XML PARSER

Edinson Sanchez
Kevin Filella
Adrian Aguilar

19 de febrero de 2014

Índice

1. Introducción

Un parser, aplicado a lenguajes de programación computacionales, tiene como objetivo el de manipular ciertas cadenas de caracteres con el fin de asignar o guardar la información presentada en una estructura de datos. Una vez conocido esto, el objetivo de este proyecto es claro y conciso; el de crear un parser XML en Pythonl que pueda manipular, leer y guardar los datos presentados en un archivo XML específico, con una estructura fija y con un propósito previsto; el de realizar consultas a la estructura.

1.1. Objetivo

Nuestro objetivo en este proyecto es el de crear un parser XML en Pythonl que pueda manipular, leer y guardar los datos presentados en un archivo XML específico, con una estructura fija y con un propósito previsto; el de realizar consultas a la estructura.

2. Desarrollo

2.1. Desarrollo inicial

Inicialmente, debido a la naturaleza del proyecto y a las características del curso presente, tuvimos que aprender a instalar y manejar las herramientas de desarrollo adecuadas. Siendo este lenguaje funcional y de uso masivo fue muy fácil instalar las herramientas adecuadas y empezar a programar en Python.

3. Problemas

Problemas en el aprendizaje e implementación del lenguaje Python a la creación de un parser XML.

3.1. Primer Problema - Sintaxis

Al momento de iniciar el curso de lenguajes de programación, nosotros como estudiantes, debido al programa de estudios, hemos sido iniciados en algunos lenguajes de programación. Lenguajes como Pascal, BASIC, C, C++, Java, Android etc. Estos lenguajes son todos de alguna manera u otra muy similares entre sí. Python es un lenguaje funcional con características muy similares a los lenguajes anteriormente mencionados. Lo que hizo bastante fácil el trabajo de aprender el correcto uso de la sintaxis en Python. El único aspecto un poco confuso es la estricta rigurosidad del lenguaje en cuanto a la indentación de las líneas.

3.2. Segundo Problema - Estructura

Como este proyecto ya había sido realizado anteriormente en lenguaje Haskell, este proyecto en principio parecía simplemente un trabajo de traducción. Por lo cual de-

cidimos implementar una estructura similar a la del proyecto anterior. Esta labor en principio sencilla en etapas posteriores del desarrollo demostraría ser una mala elección de diseño, ya que en Haskell todo lo resolvíamos recursivamente mientras que Python no maneja tan bien la recursividad.

3.3. Tercer Problema - Recursividad

El hecho de que Python maneje la recursividad de una manera muy distinta a la de Haskell fue el problema más grande en el desarrollo de este proyecto. Python guarda la salida de cada función recursiva en una pila de tamaño limitado. Puede guardar hasta 14000 líneas de recursión, Esto es un problema irresoluble cuando se quiere parsear documentos de gran tamaño.

3.4. Cuarto Problema - Busquedas

El parseo del documento se lo tuvo que realizar iterativamente, línea por línea, lo que demora algunos segundos lo cual es demasiado tiempo. Pero para el alcance y las limitaciones de este proyecto podría ser incluso aceptable. El problema mayos se daría cuando realizamos las búsquedas en la lista de devices que grabamos. Debido a que no pudimos aplicar recursividad en el parseo inicial los devices quedaron guardados en una larga lista sin orden aparente.

La búsqueda en una estructura tan grande consume demasiado tiempo, en el orden de las decimas de minutos, por lo cual intentamos forzar la recursividad utilizando una librería que aumenta l límite de líneas de recursividad que permite nativamente Python. Pero debido a la poca experiencia con este lenguaje y más aun con esta librería no estándar no pudimos hacer un uso correcto de la misma ni implementar búsqueda con recursión.

4. Alcance del Proyecto

Al final nuestro proyecto tiene un alcance incompleto, más que nada debido a la falta de tiempo, experiencia y al hecho de que perdimos muchos días tratando de traducir el proyecto que habíamos realizado anteriormente en Haskell a Python. Nuestro proyecto puede realizar búsquedas lineales como devices que tengan un determinado fallback o determinadas características(capabilities). No puede realizar búsquedas recursivas como producir toda la línea de devices hasta llegar a generic.

5. Conclusiones

Como conclusión podemos decir que Haskell es un lenguaje mucho más efectivo para hacer parseo de grandes estructuras que Python. La forma iterativa de realizar las operaciones y el límite del tamaño que tiene la pila de recursión hacen muy difícil la tarea de realizar búsquedas grandes. Python tiene una sintaxis muy fácil de aprender

y utilizar, además que es un lenguaje muy flexible en el que puedes importar librerías de otros lenguajes, lo que lo hace ideal para programar cuando se requiere capacidad multiplataforma.