Софийски университет „Св. Кл. Охридски”

Факултет по математика и информатика

Бакалавърска програма
„Софтуерно инженерство”

Предмет: XML технологии за семантичен Уеб

Зимен семестър, 2017/2018 год.

Тема 37: Преглед на Graph Markup Language (GraphML)

Есе

Автори:

Бойко Цветанов, фак. номер 61953

Силвия Янакиева, фак. номер 61881

ноември, 2017
София

Съдържание

1 Въведение

Голяма част от приложенията, които използваме в ежедневието си, използват графи за структурирането, съхраняването и обработката на информация.

Използвайки социални мрежи, всеки потребител бива “възел” на графа. Свързвайки се с друг, те заедно създават “ребро”. Използвайки GPS, Google Maps и други приложения за навигация, ние търсим пътя към избраната от нас дестинация, използвайки свойството на графите за намиране на най-кратък път. Страниците в Интернет представляват възли на граф, свързани чрез ребра - хиперлинкове .

Различните системи, използващи данни под формата на графи, имат нужда да съхраняват и обменят тази информация. Обмяната на тази информация е трудоемък процес, когато различните системи я представят по различен начин. През годините са правени опити за създаване на унифициран език за комуникация между тези системи.

През 2000 година на 8-мия Симпозиум по Изобразяване на графи (GD 2000) се поставя началото на разработката на модерен формат за обмен на графи между инструменти и приложения, работещи с тях. Този формат бива наречен Graph Markup Language (GraphML). Благодарение на XML базирания си синтаксис, GraphML може да се използва в комбинация с други XML-базирани формати.

2 Характеристики и използване на езика

2.1 Дефиниции

Графът представлява абстрактна структура, имаща за цел да представи данните в свързан вид. В математиката графът G се представя чрез наредената тройка:

$$G = (V,E,D),$$

където V е множество от възли, E е множество от ребра, а D е функция, съпоставяща на всяко ребро двойка върхове.

Възлите са елементите, в които се съдържа основната информация на графа. Те могат да съдържат както данни, така и други графи.

Ребрата представляват връзката между възли. Ребрата биват насочени и ненасочени. Всяко ребро може да притежава тегло, чрез което се сравнява с другите ребра.

2.2 Основни характеристики

Най-голямото удобство на GraphML е близосста му до стандартния XML синтаксис.

Подобно на останалите XML-базирани езици, е дефиниран чрез XML Schema. Съществува и DTD документ, предназначен за парсъри, които не работят със Schema, но единствената регламентирана спецификация е Schema, намираща се на адрес: http://graphml.graphdrawing.org/xmlns/1.1/graphml.xsd

```
<xs:schema xmlns="http://graphml.graphdrawing.org/xmlns"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://graphml.graphdrawing.org/xmlns" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:annotation>
    <xs:documentation source="http://graphml.graphdrawing.org/" xml:lang="en">
      This document defines the GraphML language including GraphML attributes and GraphML
      parseinfo.
    </xs:documentation>
  </xs:annotation>
  <xs:redefine schemaLocation="http://graphml.graphdrawing.org/xmlns/1.1/graphml-
  structure.xsd">
    <!-- redefinition as in graphml-attributes.xsd  -->
    <xs:attributeGroup name="key.extra.attrib">
      <xs:attributeGroup ref="key.extra.attrib"/>
      <xs:attributeGroup ref="key.attributes.attrib"/>
    </xs:attributeGroup>
    <!-- redefinition as in graphml-parseinfo.xsd  -->
    <xs:attributeGroup name="graph.extra.attrib">
      <xs:attributeGroup ref="graph.extra.attrib"/>
      <xs:attributeGroup ref="graph.parseinfo.attrib"/>
    </xs:attributeGroup>
    <xs:attributeGroup name="node.extra.attrib">
      <xs:attributeGroup ref="node.extra.attrib"/>
      <xs:attributeGroup ref="node.parseinfo.attrib"/>
    </xs:attributeGroup>
  </xs:redefine>
  <!-- types as in graphml-attributes.xsd  -->
  <xs:simpleType name="key.name.type" final="#all">
    <xs:annotation>
      <xs:documentation source="http://graphml.graphdrawing.org/(Dokumentation der Attributes
      Erweiterung; entsprechende Stelle.html)" xml:lang="en">
        Simple type for the attr.name attribute of <key>. key.name.type is final, that is, it
        may not be extended or restricted. key.name.type is a restriction of xs:NMTOKEN
        Allowed values: (no restriction)
      </xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:NMTOKEN"/>
  </xs:simpleType>
```

Фигура 1. Част от XML Schema дефиницията на GraphML (източник: [2]).

GraphML има голямо предимство - разширяемост. С малки добавки към Schema дефиницията, към GraphML документа могат да се добавят XML тагове и атрибути. Благодарение на тях в графа освен данните от стандартните атрибути може да бъде съхранявана още разнообразна по вид и формат информация.

