Софийски университет „Св. Кл. Охридски”

Факултет по математика и информатика

Бакалавърска програма
„Софтуерно инженерство”

Предмет: XML технологии за семантичен Уеб

Зимен семестър, 2021/2022 год.

Тема 63: Преглед на Graph Markup Language (GraphML)

Ece

Автори:

Мирослав Дионисиев, фак. номер 62390

Павел Сарлов, фак. номер 62393

декември, 2021

София

Съдържание

1 Въведение 3
2 Характеристики и използване на GraphML 3
2.1 Дефиниции 3
2.2 Основни характеристики 4
2.2.1 Разширяемост 4
2.2.2 Опростеност 4
2.2.3 Генерализация 4
2.2.4 Устойчивост 4
2.2.5 XML синтаксис 4
2.3 Въведение в използването на GraphML 5
2.3.1 Заглавна част 5
2.3.2 Самият граф 5
2.3.3 Атрибути 6
2.3.4 Информация за парсера 9
2.3.5 Допълнителни концепции I: вложени графи, хиперграфи и портове 11
2.3.6 Допълнителни концепции II: разширение на GraphML 15
2.4 Ограничения при използването на GraphML 19
3 Сравнителен анализ 19
3.1 Критерии за сравнение 19
3.2 Сравнение с GXL 19
3.3 Сравнение с DyNetML 20
4 Примери на използване 21
4.1 yEd 21
4.2 Gephi 23
5 Добри практики и методи за използване 24
6 Заключение и очаквано бъдещо развитие 24
7 Разпределение на работата 24
8 Използвани литературни източници 24

1 Въведение

Науката, която се занимава с изучаване на графите и задачите свързани с тях, се нарича „Теория на графите“. Началото ѝ е поставено от математика Леонард Ойлер, който решава задачата за седемте моста на Кьонигсберг, която търси маршрут от квартал в града, минава през всичките седем моста точно по веднъж и завършва в същия квартал.

В практиката множеството от софтуерни системи използват графите като средство за решаване на разнообразни по своята същност проблеми. Примери за това са приложенията за навигация като Google Maps, GPS системите и други. Графи се използват също така и при съхранението на данни, защото улесняват съхраняването както на структурирани, така и на неструктурирани такива.

Системите, използващи данни под формата на графи, имат нужда от формат, под който да съхраняват и обменят тази информация. Разбира се, при различните системи тези дейности се извършват по различен начин и през годините са правени много опити за унифицирането на комуникацията помежду им.

Мотивирани от целите за оперативна съвместимост на инструментите, достъп до набори от сравнителни данни и обмяна на данни в мрежата, Управителният комитет на Симпозиума по изобразяване на графи стартира нова инициатива с неформален семинар, проведен във връзка с 8-мия Симпозиум по изобразяване на графи (GD 2000). В резултат на това се сформира неформална работна група, която да предложи модерен формат за обмен на данни между инструменти за изобразяване на графи и други приложения. Този формат бива наречен Graph Markup Language (GraphML), който благодарение на XML-базирания си синтаксис, може да се използва в комбинация с други XML-базирани формати. (източник: [1])

2 Характеристики и използване на GraphML

2.1 Дефиниции

- Граф – 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.

- Възел – елемент на графа, в който се съдържа основната негова информация. Всеки възел може да съдържа данни.

- Ребро – представлява връзката между възлите. Ребрата биват насочени и ненасочени. Всяко ребро може да притежава тегло, чрез което да се сравни с другите ребра.

2.2 Основни характеристики

Синтаксисът на GraphML е описан в GraphML Schema[1]. Въпреки че това е основната дефиниция, също така се използва и по-слаба спецификация в GraphML DTD, която не различава референтни типове като идентификатори на възли и ребра. Въпреки това, някои приложения се нуждаят от DTD, за да работят коректно.

2.2.1 Разширяемост

В GraphML, освен стандартните данни, може да се съхранява още разнообразна по вид и формат информация. Това се постига чрез промени в дефиницията на GraphML схемата, като се добавят XML тагове и атрибути. По този начин може да се предостави допълнителна информация на приложения, които я изискват, или дааа се съставят по-сложни екземпляри.

2.2.2 Опростеност

Форматът GraphML е създаден с цел лесно да се анализира и интерпретира както от хора, така и от машини. Не съществуват неясноти – всеки валиден GraphML документ има своя единствена интерпретация.

2.2.3 Генерализация

Няма ограничения относно теорията на графите – т.е. йерархични графи, хиперграфи и прочие могат да бъдат представяни чрез основния синтаксис на езика, без да е нужно да се добавят разширения към синтаксиса.

2.2.4 Устойчивост

Успоредно със своята разширяемост, GraphML предоставя и прозрачност на добавената информация. Това означава, че системи, които не могат да обработват всички видове графи или добавената към тях информация, могат лесно да извличат единствено тази, която им е нужна.

