

PyAfipWs: facilitando, extendiendo y liberando los Servicios Web de AFIP (Factura Electrónica y otros)

Autor: Mariano Reingart
reingart@gmail.com

Resumen: PyAfipWs es un proyecto de software libre para brindar interfaces, herramientas y aplicativos multiplataforma a servicios web inicialmente de Factura Electrónica de la AFIP (Argentina). Actualmente soporta adicionalmente Trazabilidad de Granos (Carta de Porte), Depositario Fiel (Aduana), Consulta de Operaciones Cambiarias (Moneda Extranjera), Remito Electrónico (COT - ARBA) y Trazabilidad de Medicamentos (ANMAT). Para el mismo se han creado y mejorado herramientas técnicas de soporte para comunicación con servicios web (PySimpleSOAP compatible con los diversos tipos de servidores y lenguajes de programación), manejo simple de XML, generación de PDF adaptable por el usuario (PyFPDF) y extensión para compilación automática de instaladores. También es un ejemplo de implementación de interfaces con otros lenguajes (DLL símil OCX embebible en Windows), soporte para archivos de intercambio (texto fijo, DBF, XML) e interconexión con distintas bases de datos. Posibilita extender aplicaciones legadas, con un modelo de negocios de evaluación gratuita y soporte comercial pago que busca un equilibrio entre la filosofía del software libre, las necesidades de los clientes y la sustentabilidad económica, presentando una alternativa descentralizada a aplicativos gubernamentales, desarrollados desde la comunidad sin necesidad de recursos públicos ni consideraciones especiales.

Palabras clave: python, webservices, SOAP, PDF, component object model, java, .net, visual basic, visual fox, cobol, interoperabilidad, gobierno electrónico, administración tributaria, ERP, CRM, SCM

1. Introducción:

PyAfipWs [1] es una interfaz de software libre a los Servicios Web de la AFIP [2], desarrollada en Python. Nacida en el año 2008 de un intercambio de correos electrónicos, en la lista de correo del Grupo de Usuarios de Python Argentina (PyAr), consultando sobre unos problemas con Python y webservices. Fue creciendo una librería que se adaptara a los requerimientos y ambientes de AFIP (bastantes variados y ninguna herramienta en python funcionaba completamente).

La interfaz fue "inspirada" en los ejemplos oficiales de la AFIP [3] para PHP, ya que eran los más simples de entender y más cercanos a un lenguaje como Python. También había ejemplos para Java pero estaban incompletos (tienen algunas dificultades con el tema de dependencias e incompatibilidades con XML/SOAP) y en .NET (que estaban más completos pero solo para Windows). Igualmente, estos dos

ejemplos son bastante complicados, generan código de complejo mantenimiento ("artefactos"), y no están del todo documentados (entre otros temas).

Se tomaron características de algunas bibliotecas de PHP sobre manejo simple de XML, SOAP y PDF, por ser sencillas e intuitivas, que luego dieron lugar a su contraparte en Python.

Inicialmente se desarrolló para los web services de autenticación (WSAA: solicitud de autenticación mediante firma digital) y factura electrónica (WSFE), pero con el correr del tiempo fue creciendo y agregando otros servicios de AFIP (Administración Federal de Ingresos Públicos, ente recaudador de impuestos nacionales de Argentina), como ser:

- WSFEv1: factura electrónica mercado interno ("versión 1")
- WSBFE: bienes de capital - bono fiscal electrónico
- WSFEXv1: facturas electrónicas de exportación
- WSMTXCA: factura electrónica con detalle (proyecto "Matrix")
- WSCTG: código de trazabilidad de granos
- WSDigDepFiel: depositario fiel (aduana)
- WSCOC: código de operaciones cambiarias (compra de moneda extranjera)

A su vez, la biblioteca también se ha extendido a servicios web de otros organismos, cubriendo varias necesidades de los sistemas de gestión de nacionales:

- COT: Código de Operación de Traslado: remito electrónico de ARBA (agencia de recaudación de impuestos de la Provincia de Buenos Aires)
- Trazamed: Trazabilidad de Medicamentos PAMI / ANMAT (Administración Nacional de Medicamentos)

El proyecto ha madurado y algunas librerías desarrolladas o adaptadas han sido liberadas separadamente, para luego ser integradas a web2py [15] y así tener un marco de trabajo completo para desarrollar aplicaciones web de gestión:

- PySimpleSoap [7]: para manejo simple y completo de webservices
- PyFPDF [8]: para generación de PDF de manera más fácil y fluida..