GraphML е разработен с идеята да бъде:
-Опростен - форматът трябва да се анализира и интерпретира еднакво лесно от хора и машини. Не трябва да има неясноти и всеки валиден GraphML документ трябва да има единствена интерпретация.
-Генерализиран - не трябва да има ограничения относно теорията на графите - т.е. йерархични графи, хиперграфи и прочие трябва да могат да бъдат представяни чрез основния синтаксис на езика, без да е нужно да се добавя разширения към синтаксиса.
-Разширяем - форматът трябва да може да се разширява по добре дефиниран начин, за да може чрез документа да се предостави допълнителна информация, изисквана от приложения, или за съставяне на по-сложни екземпляри (напр. изпращане на алгоритъм за оформление заедно с информацията за данните в графа)

-Устойчив - системи, които не могат да обработват всички видове графи или добавената към тях информация трябва да могат лесно да извличат единствено информацията, която им е нужна.

2.3 Въведение в използването на езика

2.3.1 Основен синтаксис

Подобно на стандартен XML документ, за да бъде валиден, GraphML документът започва с декларация или на DTD, или Schema документ, по който да бъде валидиран. Както споменахме в предишната точка, GraphML е дефиниран чрез XML Schema с цел по-строг контрол върху съставянето на екземпляри.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
     http://graphml.graphdrawing.org/xmlns/1.1/graphml.xsd">
    <!--Content: List of graphs and data-->
</graphml>
```

Фигура 2. Минимален валиден GraphML документ (източник: [1]).

Първият ред на документа от Фиг. 2 е познатият ни, задължителен за всички XML документи, който в случая определя стандартното кодиране на символи UTF-8, че документът отговаря на версия 1.0 от стандарта XML . Вторият ред съдържа корена на GraphML документа <graphml>, принадлежащ към пространството от имена, декларирано на този ред. Тъй като основната цел на документа е описание на структурата и данните на граф, това пространство от имена е дефинирано като пространство по подразбиране. Минималното съдържание на валиден GraphML документ е коренът <graphml>.

Валидацията на такъв вид документи не е задължителна. При липса на необходимост, тя може да бъде пропусната. Един минимален GraphML документ без нея би изглеждал по следния начин:

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns" >
        <!--Content: List of graphs and data-->
</graphml>
```

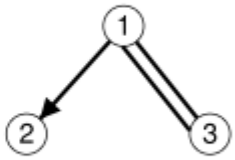
Фигура 3. Минимален валиден GraphML документ без валидация (източник: [1]).

По правилата за съставяне на XML документи, този екземпляр не е валиден.

Едно стандартно описание на граф, използвайки GraphML нотация, изглежда по следния начин:

```
<graphml>
  <graph edgedefault="directed">
    <node id="v1"/>
    <node id="v2"/>
    <node id="v3"/>
    <node id="v4"/>

    <edge source="v1" target="v2"/>
    <edge source="v1" target="v3"/>
    <edge source="v2" target="v4"/>
    <edge source="v2" target="v4" directed="false"/>
  </graph>
</graphml>
```



Фигура 4. Описание на граф чрез GraphML (източник: [1]).

Това описание силно наподобява стандартен XML документ. Това го прави по-лесен за съставяне и разбиране както от хора, така и от програми и приложения.

Основната цел на езика GraphML е представянето на структурата от данни граф в максимално опростен и лесно разбираем формат.

Един GraphML документ започва с декларацията на неговия коренов елемент <graphml>. Това е минималното съдържание, за да бъде документът валиден.

Структурата граф се представя с тага <graph>. В кореновия елемент могат да бъдат декларирани произволно много графи.

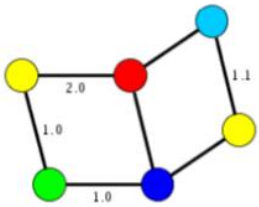
Възлите на графа се представят чрез списък от <node> елементи, като всеки възел трябва да има id атрибут. В спецификацията на GraphML схемата е осигурено, че атрибутът id на възлите е уникален в <graph> елемента.

Ребрата се представят със списък от <edge> елементи. Те сочат към source- и target- възли, идентифицирани посредством стойността на атрибутите source и target на реброто. В спецификацията на езика е осигурено, че стойностите на source и target атрибутите трябва да съответстват на стойности на id атрибут на някои от <node> елементите в същия <graph>. Възможността за налагането на това ограничение още в дефиницията на GraphML езика е една от причините използването на XML схема да е по - добрият вариант от DTD.

Възлите и ребрата могат да се подредят по произволен начин, т.е. редът, в който биват дефинирани няма значение за крайната структура на графа.