2.2.5 XML синтаксис

Благодарение на своя XML синтаксис, GraphML може да се използва в комбинация с други XML-базирани формати. Тъй като изобразяването на GraphML графи често се нуждае от предварителна обработка или трансформация до други XML формати, изключително удобно е да се използва XSLT, език специално създаден за трансформацията на XML документи.

[1] GraphML Schema

2.3 Въведение в използването на GraphML

2.3.1 Заглавна част

За да бъде валиден, всеки XML документ трябва да започва с декларация на DTD или XML Schema. Езикът на GraphML е дефиниран чрез гореспоменатата схема. На фиг. 1 е показан минимума, изискван от един GraphML документ.

```
<?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 1: минимален валиден GraphML документ. (източник: [1])

Първият ред от документа на фиг. 1 е процесорна XML инструкция, която дефинира стандарта (XML 1.0) и кодирането му (UTF-8). Вторият ред съдържа кореновия елемент на един GraphML документ: *graphml*, който подобно на всички други GraphML елементи принадлежи на пространството от имена http://graphml.graphdrawing.org/xmlns. Поради тази причина пространството от имена по подразбиране, дефинирано чрез XML атрибута *xmlns="http://graphml.graphdrawing.org/xmlns"*, е именно това. Следващите два атрибута декларира коя XML Schema е използвана за валидация на документа. Атрибутът *xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"* дефинира *xsi* като префиксно пространство от имена за схемата, а атрибутът *xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns http://graphml.graphdrawing.org/xmlns/1.1/graphml.xsd"* дефинира локацията на XML схемата за пространството от имена на 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 изглежда 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 2: Минимален GraphML документ без референция към схема. (източник: [1])

2.3.2 Самият граф

Един граф се определя от *graph* елемент. В него се влагат декларациите на възлите и ребрата. Възлите се декларират посредством *node* елемента, а ребрата - *edge* елемента. В GraphML няма

определен ред на дефиниране на възли и ребра. Следователно следния пример е напълно валиден GraphML фрагмент:

```
<graph id="G" edgedefault="directed">
  <node id="n0"/>
  <edge source="n0" target="n2"/>
  <node id="n1"/>
  <node id="n2"/>
  ...
</graph>
```

Фигура 3: Граф с неопределен ред на дефинирани ребра и възли. (източник: [2])

- Графите в GraphML са смесени, т.е. съдържат едновременно насочени и ненасочени ребра. Ако няма определена посока на реброто, тя се определя от посоката по подразбиране, която се декларира чрез XML атрибута *edgedefault* на *graph* елемента. Двете възможни стойности на този атрибут са *directed* и *undirected*. Добре е да се отбележи, че посоката по подразбиране трябва да се специфицира.
- Възлите на графа се декларират чрез *node* елемента, като всеки възел си има идентификатор, който трябва да бъде уникален в целия документ, т.е. в един документ не трябва да има два възела със съвпадащи идентификатори. Идентификаторът на един възел се дефинира чрез XML атрибута *id*.
- Ребрата на графа се декларират чрез *edge* елемента. Всяко ребро трябва да дефинира двете си крайни точки чрез XML атрибутите *source* и *target*. Стойността на *source*, съответно на *target*, трябва да бъде идентификатор на възел от същия документ. Ребра само с една крайна точка, също наречени цикли, се дефинират чрез еднакви стойности за *source* и *target*. Незадължителният атрибут *directed* определя дали реброто е насочено или не. Стойността *true* декларира насочено ребро, обратно *false* – ненасочено ребро. Ако посоката не е строго дефинирана се използва посоката по подразбиране за съответното ребро, дефинирана в обхващащия го граф (Фиг. 4).

2.3.3 Атрибути

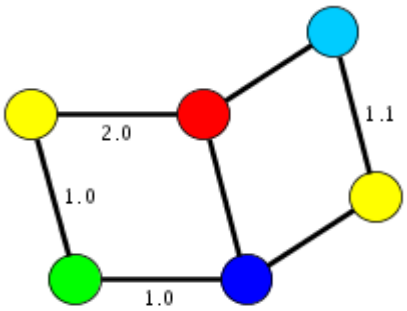
```
...
<edge id="e1" directed="true" source="n0" target="n2"/>
...
```

Фигура 4: Дефиниция на ребро с всички XML атрибути. (източник: [2])

В предишната секция разгледахме как се описва топологията на граф в GraphML. Въпреки че чисто топологична информация е достатъчна за някои приложения на GraphML, в повечето случаи е нужда и допълнителна информация. С помощта на разширението GraphML-Attributes може да се специфицира допълнителна информация от прост тип (числови стойности или низове) за елементите на графа.

Ако се налага да се добави структурирано съдържание към елементите на графа, трябва да се използва разширения механизъм на GraphML за ключ/данни. Самите GraphML атрибути са специализирани такива разширения. GraphML атрибутите не бива да се бъркат с XML атрибутите, които са напълно различна концепция.

Нека разгледаме следния пример на граф с оцветени възли и претеглени ребра:



Фигура 5: Граф с оцветени възли и претеглени върхове. (източник: [2])