Gracias a extensiones como PythonCOM [9] (win32), fue posible utilizar la interfaz desde lenguajes cerrados y/o aplicaciones legadas, habilitando una migración parcial al software libre, lo que originó una comunidad relacionada. Al mismo tiempo, esta solución posibilitó un modelo de negocio dual (basado en la licencia GPLv3), que sustentó comercialmente el desarrollo y mantenimiento del proyecto (con una evaluación gratuita pero con soporte pago para producción).

En su conjunto, la solución ha permitido ofrecer una alternativa seria, flexible y abierta a los ejemplos y aplicativos oficiales, difundiendo las características y ventajas del software libre. En las siguientes secciones se describe las motivaciones, comparación con otras soluciones (estado del arte), implementación, modelo de negocio y conclusiones finales.

2. Motivación: ¿por que otra implementación de webservices SOAP? (y en Python)

Al comenzar el proyecto para consumir los webservices de AFIP (fines de 2008), se probaron varias implementaciones de SOAP [10]:

- SOAPPy: luego de pruebas iniciales con algunos webservices, falló en varios ambientes (había que trabajar tanto con Java Axis y .NET, ya que AFIP usa ambas plataformas, en varias de sus versiones). El esfuerzo para corregir los inconvenientes era demasiado grande, mayormente relacionado con la generación de XML y espacios de nombres / prefijos.
- ZSI: no se pudo siquiera compilar esta biblioteca, algo que no era prometedor si se iba a desplegar la aplicación en muchos ambientes (tampoco había mucho tiempo para dedicar a este tema).
- suds, soaplib, etc.: no fueron analizadas, ya sea por su carácter experimental en ese momento, y debido a la complejidad que todas ellas exhibían.

Enfrentados a estas dificultades, especialmente viendo cuan complejo era implementar / adaptar / mantener un cliente simple de webservices SOAP, se decidió iniciar una biblioteca SOAP propia, con el objetivo de que sea muy simple pero sin perder funcionalidad.

Como los ejemplos para consumir webservices oficiales (AFIP) estaban programados en PHP, y ya que las extensiones SOAP [11] y Simple XML [12] para dicho lenguaje parecieron muy intuitivas y fáciles de usar, nuestra biblioteca se asemeja a dichas extensiones (incluyendo el nombre de los métodos y funcionalidad general).

Con este enfoque, se logró escribir un cliente SOAP en pocas líneas (originalmente menos de 100, y se ha mantenido en ese orden de magnitud a lo largo de los años), permitiendo que sea personalizado y adaptado para diferentes servidores. Con el tiempo, la biblioteca se estabilizó y obtuvo más características, por lo que comenzó a ser útil para otras personas.

Una de las primeras características agregada fue el soporte de múltiples transportes: urllib, httplib2, libcurl. Cada una de ellas habilitó el uso de la biblioteca en distintas plataformas y ambientes especiales, como verificación de certificados y soporte de varios tipos de servidores proxy, incluyendo autenticación no estándar como la de MS ISA server.

Otra característica importante es el análisis dinámico de descripciones de servicio en WSDL, permitiendo extraer automáticamente los elementos principales de un webservice (ubicación, acciones, métodos, parámetros), lo que simplifica la escritura de clientes. Se contemplan los casos especiales como prefijos, espacios de nombre y orden de parámetros, cuyos requisitos varían, por ej., en .NET y Java Axis. A diferencia de otras implementaciones, el análisis dinámico permite mayor flexibilidad

(por ejemplo, modificar o corregir WSDL publicados incorrectamente, tema que ha sido necesario), logrando mayor velocidad y eficiencia (utilizando tipos de datos simples que pueden ser serializados de manera fácil, sin necesidad de crear clases o artefactos estáticos).

La biblioteca ha demostrado cubrir casos de uso práctico, habiendo reunido alrededor de 20 contribuidores internacionales, incluyendo ajustes y mejoras importantes, pudiendo citar un comentario para ilustrar la diferencia respecto a otros proyectos similares pero de objetivos más tradicionales [4]:

Yes, this appears to work with the new client.py. I'm trying to migrate from suds right now -- the suds client uses ~900MB memory and ~20 minutes to parse the NetSuite wsdl. With this client.py file, pysimplesoap uses 25 MB memory and parses the wsdl in ~10 seconds!