Елементът <graph> има атрибут edgedefault, който декларира дали по подразбиране ребрата на графа се приемат за насочени или ненасочени. Индивидуалните ребра могат да пренапишат това като зададат стойността на техния атрибут directed респективно на true или false.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml>
  <key id="d0" for="node"
       attr.name="color" attr.type="string">
    <default>yellow</default>
  </key>
  <key id="d1" for="edge"
       attr.name="weight" attr.type="double"/>
  <graph id="G" edgedefault="undirected">
    <node id="n0">
      <data key="d0">green</data>
    </node>
    <node id="n1"/>
    <node id="n2">
      <data key="d0">blue</data>
    </node>
    <node id="n3">
      <data key="d0">red</data>
    </node>
    <node id="n4"/>
    <node id="n5">
      <data key="d0">turquoise</data>
    </node>
    <edge id="e0" source="n0" target="n2">
      <data key="d1">1.0</data>
    </edge>
    <edge id="e1" source="n0" target="n1">
      <data key="d1">1.0</data>
    </edge>
    <edge id="e2" source="n1" target="n3">
      <data key="d1">2.0</data>
    </edge>
    <edge id="e3" source="n3" target="n2"/>
    <edge id="e4" source="n2" target="n4"/>
    <edge id="e5" source="n3" target="n5"/>
    <edge id="e6" source="n5" target="n4">
      <data key="d1">1.1</data>
    </edge>
  </graph>
</graphml>
```



Фигура 5. Граф с атрибути. Ребрата имат тежести, а възлите - цветове. (За по - добра четимост, пространството от имена и декларацията на схемата са пропуснати.) (източник: [1]).

2.3.2 Атрибути

В повечето случаи, използването само на основната функционалност на GraphML, не ни е достатъчно. Затова е добавено разширение GraphML-Attributes, което е добавено отново в документа graphml.xsd:

`http://graphml.graphdrawing.org/xmlns/1.1/graphml.xsd`

GraphML атрибутите се смятат частично за функции, които приписват стойности на елементите на графа, които често (но не задължително!) имат един и същи тип. Например теглата на ребрата могат да се представят чрез функцията:

$$\text{weight}: E \to R,$$

където weight е теглото, което бива представено като функция между реброто E и множеството на реалните числа.

За да се добавят такива функции към елементите на графа, в GraphML е добавен key/data механизъм. Елемент <key> в началото на документа, декларира нова информационна функция, или по-точно <key> елементът определя уникалния номер, името, домейна и дефиниционната област на стойностите. Стойностите на функцията се дефинират чрез <data> елементи.

Декларацията на всички информационни функции в началото на документа помага на парсерите да определят и конструират подходящите структури от данни в самото начало на процеса на парсване. По същия начин парсърите могат да определят дали някои от задължителните полета информация липсва.

Елементът <key> има XML атрибут for**,** който определя домейна на информационната функция. Атрибутът for може да приема стойности като graph, node, edge, graphml и имена на други типове елементи на графа. Също така for може да приема стойност all, която означава, че всички тези информационни етикети могат да се приложат на всички graph елементи. Атрибутите for и id са задължителни за <key> елементите.

Разширението GraphML-Attributes предлага още два атрибута за <key> елемента - атрибута attr.name, който дефинира името на информационната функция и се използва от парсерите да разпознаят точната информация, която им е нужна, и атрибута attr.type, който определя дефиниционната област на стойностите. Възможни стойности за attr.type са boolean, int, long, float, double и string.

Парсърите, които работят с теглата на ребрата, в общия случай след парсване на <edge> елемента, ще инициализират някаква вътрешна структура от данни, която да съхранява double стойности за всяко ребро. Парсъри, които не познават или не се нуждаят от функция за ребрата с името “weight”, просто ще игнорират <data> елементите свързани с това ребро. Например следното парче код

```
<edge id="e0" source="n0" target="n2">
  <data key="d1">1.0</data>
</edge>
```

дефинира стойност 1.0 като тежест на съответния <edge> елемент. Чрез key атрибута си, <data> елементите сочат към <key> елементи. В GraphML схемата е подсигурено, че стойността на key атрибута трябва да съответства на id атрибута на някой <key> елемент в същия документ.

След като общо взето информационните етикета са единствено частични функции, <data> елементите не трябва да са на лице при всички ребра. Например реброто

```
<edge id="e3" source="n3" target="n2"/>
```

не дефинира стойност на weight функцията. Въпреки това <key> елементите могат да дефинират стойности по подразбиране на съответната информационна функция. Например

```
<key id="d0" for="node" attr.name="color" attr.type="string">
  <default>yellow</default>
</key>
```

декларира функция color върху множеството от <node> елементи и дефинира yellow като цвят по подразбиране на <node> елементите. Възлите могат да пренапишат стойността по подразбиране чрез техния <data> елемент. Например, възелът

