Documentación del 2do Proyecto del 2do Parcial: Parser para XML

Efrén Romero

1. Descripción

El siguiente proyecto trata sobre la implementación de un analizador sintáctico/-procesador para documentos XML en el lenguaje de programación Python, el cual tiene como objetivo la aplicación de los conocimientos adquiridos del paradigma orientado a objetos, la aplicación de los conceptos fundamentales de los lenguajes de programación y el aprendizaje de nuevas herramientas de desarrollo, lo cual nos permitirá tener más opciones al momento de escoger una herramienta para resolver un problema determinado y mejorar nuestra habilidad como programadores.

Para el desarrollo de esta aplicación se usó el editor de texto Sublime Text 3, la versión 3.3.4 del intérprete de Python y el documento wurfl-2.3 para realizar las pruebas de funcionamiento.

Nota: El documento wurlf-2.3 no es de mi autoría y fue obtenido en: `http://sourceforge.net/projects/wurfl/files/WURFL/2.3/`

2. Alcance del Proyecto

1. Lo que logré realizar:

   - Cargar documentos XML bien estructurados, que no contengan la línea de declaración y donde cada línea es:

     a) Sólo una etiqueta de cierre.

     b) Sólo una etiqueta de apertura.

     c) Sólo una etiqueta de cerrado automático.

     d) Sólo un fragmento(o todo) del contenido de una etiqueta.

     Ejemplo:

Código 2.1: Documento que la aplicación si logra cargar

```
1  <tag1>
2      <tag11 k1="a" k2="b">
3          <sct t1="x" t2="y"/>
4          tag11Content
5      </tag11>
6      <tag12 k1="c" k2="d">
7          tag12Content
8      </tag12>
9      <tag12 k1="e" k2="f">
10         tag13Content
11     </tag12>
12     Tag1Content
13 <tag1>
```

- Cada nodo XML es en sí un objeto de la clase xml, cuyo método constructor se muestra en el cuadro:

Código 2.2: Método constructor de la clase xml

```
1     def __init__(self, tagName, parent =
          'noParent', attributes = 'noAttributes',
          children = 'noChildren', content =
          'noContent'):
2         self.__parent = parent #Objeto de la
              clase string.
3         self.__tagName = tagName #Objeto de la
              clase string.
4         self.__attributes = attributes #Objeto
              de la clase dict.
5         self.__children = children #Lista de
              objetos de la clase XML.
6         self.__content = content #Objeto de la
              clase string.
```

- Cargar un fragmento del documento wurfl-2.3. Específicamente la aplicación logra cargar en memoria el árbol mostrado en la Figura 2.1.
- Realizar diferentes tipos de búsquedas en la estructura creada. Hasta ahora he implementado 3 funciones para realizar búsquedas:

  a) getDBID, la cual que recibe como parámetros el nombre del dispositivo y una referencia al nodo raíz y retorna una referencia al dispositivo que posee el nombre pasado como parámetro, caso contrario retorna None.

  b) getCapabilityValue, la cual me permite obtener el valor de una capacidad determinada si se conocen el nombre de la capacidad y el nombre del dispositivo (o una referencia al dispositivo)

  c) getDevicesWithCapVal, que permite recuperar todos los dispositivos que poseen en común un valor determinado para una capacidad determinada, si se conoce el nombre de la capacidad, una referencia al nodo raíz, y opcionalmente el valor de la capacidad.