3. Comparación con otras bibliotecas y soluciones

3.1. Servicios Web

Siguiendo la guía publicada por Doug Hellmann en la Revista Python Magazine [6], se presenta un análisis de la biblioteca (puede compararse con ZSI y soaplib en el artículo original):

- **Instalación:** Muy simple, sin compilación requerida. Simplemente se ejecutan los archivos (python puro). Opcionalmente, se pueden instalar algunas dependencias (httplib2 para conexiones encriptadas, socks para soporte de proxy, pycurl para ISA server). Esto contrasta con otras bibliotecas que requieren dependencias avanzadas como lxml/pyxml (algunas complejas de compilar en varios ambientes)
- **Características completas:** solución total integrada con SoapClient & SoapDispatcher (usado sobre web2py o de manera independiente / embebida, similar a la biblioteca XML RPC). Permite exponer los mismos métodos (con el mismo código fuente sin modificaciones) usando diferentes protocolos, por ej., con las herramientas incorporadas de web2py (actualmente rss, json, jsonrpc, xmlrpc, jsonrpc, amfrpc, amfrpc3). Adicionalmente, web2py fácilmente permite exponer páginas web como una interfase de usuario para pruebas (con una lista de operaciones disponibles, con la respectiva ayuda / mensajes xml de ejemplo).
- **Interoperabilidad:** esta biblioteca fue diseñada para trabajar con diferentes dialectos SOAP (como Java AXIS, .NET 2.0, JBoss, etc.). Incluye soporte para ambas especificaciones SOAP (1.1 y 1.2), varios generadores y parsers de XML (prefijos de nombre, atributos vacíos, esquemas importados), etc.
- **Elegancia:** Esta biblioteca sigue las directivas de python, en el sentido que no es obligatoria una descripción de servicio XML (WSDL), simplemente se usa código python puro, sin necesidad de clases/artefactos autogenerados o repetitivos. Esto es una distinción respecto a otras bibliotecas (citando el artículo original: "*SOAP toolkits tend to fall in one of two camps: Those that generate source from a WSDL file and those that generate a WSDL*")

document from source"). Este enfoque permitió soportar la mayor cantidad de características python, habilitando un desarrollo rápido y flexible de webservices. Adicionalmente, el WSDL puede ser dinámicamente generado o analizado (para introspección). Además, se administran prácticamente todos los aspectos de la comunicación con webservices, en la mayoría de los casos devolverá errores útiles, y en general, es de uso intuitivo.

- **Frescura:** esta es una nueva biblioteca nacida para resolver los entornos recientes de webservices.
- **Documentación:** el código es muy simple y documentado, y se provee una wiki para documentación colaborativa.
- **Complejidad de Despliegue:** para testeo, un servidor básico de HTTP es provisto (SoapServer), para producción, puede ser usada sobre web2py (infraestructura WSGI, Apache, HTTPS, etc).
- **Complejidad de Empaquetamiento:** la redistribución es simple y fácil (ya que extensiones compiladas en C no son obligatorias), se puede usar py2exe o easy_install, e incluso los fuentes pueden ser usados para redistribución.
- **Licenciamiento:** esta biblioteca está publicada bajo la licencia LGPL, por lo que puede ser integrada a software propietario o proyectos abiertos.
- **Pruebas:** se ha desarrollado una infraestructura de Unittest para probar cada ambiente y cada incidente reportado.
- **Rendimiento:** si bien no existen pruebas extensivas sobre el tema, como se hace poco overhead y se usa xml.dom.minidom (Python Standard Library), la velocidad para analizar mensajes xml es irrelevante en la mayoría de los escenarios. Se provee un cache simple de WSDL (serializando la descripción), principalmente para evitar descargar los archivos xml en cada invocación.

La siguiente tabla muestra una comparación con los ejemplos oficiales (Java, .Net, PHP [3]) publicados por AFIP:

Entorno	Java	.NET	PHP	Python
WSAA (LOC aprox.)	300	500***	100**	100
WSFE (LOC aprox.)	N/A ****	300***	140**	150
SDK (Windows)	73 MB	350 MB	17 MB	10 MB
Runtime	15 MB	22 MB	N/A	N/A
Instalador all-in-one	N/A	N/A	posible	2.5 MB
Fácil de embeber	limitado*	limitado*	no*	sí
Multiplataforma	sí	no	sí	sí
Software Libre	sí	no	sí	sí

Notas:

* Comparado con Python (COM, cantidad código, tamaño instalador, runtime, etc.)

** No tiene prácticamente líneas en blanco, comentarios, documentación ni licencia

*** Incluye breve ayuda de uso para el usuario dentro del código. Cantidades para C#

**** No compatible (sin ejemplo oficial disponible para Java)

3.2. Generación de PDF

La biblioteca PyFPDF permite generar archivos PDF con el contenido de plantillas y todo tipo de documentos. Utiliza un enfoque similar a las interfaces para webservices, por lo que su uso es muy simple, totalmente automatizado (sin intervención del usuario) y no necesita herramientas externas (ni impresoras PDF ni tipografías para el código de barras, ni otras bibliotecas o componentes compilados). Esta basada en FPDF [16], una biblioteca con primitivas de bajo nivel para generar de manera simple y fácil documentos PDF. Es similar al canvas gráfico de ReportLab [17] (generador tradicional de PDF para Python), pero tienen métodos para celdas "fluidas" ("flowables" que pueden expandirse a múltiples filas, páginas, tablas, columnas, etc.), con varios métodos ("hooks") que pueden ser redefinidos para controlar encabezados, pie de página, etc. Originalmente desarrollada en PHP varios años atrás (como una alternativa libre a bibliotecas C propietarias), ha sido portada a muchos lenguajes, incluyendo ASP, C++, Java, PL/SQL, Ruby, Visual Basic, y Python.