```
<node id="n0">
  <data key="d0">green</data>
</node>
```

предефинира цвета си чрез <data> елемента на зелен. Механизмът по подразбиране служи за пестене на място в документа ако много елементи приемат една и съща стойност.

2.3.3 Разширяване

GraphML е създаден да бъде разширяем. Чрез него лесно може да бъде описана топологията на граф и основните атрибути на възлите и ребрата.. За съхраняването на повече и по-сложна информация, GraphML има възможността да се разширява в две посоки - добавяне XML тагове за разширяване на <data> секцията и добавяне на атрибути. Разширенията на GraphML се дефинират в XML Schema документ. В следващите подточки е представена основната идея за разширяване на синтаксиса.

В повечето случаи допълнителната информация може и е редно да бъде добавяна като GraphML атрибути. Това осигурява разбираемост от други GraphML парсъри. Но все пак понякога е по-удобно да се използват специфични XML атрибути.

Нека си представим, че възлите на граф представят уеб страници. Тогава възелът може да сочи към съответната страница чрез нейния URL адрес, съхранен в xlink:href атрибут. Тъй като за GraphML такъв атрибут не е валиден, неговата дефиниция може да бъде добавена посредством разширяване на GraphML чрез XML Schema. На фиг. 6 е представен Schema документ, който разширява езика, добавяйки нов атрибут към елемента <node>. Добавянето става като старото съдържание на <node> се предефинира и бъде допълнено с новия атрибут, отбелязан в случая като опционален.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
   targetNamespace="http://graphml.graphdrawing.org/xmlns"
   xmlns="http://graphml.graphdrawing.org/xmlns"
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">

<xs:import namespace="http://www.w3.org/1999/xlink"
           schemaLocation="xlink.xsd"/>

<xs:redefine
  schemaLocation="http://graphml.graphdrawing.org/xmlns/1.1/graphml.xsd">
  <xs:attributeGroup name="node.extra.attrib">
    <xs:attributeGroup ref="node.extra.attrib"/>
    <xs:attribute ref="xlink:href" use="optional"/>
  </xs:attributeGroup>
</xs:redefine>

</xs:schema>
```

Фигура 6. Добавяне на атрибут към елемента <node> в GraphML (източник: [1]).

В някои случаи е по-удобно да се използват други XML езици за представяне на данни в GraphML. Например ако потребителят иска да съхранява като възли изображения, написани на SVG (XML-базиран маркиращ език за описване на двумерна векторна графика с възможност за включване и на растерни изображения [3]) както в следващите редове:

```
    ...
xmlns:svg="http://www.w3.org/2000/svg"
    ...
<node id="n0" >
  <data key="k0">
    <svg:svg width="4cm" height="8cm" version="1.1">
      <svg:ellipse cx="2cm" cy="4cm" rx="2cm" ry="1cm" />
    </svg:svg>
  </data>
</node>
    ...
```

GraphML може да бъде разширен с възможност да валидира такъв тип документ.

Произволни елементи могат да бъдат добавяни като съдържание само на елемента <data>. Това решение е взето с цел парсърите да разбират информацията за структурата на графа и да могат да пропускат евентуално непознато съдържание в <data> елемента.

Документът на фиг. 6 представя XML Schema чрез която ще се валидира документ по дадения пример. Тази схема разширява GraphML добавяйки таг <svg:svg> и съответно пространство от имена за него.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
   targetNamespace="http://graphml.graphdrawing.org/xmlns"
   xmlns="http://graphml.graphdrawing.org/xmlns"
   xmlns:svg="http://www.w3.org/2000/svg"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified"
>

<xs:import namespace="http://www.w3.org/2000/svg"
           schemaLocation="svg.xsd"/>

<xs:redefine
    schemaLocation="http://graphml.graphdrawing.org/xmlns/1.1/graphml.xsd">
  <xs:complexType name="data-extension.type">
    <xs:complexContent>
      <xs:extension base="data-extension.type">
        <xs:sequence>
          <xs:element ref="svg:svg"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:redefine>

</xs:schema>
```

Фигура 7. Добавяне на елемент в GraphML (източник: [1]).

2.3.4 Допълнителни концепции

В тази подточка са представени типове графи с по-сложна топология и характеристики. Тъй като не всички приложения поддържат този усложнен синтаксис, за всеки тип е обяснено как приложението процедира с допълнителната информация.

2.3.4.1 Вложени графи

GraphML поддържа вложени графи - граф в който възлите са йерархично организирани. Елемент <node> може да съдържа <graph> елемент, който сам по себе си също да съдържа <node> елементи, които в йерархията са разположени под дадения. На фигура ТРЪЦ е даден примерен документ на вложен граф. Ребрата между дадени два възела трябва да бъдат декларирани в граф, който е "родител" и на двата възела в йерархията. В примера на фиг. 9 декларирането на ребро между възлите n2 и n3 в граф G1 би било неправилно за разлика от декларирането му в граф G0.

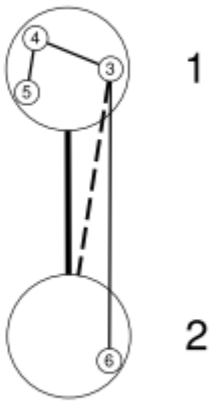
Езикът GraphML съдържа елемент <locator>, който прави възможно дефинирането на част от съдържанието на документа в друг файл. По-точно казано, елементите <graph> и <node> могат да съдържат елемент <locator>, чийто атрибут xlink:href сочи към файл, където съдържанието на този <graph> или <node> е дефинирано. Ако конкретни <graph> или <node> елемент съдържат <locator> това означава, че те не съдържат други елементи. Фрагментът от документ на фиг. 8 казва на парсъра, че съдържанието на граф с id = "G1" е дефинирано във файла content_of_G1.graphml . По същия начин съдържанието на елемент <node> може да бъде пренасочено в друг файл с помощта на <locator>.