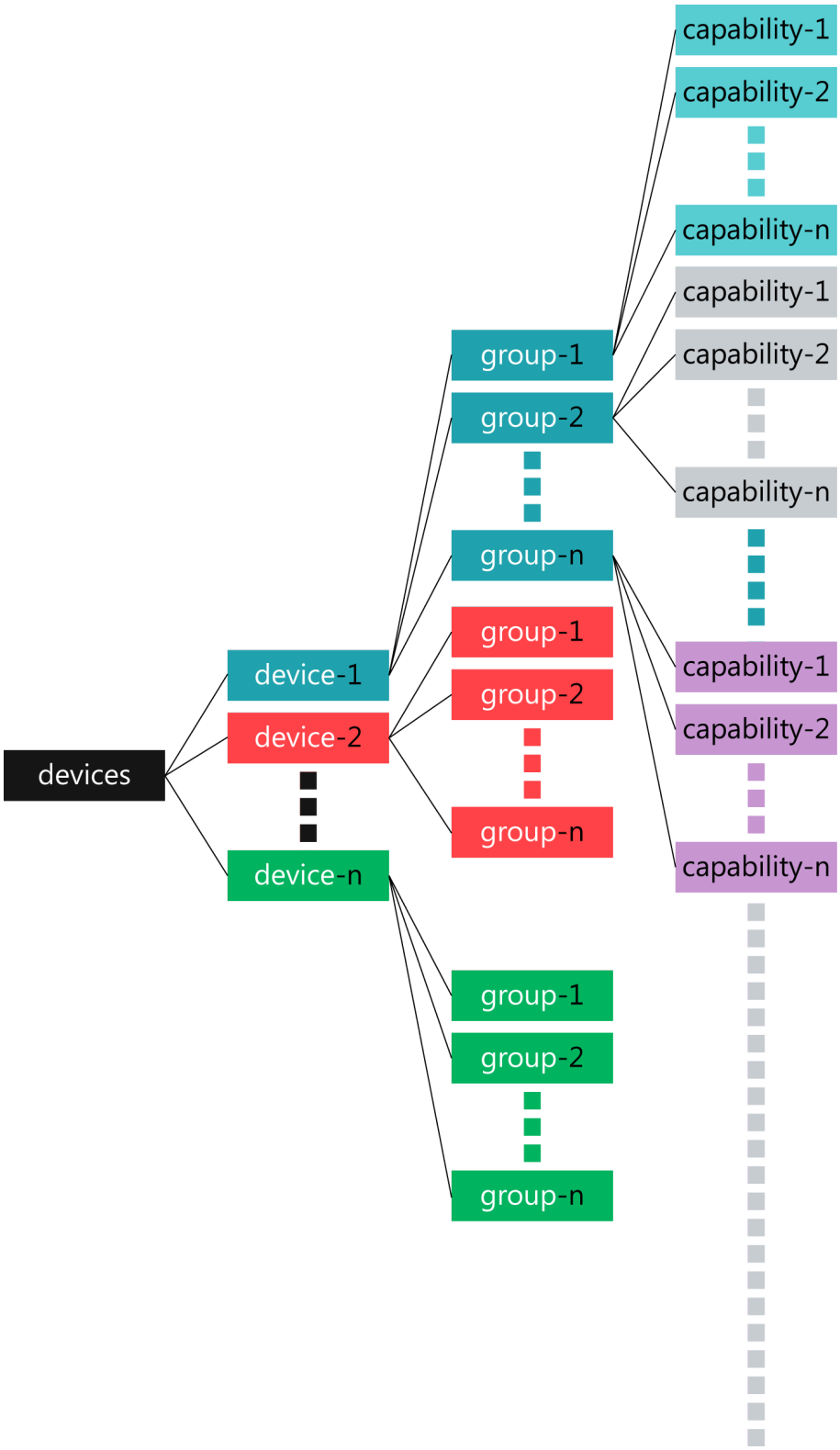

Figura 2.1: Árbol XML

2. Lo que no logré realizar:

   - Determinar si un documento XML está bien estructurado.
   - Cargar documentos XML en donde exista una línea (o más) que posea una etiqueta de apertura y cierre en si misma. Ejemplo:

Código 2.3: Documento que la aplicación no logra cargar

```
<tag1>
    <tag11 k1="a" k2="b">
        <sct t1="x" t2="y"/>
        tag11Content
    </tag11>
    <tag12 k1="c" k2="d">tag12Content</tag12>
    <tag12 k1="e" k2="f">tag13Content</tag12>
    Tag1Content
<tag1>
```

3. Experiencia usando Python

- Me agrada mucho el hecho de que python es un lenguaje multiparadigma, ya que aunque es posible programar usando el paradigma estructurado, orientado a objetos o el paradigma funcional.
- Considero que la sintaxis es bastante simple y clara, lo cual facilita la lectura, escritura de código y el aprendizaje.
- Python es un lenguaje interpretado y el intérprete del lenguaje está disponible para varias plataformas, lo cual es un gran punto a favor para la portabilidad.
- Debido a que en este lenguaje no se realizan declaraciones explícitas de tipos, prácticamente no es necesario realizar sobrecarga de métodos, ya que los tipos de variables y funciones estarán determinados por las asignaciones realizadas. Esta cualidad, aunque es un punto a favor de la expresividad/facilidad de escritura, puede resultar un poco confusa (detrimento para la legibilidad) si se trata de funciones con muchos bloques anidados y además pueden resultar problemas de seguridad.
- Considero que la simplicidad y versatilidad que poseen las estructuras básicas como: listas, tuplas y diccionarios realmente facilita la tarea de representar objetos complejos.
- Estoy acostumbrado a usar la indentación aunque no sea un requerimiento del lenguaje, ya que pienso que mantiene el código organizado y facilita ampliamente la lectura y comprensión del mismo, por esta razón considero que el hecho de que la indentación sea obligatoria en este lenguaje, es un punto a favor del mismo.

Conclusión: Pienso que el estudio de nuevos lenguajes amplía en gran cantidad la capacidad para expresar nuestras ideas, por esta razón y por todo el potencial que veo en este lenguaje, considero que profundizar en el estudio de Python es una excelente decisión.