A diferencia de ReportLab, es mucho más compacta (1 archivo), y no tiene soporte incorporado para gráficos y widgets (pero se pueden importar PNG o JPG o usar ejemplos externos), no contempla una "flexible page layout engine" como Reportlab PLATYPUS (pero tiene celdas y puede hacer columnas, capítulos, etc.) y no se basa en XML u Objetos como Gerdal Reports, Jasper Reports o similares (posee un método WriteHTML para reportes simples, y reportes complejos se pueden programar con primitivas). Comparada con otras soluciones, esta biblioteca debería ser más simple de usar y adaptar para los tipos de documentos más comunes (sin necesidad de usar motores de diseño de página, hojas de estilo, historias), y posee un control total del documento PDF (incluyendo características avanzadas y extensiones). Es pequeña (un simple archivo .py <77K) y no necesita compilación.

Adicionalmente, se desarrollo el programa designer.py para modificar visualmente los diseños de las plantillas, para posibilitar a los usuarios adaptar sus plantillas sin necesidad de conocimientos de programación (tema vital para el presente proyecto, ya que el público contemplado incluye usuarios finales). Los archivos de entrada son planillas CSV describiendo el diseño. Al abrirlo el diseñador muestra la plantilla con los elementos donde se ubicarán los futuros valores de cada plantilla. A modo de ejemplo se muestra un pantallazo del Diseñador Visual (Figura 1) con el elemento "logo" seleccionado, editando sus propiedades, y su correspondiente muestra (Figura 2):

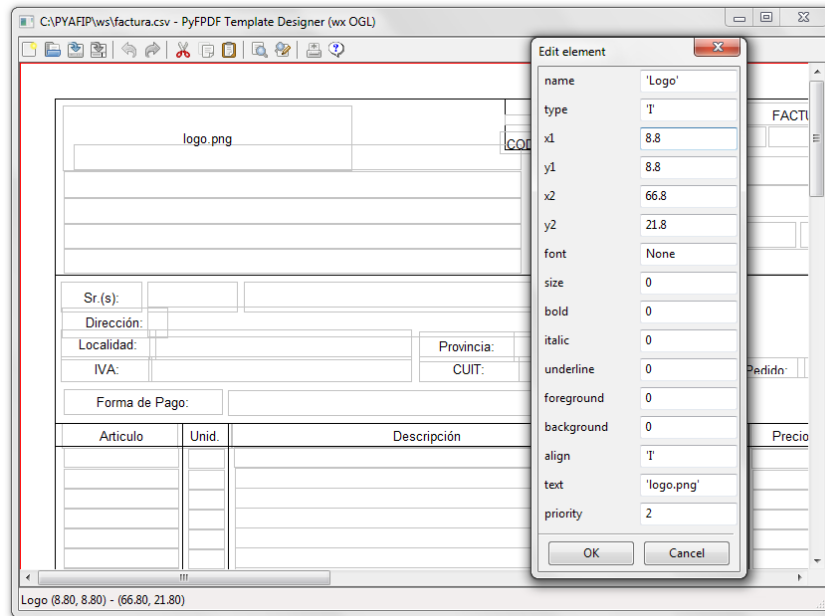


Figura 1: Diseñador de plantillas en PDF


		<input checked="" type="checkbox"/> X <small>copiar</small>		FACTURA <small>Original</small> <small>Página</small>			
Sr.(s): Dirección: Localidad: IVA: Forma de Pago:		Provincia: CUIT:		Pedido: Remito N°:			
Artículo	Unid.	Descripción	Cantidad	% Bonif.	Precio	IVA	Importe
Desuento:		Subtotal:		Sub Total Neto:			
Observaciones AFIP							
Transporte: C.A.E. N° 61101021770094 Fecha Vto. CAE: 31/12/2010						I.V.A. 21% I.V.A. 10,5% Total:	

Figura 2: muestra de factura generada con PyFPDF

3.4. Generación de Instaladores

Si bien la plataforma Python incluye herramientas para compilar y empaquetar software, es un proceso manual y no genera archivos autoextraíbles fácilmente distribuibles por Internet. En windows, al principio se utilizó 7-zip para generar dichos archivos, con un script .BAT, y luego se mejoró el proceso con Nullsoft Scriptable Install System (NSIS), con lo cual se eliminó el script intermedio y se automatizó prácticamente todo el proceso, con lo cual para generar los nuevos instaladores es posible utilizar una única línea de comando:

```
c:\python25\python.exe setup_wsfev1.py py2exe
```

Esto se logró gracias a desarrollar nsis.py, una extensión para el sistema de empaquetamiento y distribución de Python, que incorpora automáticamente la configuración necesaria (extrayendo por ejemplo número de versión desde los archivos fuentes), calculando dependencias y buscando archivos requeridos.