```
<graph id="G0" edgedefault="undirected">
  <node id="n1">
    <graph id="G1" edgedefault="undirected">
      <locator xlink:href="content_of_G1.graphml"/>
    </graph>
  </node>
  ...
</graph>
```

Фигура 8. Използване на елемент <locator> (източник: [1]).

Поведението на приложения, които не поддържат вложени графи, е да игнорират възли, които не са декларирани в графа от най-горно ниво и да игнорират ребра, чиито крайни точки не са в графа от най-горно ниво.

```
<graphml>
  <graph id="G0" edgedefault="undirected">
    <node id="n1">
      <graph id="G1" edgedefault="undirected">
        <node id="n3"/>
        <node id="n4"/>
        <node id="n5"/>
        <edge source="n3" target="n4"/>
        <edge source="n4" target="n5"/>
      </graph>
    </node>
    <node id="n2">
      <graph id="G2" edgedefault="undirected">
        <node id="n6"/>
      </graph>
    </node>
    <edge source="n1" target="n2"/>
    <edge source="n3" target="n2"/>
    <edge source="n3" target="n6"/>
  </graph>
</graphml>
```



Фигура 9. Вложен граф (източник: [1]).

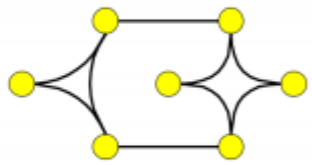
2.3.4.2 Хиперграфи

Хиперграфите представляват обобщение на ребрата в такъв смисъл, че те не само свързват две крайни точки една с друга, а представляват взаимоотношения между

произволен брой възли. Декларират се с <hypergraph> елемент. За всяка крайна точка от хиперреброто, <hyperedge> тагът съдържа <endpoint> елемент. Елементът <endpoint> трябва да съдържа атрибут node, който съдържа id на <node> от документа. На фигура 10 е представен хиперграф с две хиперребра, изобразени с дъги, и две ребра, изобразени с прави линии.

Поведението на приложения, които не поддържат хиперграфи, е да ги игнорират.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml>
  <graph id="G" edgedefault="undirected">
    <node id="n0"/>
    <node id="n1"/>
    <node id="n2"/>
    <node id="n3"/>
    <node id="n4"/>
    <node id="n5"/>
    <node id="n6"/>
    <hyperedge>
       <endpoint node="n0"/>
       <endpoint node="n1"/>
       <endpoint node="n2"/>
     </hyperedge>
    <hyperedge>
       <endpoint node="n3"/>
       <endpoint node="n4"/>
       <endpoint node="n5"/>
       <endpoint node="n6"/>
     </hyperedge>
    <hyperedge>
       <endpoint node="n1"/>
       <endpoint node="n3"/>
     </hyperedge>
    <edge source="n0" target="n4"/>
  </graph>
</graphml>
```



Фигура 10. Хиперграф (източник: [1]).

2.3.4.3 Портове

Един възел може да определи няколко логически локации, където ребрата или хиперребрата да се "закачат". Тези логически локации се наричат портове. Като аналогия можем да си представим графа като платка, възлите са интегрални схеми, а свързващите жици за ребрата. Щифтовете на интегралните схеми са въпросните портове на възела.

Портовете се декларират с елементи <port> като деца на <node>. <port> елементите могат да са вложени, т.е да съдържат други <port> тагове. Всеки <port> трябва да има атрибут name, който да му е идентификатор. Тези имена са уникални в рамките на възела, в който са декларирани портовете. Елементът <edge> има опционални атрибути

