pyxser, Serialización XML en Python

Daniel Molina Wegener
dmw@coder.cl

16 de octubre de 2009

Resumen

Pyxser es un proyecto *FOSS* desarrollado por Daniel Molina Wegener como una investigación e iniciativa propias. Es una *extensión Python*[1] que utiliza las rutinas de *libxml2* para realizar tareas de serialización y deserialización de objetos Python. **Pyxser** utiliza un *modelo* de serialización definido en un *esquema XML*[2] y también en una *DTD*[3]. La definición de un modelo de serialización permite la validación de la estructura del objeto serializado. El modelo también permite empotrar objetos Python que estén serializados en otros documentos *XML*, como por ejemplo, incluir la definición de objetos Python dentro de un *WSDL*[4] para utilizarlo con *Web Services*.

- ***Palabras Clave*** Python, XML, serialización, deserialización, modelo, implementación, C, Python C/API, XML Schema, DTD, normalización.

Introducción

Comúnmente el *modelo de serialización* que siguen las distintas plataformas presenta un nivel de *personalización* demasiado alto. El proceso común de serialización, en lugar de buscar una *definición*, crea instancias XML sin una estructura previa.

El proceso de serialización en **.NET**, por ejemplo, crea *elementos XML* por cada clase, a veces omitiendo el *espacio de nombres* de las clases. Esto conlleva a la necesidad de generar un *esquema XML* por cada clase serializada o uno que los agrupe a todos, generando un alto nivel de acoplamiento[7]. De la misma forma, los objetos serializados son difíciles de *transportar* entre las diferentes plataformas.

Pyxser en cambio, presenta un único modelo de serialización, de manera constante y con un nivel apropiado de cohesión[7], sin un acoplamiento respecto de su implementación y que permite realizar transferencias de objetos desde un lugar a otro[2], sólo con adoptar el *esquema XML* y el conocimiento de las *clases* que se van a serializar.

[1] Un módulo escrito en **C** utilizando la *Python C/API*
[2] XML Schema
[3] Document Type Definition
[4] Web Services Description Language

Motivación

> *Definir un modelo de serialización normalizado para objetos, que permita serializar objetos de cualquier plataforma, bajo un esquema XML bien definido y entregar un nivel apropiado de interoperabilidad entre distintas plataformas.*[6]

Pyxser es la implementación de este modelo. Se escogió *Python* como lenguaje, dada la potente capacidad de *introspección*[9] que posee su interfaz de extensión en **C** o más conocida como *Python C/API*. El modelo claramente es migrable a otros lenguajes que soporten similares características de *introspección*.

Modelo de Serialización

El modelo de serialización es *simple*[5]. Reune las características suficientes para realizar serializaciones de objetos de varios lenguajes. Esta misma *simplicidad*, desde el punto de vista del *diseño de la estructura*, le permite ser portado y empotrado en otros documentos XML[4].

El elemento *pyxs:obj*[6] se utiliza para serializar objetos. Cada objeto presenta los atributos necesarios para su representación con el adecuado soporte para *referencias cruzadas* y *referencias circulares*. Este elemento posee 2 atributos esenciales, *module* y *type*, donde *module* se utiliza para declarar el espacio de nombres y *type* para declarar la clase. Como identificador se utiliza el atributo *objid*, el cual esta declarado como *ID* en la *DTD* y *xs:ID*[7] en el esquema XML.

```
<xs:element name="obj">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="prop"/>
      <xs:element ref="col"/>
      <xs:element ref="obj"/>
    </xs:choice>
    <xs:attribute name="version" default="1.0">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="1.0"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="objid" type="xs:ID"/>
    <xs:attribute name="objref" type="xs:IDREF"/>
    <xs:attribute name="type" type="xs:NMTOKEN"/>
    <xs:attribute name="name" type="xs:NMTOKEN"/>
    <xs:attribute name="module" type="xs:NMTOKEN"/>
    <xs:attribute name="size" type="xs:NMTOKEN"/>
  </xs:complexType>
</xs:element>
```

Listing 1: Definición del Objeto

[5] K.I.S.S.

[6] Con el namespace *pyxs* [3] referente a `http://projects.coder.cl/pyxser/model/`

[7] *xs:* es el espacio de nombres para los esquemas XML, `http://www.w3.org/2001/XMLSchema`

Cuando un objeto ya se encuentra serializado y se encuentra la misma referencia en el árbol de objetos o el caso de una referencia circular, se vuelve a crear un instancia del elemento *pyxs:obj*[8], con la diferencia de que no agrupa subelementos, y presenta el atributo *objref* para referenciar al objeto ya serializado. Este atributo esta declarado como *IDREF* en la *DTD* y como *xs:IDREF*[1] en el esquema XML. Lo mismo ocurre con las referencias cruzadas, si un objeto ya se encuentra serializado, se crea una referencia al objeto de la misma forma. Se mantiene *type* y *module* por una cuestión de integridad, pero no se hace uso de estos de manera concreta.