4. Implementación

A modo de ejemplo, se mostrará como se autoriza una factura electrónica, para analizar la implementación de la biblioteca.

Primero se debe solicitar un ticket de acceso (utilizando un certificado y clave privada tramitadas previamente), que permitirá posteriormente utilizar el servicio web de factura electrónica:

```
from pyafipws import wsaa

# crear ticket acceso, firmarlo y llamar al ws
tra = wsaa.create_tra()
cms = wsaa.sign_tra(tra, "homo.crt", "homo.key")
ta_xml = call_wsaa(cms, wsaa.WSAAURL)
```

Detrás de escena se usa M2Crypto (que es el enlace de OpenSSL para encriptación en Python), necesario para firmar la solicitud de acceso en XML.

Del ticket de acceso se extrae el Token y Sign (usando SimpleXMLElement para analizar y convertir el XML en una estructura de objetos y datos python), que contienen los códigos de seguridad requeridos en los otros webservices:

```
# procesar el xml
ta = SimpleXMLElement(ta_xml)
token = str(ta.credentials.token)
sign = str(ta.credentials.sign)
```

Una vez obtenida la autenticación para acceder a los servicios web, se procede a solicitar autorización para emitir una factura electrónica. Para ello se crea una conexión con el servidor y se llama remotamente al procedimiento de autorización:

```
from pyafipws import wsfe

# crear cliente SOAP
client = wsfe.SoapClient(wsfe.WSFEURL,
                        action=wsfe.SOAP_ACTION,
                        namespace=wsfe.SOAP_NS)

# autorizar factura electronica (obtener CAE)
res = wsfe.aut(client, token, sign,
               CUIT=20234567393, tipo_cbte=1, punto_vta=1,
               cbt_desde=1, cbt_hasta=1,
               tipo_doc=80, nro_doc=23111111113,
               imp_total=121, imp_net=100, impto_liq=21)

print res['cae'], res['motivo']
```

Si todo funciona bien, esta llamada devolverá el CAE (Código de Autorización Electrónico), necesario para confeccionar la factura electrónica.

4.1. Aplicaciones legadas

Más allá de las aplicaciones en Python, esta biblioteca es compatible con lenguajes como Visual Basic, ASP, Fox Pro, Cobol, Delphi, Genexus, PowerBuilder, PHP, .Net, Java, ABAP (SAP), etc. y cualquier lenguaje/aplicación que pueda crear objetos COM (automatización) en Windows (por ej. Excel o Access).

Esto se logra fácilmente utilizando PythonCOM [9] (parte de las extensiones win32), envolviendo una clase común de python para que pueda ser expuesta a otras aplicaciones, definiendo los métodos y atributos públicos, el nombre publicado, id de clase (registro), por ej.:

```
class WSAA:
    "Interfase para el Webservice de Autenticación"
    _public_methods_ = ['CreateTRA', 'SignTRA', 'CallWSAA']
    _public_attrs_ = ['Token', 'Sign', ...]
    _readonly_attrs_ = _public_attrs_
    _reg_progid_ = "WSAA"
    _reg_clsids_ = "{6268820C-8900-4AE9-8A2D-F0A1EBD4CAC5}"
```

Una vez registrada la interfaz, es posible llamarla desde cualquier otra aplicación con esta tecnología, por ej, en Visual Basic sería:

```
Set WSAA = CreateObject("WSAA")
tra = WSAA.CreateTRA()
cms = WSAA.SignTRA(tra, "homo.crt", "homo.key")
ta = WSAA.CallWSAA(cms, url)

Set WSFE = CreateObject("WSFE")
WSFE.Token = WSAA.Token
WSFE.Sign = WSAA.Sign
WSFE.Cuit = "3000000000" ' CUIT del emisor

ok = WSFE.Conectar(url)

cae = WSFE.Aut(id, presta_serv, tipo_doc, ...
             imp_tot_conc, imp_netto, imppto_liq, ...)
```

Este tema fue muy útil y permitió a muchas aplicaciones contemplar estas nuevas funcionalidades (webservices, encriptación, etc.) con modificaciones menores, que de otro modo hubieran sido muy difíciles o imposibles (lenguajes legados, ambientes cerrados, etc.). Se tuvieron que tomar algunas consideraciones, como simplificar el manejo de errores (algunos lenguajes no soportan capturar las excepciones), y simplificar los parámetros y tipos de datos (algunos lenguajes no soportan completamente trabajar con listas o más de 20 parámetros, por ejemplo).