sourcepoint и targetpoint, с които реброто може да укаже началния и крайния си порт. <endpoint> елементите също имат опционален атрибут port.
Поведението на приложения, които не поддържат портове, е да ги игнорират.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml>
  <graph id="G" edgedefault="directed">
    <node id="n0">
      <port name="North"/>
      <port name="South"/>
      <port name="East"/>
      <port name="West"/>
    </node>
    <node id="n1">
      <port name="North"/>
      <port name="South"/>
      <port name="East"/>
      <port name="West"/>
    </node>
    <node id="n2">
      <port name="NorthWest"/>
      <port name="SouthEast"/>
    </node>
    <node id="n3">
      <port name="NorthEast"/>
      <port name="SouthWest"/>
    </node>
    <edge source="n0" target="n3"
          sourceport="North" targetport="NorthEast"/>
    <hyperedge>
       <endpoint node="n0" port="North"/>
       <endpoint node="n1" port="East"/>
       <endpoint node="n2" port="SouthEast"/>
     </hyperedge>
  </graph>
</graphml>
```

Фигура 11. Документ на граф с портове (източник: [1]).

2.4 Ограничения при използването на езика

Файловете в разширения синтаксис на GraphML използват имплементацията на информационни средства за достъп на библиотеката на разширения GraphML с цел съхраняване на стойностите на GraphML атрибутите. Обаче тези имплементации не поддържат нито един от TYPE_LONG и TYPE_FLOAT типовете. Затова когато се съхраняват стойности на GraphML атрибути, TYPE_LONG бива превърнат в int, а TYPE_FLOAT в double.

Използването на GraphML чрез функционалността на XSLT е достатъчно ефективно за решаването на по-сложни проблеми свързани с GraphML. Това обаче страда от няколко недостатъка:

- С увеличаването сложността на проблема, стиловите листове стават все по - несъразмерно многословни.

- Алгоритмите трябва да се пренапишат за рекурсивни шаблони, но няма как да се използват съществуващи имплементации.
- Компютърните изчисления могат да имат ниска производителност, особено при голям набор от входни данни. Това често е причинено от прекалено много преплитания на DOM дърветата и излишък от изчисления, генерирани от шаблонно вътрешно инстанциране на XSLT процесора.
- Няма пряк начин за достъпване на системни услуги, като например информационни функции или свързване към бази данни.

3 Сравнителен анализ

Ще сравним GraphML с два други популярни формата за изобразяване на графи: .dl файловете на UCINET и .net файловете на Pajek. Пакетът UCINET се използва широко при анализа на данни за социални мрежи:
https://sites.google.com/site/ucinetsoftware/home
Pajek е програма (на Windows) за анализиране и изобразяване на големи мрежи - с хиляди, дори милиони възли:
http://vlado.fmf.uni-lj.si/pub/networks/pajek/doc/pajekman.pdf

3.1 Критерии за сравнение

Ще сравним тези формати с GraphML по няколко основни критерия:

- Четимост на документа
- Описание на елементите
- Организация на документа
- Поддържани типове
- Достъпност

3.2 Сравнение с език 1

Таблица 1: Сравнение на GraphML с UCINET *.dl файлове

Критерий	GraphML	UCINET *.dl файлове
Четимост на документа	Еднакво лесен за четене както от машини, така и от хора	Опростен, трудно четим синтаксис. Два формата на записване - edgelist1 и fullmatrix.
