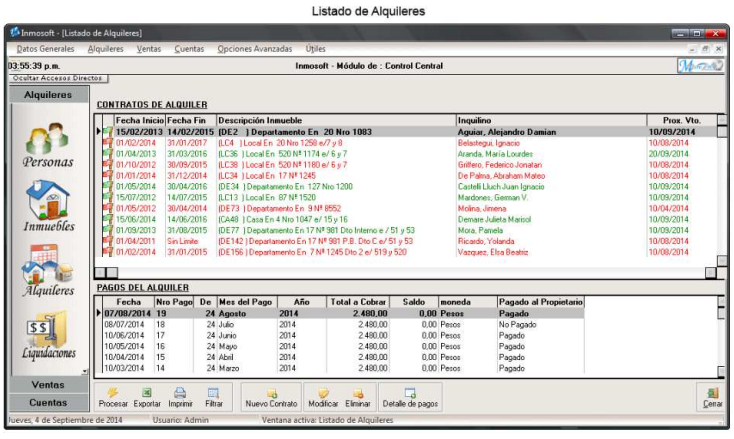


Ilustración 68: (Alquileres Inmosoft)

Más detalles: www.inmosoft.com.ar/caracteristicas/index.html

Detalles técnicos

Base de datos: DBMS- SQL Server (2000 - 2008) | Modalidad Online - Offline.
Entorno de desarrollo: RAD - Bordland Delphi 6.
Conexión con BD: BDE - Borland Batabase Engine
Versión actual: 5.0
Web: www.inmosoft.com.ar
Mail: info@inmosoft.com.ar
Licencia: Shareware.
Autor: Mauro Barzola (100 %)