Si bien la interfaz COM es muy útil para aplicaciones relativamente modernas, todavía hay lenguajes o entornos de muy difícil acceso, donde prácticamente la única forma de interoperabilidad son los archivos de texto.

Viendo que lenguajes como COBOL manejan archivos con campos de longitud fija (y esto ya era soportado y es el mecanismo de intercambio del aplicativo oficial de AFIP: RECE de SIAP), se tomó ese camino, y con Python fue bastante directo.

La interfaz incluye herramientas como RECE.PY y RECEX.PY que por línea de comandos reciben y procesan los archivos de entrada, guardando los resultados en archivos de salida. Esto es controlado con un archivo de configuración (RECE.INI) que utiliza define las URL, certificados y rutas a utilizar.

4.2. PyRece: aplicativo de SIAP libre

Viendo algunas dificultades del Aplicativo oficial RECE [13] o los servicios en línea para hacer facturas electrónicas (funcionalidad limitada, demoras o complejidades al no poder automatizar el proceso, pérdida de sesión por tiempo de espera agotado, falta de flexibilidad al crear las salidas y reportes, etc., sobre todo para los contribuyentes que no poseen sistema de gestión), se desarrolló un aplicativo visual (de "escritorio", Figura 3) para facilitar y extender las posibilidades brindadas por el aplicativo oficial de AFIP:

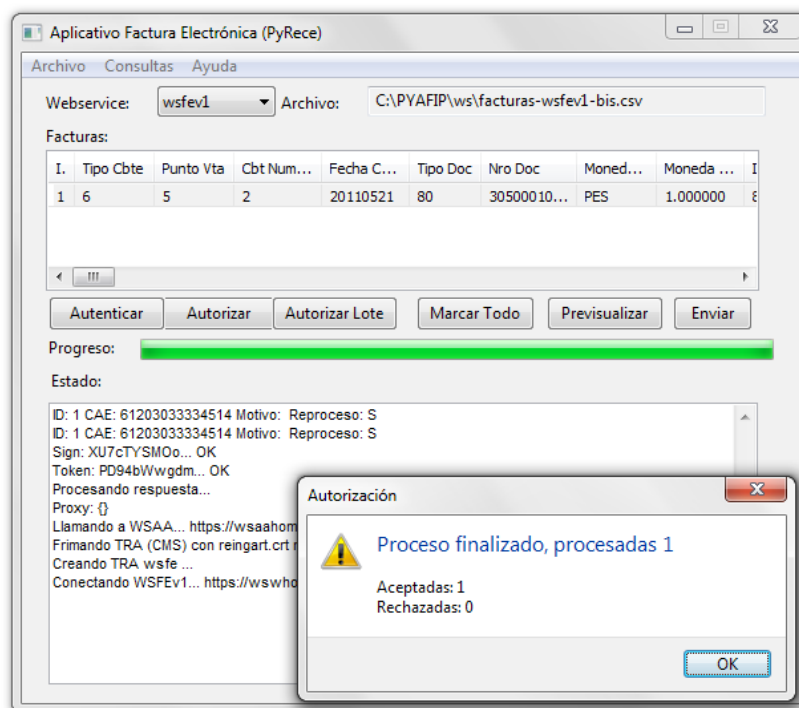


Figura 3: Captura de Pantalla de Aplicativo PyRECE

La interfaz del usuario es una pantalla simple pero contempla:

- Elección del webservice a utilizar (WSFEv1, WSFEXv1)
- Lectura de planilla de calculo CSV / XML / JSON / TXT de ancho fijo, etc.
- Autenticación y Autorización on-line en el momento
- Generación del PDF personalizable (textos, logos, lineas, etc.) con la imagen de la factura, con total control en el diseño y contenido de la factura.
- Envío por mail del PDF a los clientes, con un breve mensaje configurable

Todas estas funciones no están disponibles (en su conjunto) en el aplicativo o sitio web de AFIP (algunas pueden realizarse con limitaciones o mediante el sitio web de AFIP, pero no de una forma totalmente ágil y transparente).