Описание на елементите	Ребрата и възлите се описват с XML тагове. Поддържа атрибути.	Първо се оказва броят на възлите. Fullmatrix: всеки възел е представен от етикет, графът се представя като квадратна матрица от 0 и 1, където 1 представлява ребро. За ребра с тежест 1 се замества с тежестта на реброто. Edgelist1: представя се списък от възли, ребрата се представят всяко на нов ред като двойки от възли, по възможност следвани от тежестта на реброто. Не поддържа атрибути.
Организация на документа	Всеки GraphML документ започва с <graphml> таг. Всеки граф започва с <graph> таг. Елементите могат да бъдат декларирани в разбъркан ред.	Първо се оказва броят на възлите, след това формата на документа (fullmatrix или edgelist1), последван от списък на възлите, след което е описана матрицата/списъкът на ребрата.
Поддържани типове	Boolean, string, int, double, float, long	Byte, small int, real. Small int изисква 2 byte-а и описва всички цели числа в интервала(-32000,32000). Byte може да опише всички цели числа от 0 до 255. Real описва всички стойности от -1E36 до 1E36.
Достъпност	Безплатен	Платен (има 90-дневен безплатен пробен период)

3.2 Сравнение с език 2

Таблица 2: Сравнение на GraphML с Pajek *.net файлове

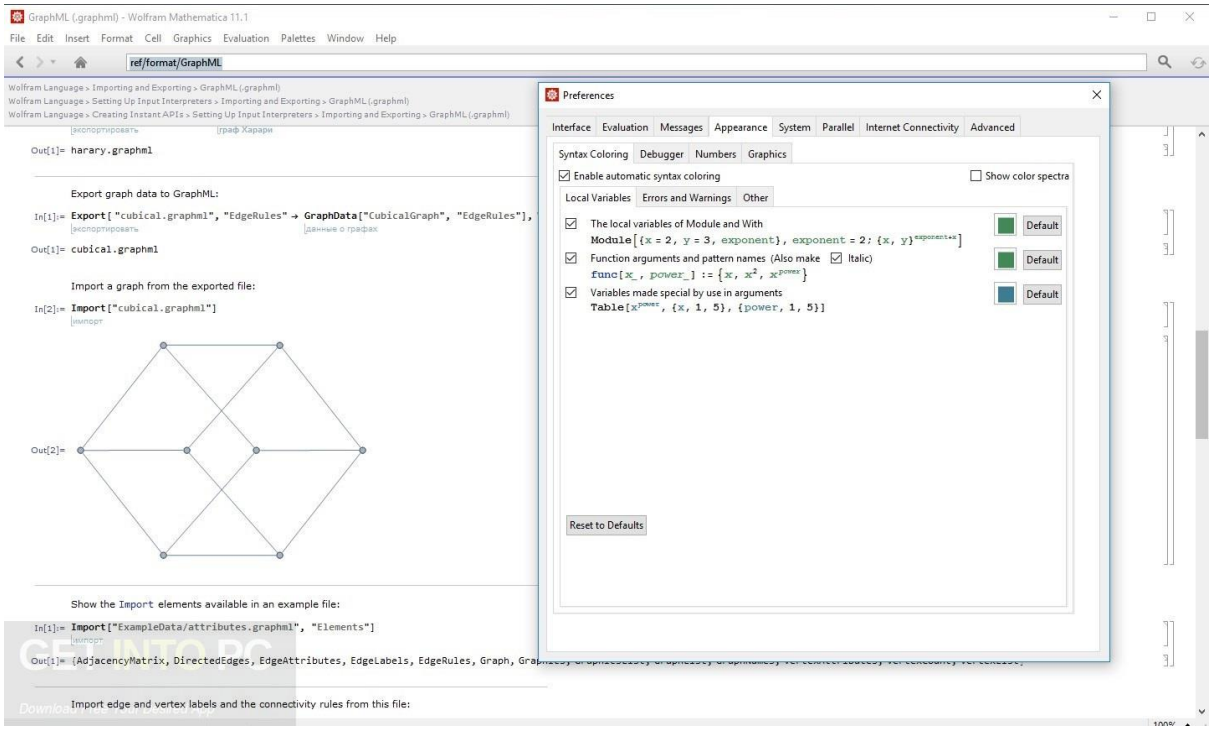
Критерий	GraphML	Pajek *.net файлове
Четимост на документа	Еднакво лесен за четене както от машини, така и от хора	Опростен синтаксис, но трудно четим.
Описание на елементите	Ребрата и възлите се описват с XML тагове. Поддържа атрибути.	Форматът на възлите е идентификатор и етикет на възела, разделени с интервал. Има два начина на записване на ребра - като списък от двойки възли или като списък от n-орки (edgelist), където първият елемент на n-орката е изходният възел, а останалите са съседите му. Теглото на едно ребро се добавя като последна колона в реда на съответното ребро. Не поддържа атрибути.
Организация на документа	Всеки GraphML документ започва с <graphml> таг. Всеки граф започва с <graph> таг. Елементите могат да бъдат декларирани в разбъркан ред.	Документът започва с декларация на броя възли - Vertices N, където N е броят на последващите възли. Всеки ред на документа е елемент. Ребрата се записват след като приключи списъкът на възлите.
Поддържани типове	Boolean, string, int, double, float, long	Int, string, double
Достъпност	Безплатен	Безплатен

4 Примери на използване

GraphML се използва за описание на графи. Описанието им се запазва във файлове с разширение *.graphml. Тези файлове служат за обмен на данни между различни системи.

4.1 Пример 1

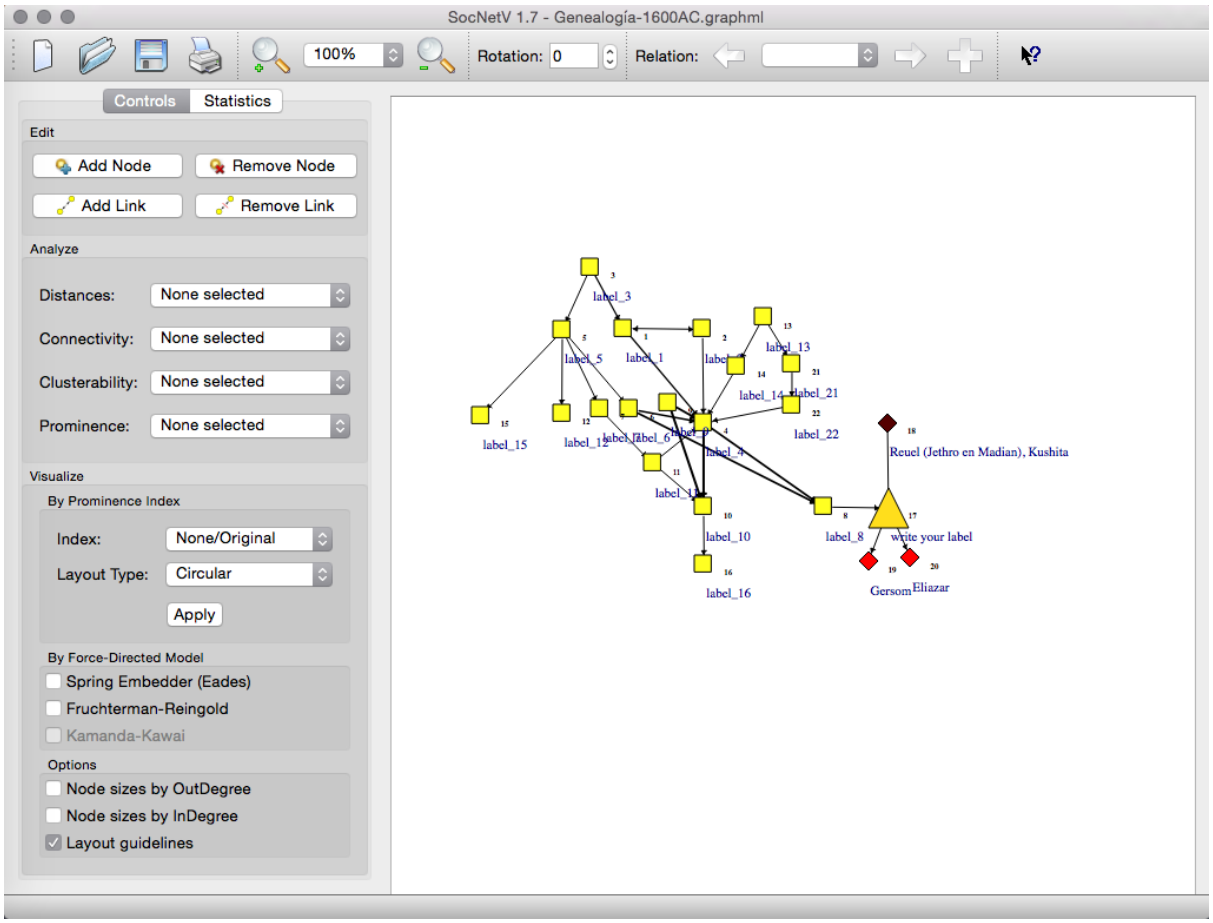
Добър пример за употребата на GraphML са продуктите на Wolfram. Те поддържат Import и Export на *.graphml файлове. Поддържат се стандартните елементи и атрибути, насочени и ненасочени графи, хиперграфи и вложени графи както и дефинирани от потребителя атрибути и елементи.
Примери за използване на *.graphml файл:
`Import["file.graphml"]` - вмъкване на графа/графите, описани в документа
`Export["file.graphml",expr]` - експортира матрица на съседство или информация за ребрата на граф/графи към файл във формата на GraphML
`Import["file.graphml",elem]`- импортира определен елемент от GraphML документ
Тези и други примери могат да бъдат намерени в източник [6]



Фигура 12. Използване на GraphML документ във Wolfram Mathematica. [източник: Google Images]