Para los tipos primitivos o básicos, como enteros y cadenas, se utilizan elementos *pyxs:prop*. El elemento *pyxs:prop*, presenta el atributo *type* para declarar el tipo y el atributo *size* para el tamaño. También presenta el atributo *name* para declarar el nombre de la propiedad.

```
<xs:element name="prop">
  <xs:complexType mixed="true">
    <xs:attribute name="type" use="required" type="xs:NMTOKEN"/>
    <xs:attribute name="name" type="xs:NMTOKEN"/>
    <xs:attribute name="size" type="xs:NMTOKEN"/>
    <xs:attribute name="compress">
      <xs:simpleType>
        <xs:restriction base="xs:token">
          <xs:enumeration value="yes"/>
          <xs:enumeration value="no"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="encoding">
  <xs:simpleType>
    <xs:restriction base="xs:token">
      <xs:enumeration value="ascii"/>
      <xs:enumeration value="base64"/>
    </xs:restriction>
  </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>
```

Listing 2: Definición de las Propiedades

Para declarar las colecciones — y debido a que no siempre las colecciones son clases propiamente tal — se creó el elemento *pyxs:col*, el cual también presenta los atributos *type* y *name*. El tamaño de la colección se da por la cantidad de subelementos que esta posea, en lugar de utilizar un atributo *size* o *length*.

```
<xs:element name="col">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="prop"/>
      <xs:element ref="col"/>
      <xs:element ref="obj"/>
    </xs:choice>
    <xs:attribute name="type" use="required" type="xs:NMTOKEN"/>
    <xs:attribute name="name" use="required" type="xs:NMTOKEN"/>
  </xs:complexType>
</xs:element>
```

Listing 3: Definición de las Colecciones

El modelo es *recursivo*, ya que puede reunir objetos bajo el elemento *pyxs:obj* o *pyxs:col*.

El modelo, globalmente, presenta una serie de ventajas respecto de otros modelos de serialización. Este es reutilizable, importable y reimplementable en otros lenguajes que soporten DOM, permitiendo crear objetos en runtime utilizando métodos DOM estándar, sin caer en elementos propietarios de este tipo de API. También se pueden generar las clases necesarias para trabajar con el modelo bajo APIs externas a Python, como por ejemplo JAXB en Java.