Esta aplicación a su vez demuestra que PyRECE es una posibilidad concreta de desarrollar con Python un Sistema Integrado de Aplicaciones (SIAP) de software libre, en este caso como alternativa al aplicativo RECE de factura electrónica, pero hay otros casos donde se podría hacer lo mismo si los servicios web estuvieran disponibles (por ej., IVA para DD.JJ., SICOSS para seguridad social., etc.). Cabe aclarar que el desarrollo es independiente de AFIP (sin compartir siquiera una sola línea de código), y para llevar a cabo este proyecto no ha sido necesario solicitar fondos públicos, reuniones, entrevistas o conformar grupos de trabajos mixtos o modificar en forma alguna las normas y procedimientos del Estado.

Por último, se ha desarrollado una herramienta superadora que unifica los distintos webservices (factura electrónica nacional, exportación, bienes de capital, etc.) e integra todo lo expuesto anteriormente, utilizando una base de datos para almacenar e intercambiar la información, generando las facturas en PDF, pudiéndolas cargar desde y hacia archivos de texto, envío por email o FTP, entre otras cuestiones.

En general utiliza PostgreSQL, pero también se puede usar otras bases de datos (PyODBC o SQLite).

5. El Modelo de Negocios

Un tema resuelto fue el modelo de negocios, sobre todo conociendo que hay otras alternativas cerradas y se decidió mantener el camino del software libre, por lo que encontrar la combinación justa para poder competir no fue un tema menor.

El código fuente esta publicado en el repositorio de GoogleCode [1] bajo la licencia GPLv3, con los respectivos scripts e instructivos de instalación.

Para los personas que desean evaluar la interfaz pero no conocen Python o no están familiarizadas con el software libre, se pone a disposición un instalador para demostración totalmente funcional en homologación (testing). Se ofrece un soporte comercial pago para los que deseen tener acceso al instalador para producción o necesiten realizar consultas, soliciten corrección ajustes, actualizaciones prioritarias o particulares. Dentro del marco del soporte comercial, se ofrecen excepciones a la

licencia GPLv3 para los que necesiten garantía limitada o poder incorporar y distribuir la interfaz con software cerrado.

El resultado se entiende que ha sido bastante satisfactorio, posibilitando extender y mantener el proyecto financieramente, contribuyendo al software libre y a la comunidad con herramientas alternativas y superadoras, empezando a crear un grupo de desarrolladores interesados en el tema.

Los instaladores de la interfaz tienen miles de descargas desde el sitio del proyecto (sin contabilizar las descargas directas del código fuente y sitios de distribución alternativos), cuenta con un grupo de usuarios de más de cien miembros [14] y muchas empresas y compañías importantes han contratado el soporte, pudiendo citar como referencias comerciales a The Coca-Cola Company Latin America; La Hispano Argentina Curtiembre SA; Boehringer Ingelheim S.A.; Tupperware Brands Argentina; Grupo Solunet S.A. (ISP); Diario El Litoral SRL; Laboratorio Scienza Argentina; E-Payment SRL (DineroMail); entre más de 200 clientes en distintos rubros (Empresas y desarrolladores de Sistemas de Gestión y ERP Corporativos; Distribuidores, Consumo Masivo y Retail; Productos Farmacéuticos y Hospitalarios; Importadores y Exportadores; Empresas de Turismo; Editoriales, Librerías y Diarios; Proveedores de Internet y Televisión por Cable; Profesionales Independientes; Servicios de facturación e impresión masivos).

Esto no ha sido totalmente sin contratiempos ni esfuerzos:

- Los instaladores y paquetes fueron fundamentales para que las personas puedan evaluar los productos, sobre todo en tecnologías no tan difundidas como Python, y principalmente para Windows, que ha sido el mayor mercado para esta interfaz.
- La documentación y ejemplos fueron otro punto importante, y por experiencia es un tema donde se debe profundizar constantemente, aún en los casos que parezca que es suficiente (incluimos preguntas frecuentes, recortes de código, aclaraciones importantes, etc.).
- Los cursos de capacitación y talleres son muy productivos, permiten extenderse un poco más sobre el tema y conocer a los interesados.
- Para la difusión ha ayudado mucho blogs, noticias, eventos de software libre (aunque hay que notar que al principio han sido reacios a considerar el proyecto, por ej. rechazándolo en la primer conferencia de python por no considerar a sus desarrolladores y usuarios lo suficientemente relevantes para el evento). A medida que el proyecto fue apareciendo en los buscadores fue creciendo su popularidad y utilidad real para los usuarios. Incluso ha sido necesario contratar anuncios publicitarios (Google AdWords) para sostener la difusión del proyecto.
- El soporte comunitario en nuestro caso no ha sido efectivo, la lista de correo y los sistemas de tickets/issues no fueron muy utilizados, tampoco ha habido muchas contribuciones y revisiones al código fuente, quizás por el carácter sensible y/o particular del tema (amén que la filosofía del software libre todavía no ha sido muy bien transmitida en algunos ambientes). Como