4.2 Пример 2

Друга система, която поддържа и работи с GraphML е SocNetV. SocNetV е open-source платформа за визуализация и анализ на социални мрежи. Форматът по подразбиране за записване на мрежи е GraphML. Поддържа GraphML файлове и съответните елементи на езика: graph, comment, directed, node, edge, id, label(node), label(edge), source, target.



Фигура 13. Използване на GraphML файл в SocNetV. [източник: Google Images]

5 Добри практики и методи за използване

- <key> елементите да се дефинират в началото на <graphml> тага, преди всички <graph> елементи, за да важат за цялата структура на графа.
- Трябва да се има предвид, че елементите, които приложението, използващо graphml документа, не познава, биват игнорирани. Парсерът може да изкарва грешки при срещането на “непознати” елементи. Възможни такива елементи са <port>, <hyperedge>, <endpoint> и <locator>.
- За приложения, които не поддържат определени типове елементи и видове графи е по - добре те да не бъдат използвани в документите, обработвани от приложението.

- Приложенията, използващи graphml документа и обработващи само единични графи, трябва да имат ясно деклариран начин за справяне с документ с повече от един <graph> елементи. Има два начина за справяне с този проблем:
    - обработване само на първия срещнат граф и пропускане на всички следващи.
    - обработване на обединението на всички графи.

  Във всеки случай, приложението трябва да предупреди при настъпване на такъв случай.
- Приложенията, използващи graphml документа, които не могат да обработват документи с вложени графи, трябва да декларират начин за обработка на такъв тип документи, използвайки някой от следните два начина:
    - игнориране на всички графи от по-ниско ниво.
    - приравняване на нивата на всички графи - графите от ниско ниво се вдигат на по - високо.

  Във всеки случай, приложението трябва да предупреди при настъпване на такъв случай.
  Разбира се, разработчиците могат да дефинират собствена практика за справяне с горните два проблема.

6 Заключение и очаквано бъдещо развитие

С развитието на семантичния Уеб, XML и базираните на него езици намират широко приложение. Форматът GraphML е лесен за разбиране и удобен за използване както от хора, така и от програми. Това предполага, че в бъдеще той може да намери приложение при описание на набиращите популярност социални мрежи, описанието на свързани данни в помощ на самообучаващи се програми и статистически проучвания.

7 Разпределение на работата

Използвахме споделен документ в Google Docs, за да може всеки по всяко време да изпълнява своята част от заданието в общия документ. Разделихме работата на равни части, които зависеха една от друга, задължавайки всеки да познава материала, написан от другия, за да може да пише по своята част. Редовно обсъждахме последните промени. Всеки даваше идеи и насоки на другия за оформлението на съдържанието.

8 Използвани литературни източници

1. https://cs.brown.edu/~rt/gdhandbook/chapters/graphml.pdf
2. http://graphml.graphdrawing.org/xmlns/1.1/graphml.xsd
3. https://bg.wikipedia.org/wiki/SVG
4. https://gephi.org/users/supported-graph-formats/ucinet-dl-format/
5. https://gephi.org/users/supported-graph-formats/pajek-net-format/
6. http://reference.wolfram.com/language/ref/format/GraphML.html