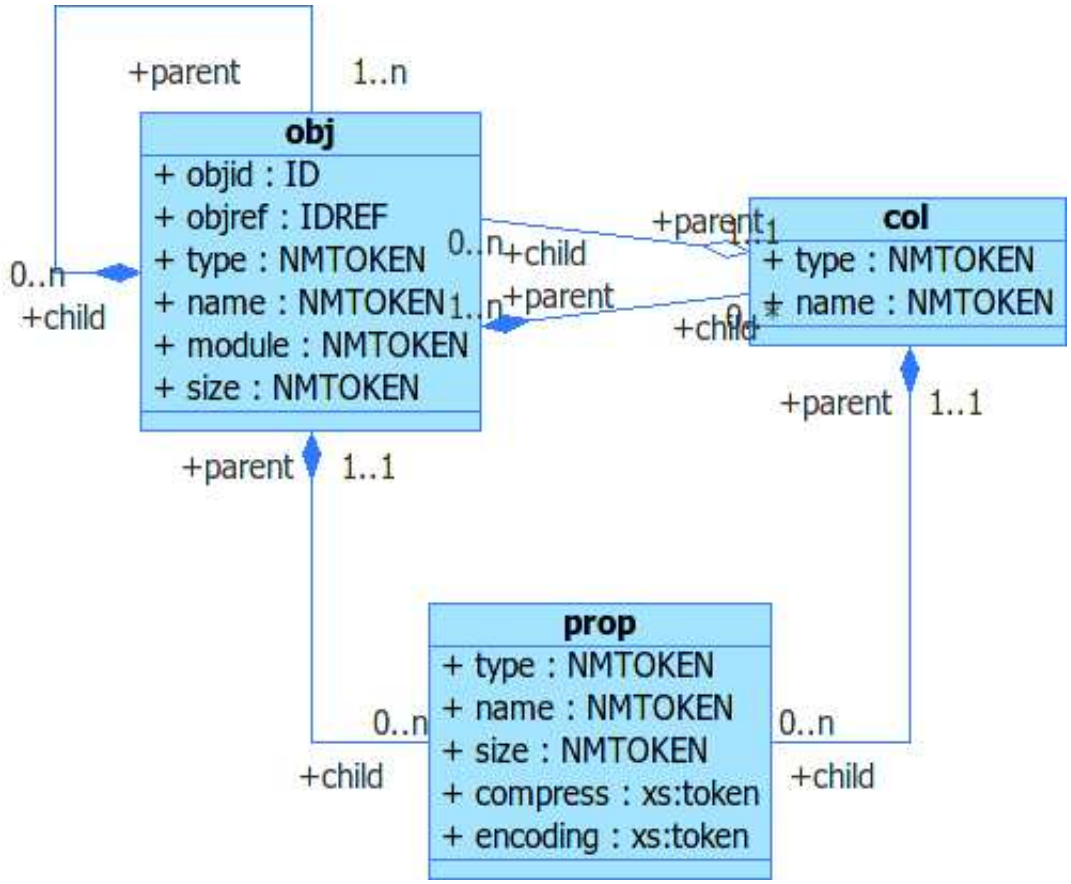

Figura 1: Modelo de Serialización

Reutilización del Modelo

El modelo al estar representado bajo la forma de *DTD*[8] y esquema XML[9] es reutilizable en otras plataformas. Por ejemplo bajo *Java* se pueden generar los *XML Beans* necesarios para interactuar con este o bien replicar la serialización bajo las características de introspección que posee *Java*. Sin embargo, *Java*, a diferencia de *Python* carece de constructores crudos por medio de introspección y requiere que sea declarado un constructor de firma sin parámetros.

[8] Document Type Definition
[9] XML Schema

Otro caso particular, podría ser *PHP*, el cual carece de espacios de nombres y una *estructura ordenada* de carga de clases — se puede cargar una clase desde cualquier parte del sistema.

Sin embargo, *Python* podría recibir objetos serializados bajo este modelo y reconvertirlos en un objeto concreto respecto de su clase original sin problemas respecto de su uso. *Java* posibilita la creación de objetos *mock* — imitaciones de clases reales — utilizando introspección, pero es sabido que *Java* posee *bajo rendimiento* en su implementación de introspección pura desde *Java*. Es posible que una implementación con *JNI*[10] sea más rápida[11]. Aun así *JNI* carece de constructores crudos[5].

El modelo de serialización, al estar representado por un único esquema XML, sin instancias específicas para cada objeto que sea serializado, permite reutilizar el modelo en otros documentos XML. De esta forma, es posible incluir el documento dentro de otros esquemas, por ejemplo:

```
<wsdl:types>
    <xsd:schema targetNamespace="http://www.example.org/Hello/"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns:tns="http://www.example.org/Hello/"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:pyxs="http://projects.coder.cl/pyxser/model/">
        <xsd:import
            namespace="http://projects.coder.cl/pyxser/model/"
            schemaLocation="pyxser-1.0.xsd" />
        <xsd:element name="obj" />
    </xsd:schema>
</wsdl:types>
```

Listing 4: Ejemplo dentro de WSDL

Dentro de una especificación *WSDL*, permite hacer uso de los objetos serializados como si fuesen parte del servicio y tipifcar directamente los objetos a serializar, evitando el uso de clases específicas [6].

Sin embargo, esto no permite crear por si sola una instancia XML que efectivamente represente el objeto. Es necesario conocer la clase en sí, por lo que se debe generar el algoritmo apropiado en otros lenguajes para comunicar el mensaje con la reprsentación exacta que se desea.

Otro detalle importante, más referente a la implementación, es que se debe serializar un objeto dandole prioridad a los objetos. Esto implica que se deben serializar los objetos primero, para evitar la perdida de datos a momento de encontrar referencias cruzadas o circulares. Esto permite que el algoritmo sea $O(n)$ en lugar de ser $O(n^2)$, dado el doble recorrido que se debe realizar para verificar las referencias a objetos. También, en efecto, se reduce la serialización de $O(n)$ a $O(\log n)$ a medida que incrementan las referencias.

Referencias

[1] Paul V. Biron and Ashok Malhotra. Xml schema part 2: Datatypes second edition. World Wide Web Consortium, Recommendation REC-xmlschema-2-20041028, October 2004.

[2] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible Markup Language (XML) 1.0 (Third Edition), W3C Recommendation 04 February 2004. *The World Wide Web Consortium*, 2004.

[10] Java Native Interface

[11] Sun recomienda no usar introspección via java.lang.reflect en entornos de producción

[3] Tim Bray, Dave Hollander, Andrew Layman, and Richard Tobin. Namespaces in xml 1.0 (second edition). World Wide Web Consortium, Recommendation REC-xml-names-20060816, August 2006.

[4] David C. Fallside and Priscilla Walmsley. XML Schema Part 0: Primer Second Edition, W3C Recommendation 28 October 2004. *The World Wide Web Consortium*, 2006.

[5] Sheng Liang. *Java Native Interface: Programmer's Guide and Reference*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA., 1999.

[6] Ian Piumarta and Alessandro Warth. Open, reusable object models. Technical Report VPRI Research Note RN 2006-003-a, 2006.

[7] Diomidis Spinellis and Georgios Gousios. *Beautiful Architecture: Leading Thinkers Reveal the Hidden Beauty in Software Design*. O'Reilly Media, Inc., January 2009.

[8] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. Xml schema part 1: Structures second edition. World Wide Web Consortium, Recommendation REC-xmlschema-1-20041028, October 2004.

[9] Guido van Rossum. *Python/C API Reference Manual*. Python Software Foundation, 2.5 edition, Septiembre 2006.