muestra se ofrecen los comentarios en la página de Instalación [5], donde se tuvo que subir un video ya que algunos usuarios tenían dificultades con la instalación de Python y descargar el código desde el repositorio, utilizando los comentarios para solicitar ayuda (en vez de los issues). Es notable remarcar que, las personas que más se interesaron y colaboran, por ej. respondiendo consultas o enviando ejemplos, en principio no eran participantes de ninguna comunidad de software libre. De hecho, un proyecto similar y anterior nacido desde la comunidad, en el que se intentó contribuir,, no tuvo éxito y fue abandonado (originalmente desde el grupo de usuarios de Ubuntu Argentina, con la lista de correo linux-afip@googlegroups.com y sitio web <https://wiki.ubuntu.com/ArgentinaTeam/AFIP> que actualmente fueron dados de baja).

6. Conclusión

Del trabajo efectuado se desprende que es factible una solución para lograr interoperabilidad, no solo entre distintos lenguajes y plataformas, sino también entre distintas filosofías de licenciamiento de software. Se pone de manifiesto a su vez, que es posible un desarrollo colaborativo de herramientas y aplicativos para consumir servicios de organismos gubernamentales (de manera independiente y sin necesidad de solicitar recursos públicos o cambiar la normativa vigente), posibilitando modelos de negocios para aportar conocimientos mediante la creación de software libre local con proyección internacional.

7. Agradecimientos

Se agradece especialmente a las comunidades de software libre, especialmente a los desarrolladores y usuarios de Python, web2py, wxPython, PostgreSQL, GNU/Linux y Debian/Ubuntu, ya que sin el código fuente, documentación y demás contribuciones, este trabajo no hubiera sido posible.

Se agradece en particular a los siguientes desarrolladores que han colaborado con el proyecto PyAfipWS [1]: alanizmarcelo, hexa662, soriasoft, margamanterola, matigro, deluca.vicente, marcelorobin, gerardoallende, gcorrad, RedianOfPiet, kevin.bullmann42, arielbvargas, correoariel. También se agradece a los siguientes colaboradores nacionales e internacionales que han contribuido a la biblioteca de PySimpleSOAP [7] y PyFPDF [8]: marcelo.fidel.fernandez, darell.tan, r1chardj0n3s, matee, jinyinqiao, jtatum, pckujawa, a.saglimbeni, dragonfyre13@gmail.com, ralf.henschkowski, allan.crooks, pjimenez3, pedrocerezo, jpc, luka.birsa, jredrejo, simonm3.

Se agradece a massimo.dipierro, creador del framework web2py, por haber agregado estas bibliotecas a dicha herramienta, logrando una difusión aún mayor.

Referencias

1. Proyecto PyAfipWS. <http://pyafipws.googlecode.com/>
2. Minisitio AFIP factura electrónica: <http://www.afip.gov.ar/fe>

3. Documentación Técnica de AFIP: <http://www.afip.gov.ar/ws>
4. <http://code.google.com/p/pysimplesoap/issues/detail?id=49>
5. <http://code.google.com/p/pyafipws/wiki/InstalacionCodigoFuente>
6. Doug Hellman, Evaluating Tools for Developing with SOAP in Python. Originally published in Python Magazine Volume 3 Issue 9 , September, 2009. ISSN. 1913-6714
<http://www.doughellmann.com/articles/pythonmagazine/features/building-soap-service/index.html>
7. Proyecto PySimpleSoap. <http://pysimplesoap.googlecode.com/>
8. Proyecto PyFPDF. <http://pyfpdf.googlecode.com/>
9. Mark Hammond & Andy Robinson. Python Programming on Win32. 1st Edition January 2000 1-56592-621-8, O'Reilly Order Number: 6218
10. SOAP Version 1.2 specification. W3C. Second Edition. 27 de Abril de 2007.
<http://www.w3.org/TR/soap/>
11. PHP SOAP extension. <http://php.net/manual/es/book.soap.php>
12. PHP SimpleXML extension. <http://php.net/manual/es/book.simplexml.php>
13. Aplicativos AFIP. <http://www.afip.gob.ar/aplicativos/>
14. <https://groups.google.com/forum/?fromgroups#!forum/pyafipws>
15. Mariano Reingart, Bruno Cezar Rocha, Jonathan Lundell, Pablo Martin Mulone, Michele Comitini, Richard Gordon, Massimo Di Pierro. web2py application development cookbook. Packt Publishing, March 2012. ISBN: 1849515468
16. <http://www.fpdf.org/en/links.php>
17. <http://www.reportlab.com/software/opensource/>