```
<?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/1.0/graphml.xsd">
  <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 6: GraphML документ с GraphML атрибути. (източник: [2])

Ще използваме GraphML атрибутите за съхранението на допълнителните данни за възлите и ребрата. Можем да видим резултата на фиг. 6.

GraphML атрибутите се дефинират чрез *key* елемент, който определя идентификатор, име, тип и домейн на атрибута.

- Идентификаторът се специфицира от XML атрибута *id* и се използва за реферирането към GraphML атрибут в документа.
- Името на GraphML атрибута се определя от XML атрибута *attr.name* и трябва да бъде уникално сред всички декларирани GraphML атрибути в документа. Целта на името е да позволи на приложенията да идентифицират значението на атрибута. Хубаво е да се отбележи, че името на GraphML атрибута не се използва вътре в самия документ, за тази цел се използва идентификаторът.
- Типът на GraphML атрибутите може да бъде един от *boolean*, *int*, *long*, *float*, *double* или *string*. Тези типове са дефинирани подобно на типовете в Java програмния език.
- Домейнът на GraphML атрибутите определя за кои елементи от графа е деклариран самия атрибут. Възможни стойности включват *graph*, *node*, *edge* и *all*.

```
...
<key id="d1" for="edge" attr.name="weight" attr.type="double"/>
...
```

Фигура 7: Декларация на GraphML атрибут. (източник: [2])

Възможно е също така да се дефинираа и стойност по подразбиране на GraphML атрибут. Това се определя от текстовото съдържание на *default* елемента.

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

Фигура 8: Декларация на GraphML атрибут със стойност по подразбиране. (източник: [2])

Стойностите на GraphML атрибут за елемент от графа се дефинират чрез елемента *data*, който се влага в съответния елемент от графа. За *data* се използва XML атрибута *key*, който реферира към идентификатора на GraphML атрибута. Стойността на GraphML атрибута е текстовото

съдържание на *data* елемента. Тази стойност трябва да бъде със същия тип, деклариран в съответната *key* дефиниция.

```
...
<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"/>
  ...
  <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"/>
  ...
</graph>
...
```

Фигура 9: Стойност на GraphML атрибут. (източник: [2])

Може да има елементи от графа, за който е дефиниран GraphML атрибут, но няма декларирана стойност чрез съответен *data* елемент. Ако име дефинирана стойност по подразбиране за този GraphML атрибут, то се използва тя. В примера на фиг. 9 няма дефинирана стойност за възела с идентификатор *n1* и GraphML атрибутът с име *color*. Следователно този GraphML атрибут има стойност по подразбиране *yellow* за текущия възел. Ако няма специфицирана стойност по подразбиране, както при GraphML атрибута *weight*, стойността на съответния атрибут не е дефинирана за елемента на графа. В примера от фиг. 9 стойността на GraphML атрибута *weight* за реброто с идентификатор *e3* не е дефинирана.

2.3.4 Информация за парсера

С цел да се направи възможна имплементацията на оптимизирани парсери на GraphML документи може да се прикрепят метаданни към XML атрибутите на някои GraphML елементи. Всички XML атрибути, обозначаващи метаданни, се предхождат от *parse*. Съществуват два вида метаданни:

- информация за броя на елементите – дефинирани са следните XML атрибути:
  - за *graph* елемента:
    - *parse.nodes* – обозначава броя на възлите в графа;
    - *parse.edges* – обозначава броя на ребрата в графа;
    - *parse.maxindegree* – обозначава максималния брой на входящите ребра;

        - *parse.maxoutdegree* – обозначава максималния брой на изходящите ребра;
    - за *node* елемента:
        - *parse.indegree* – обозначава броя на входящите ребра;
        - *parse.outdegree* – обозначава броя на изходящите ребра;
- информация за кодировката на данните в документа – дефинирани са следните XML атрибути:
    - за *graph* елемента:
        - *parse.nodeids* – има стойност *canonical*, всички възли имат идентификатори следващи шаблона *nX*, където *X* обозначава броя на срещанията на предходния *node* елемент; в противен случай стойността на атрибута е *free*;
        - *parse.edgeids* – аналогично на *parse.nodeids* с единствената разлика, че идентификаторите на ребрата следват шаблона *eX*;
        - *parse.order* – обозначава подредбата на *node* и *edge* елементите в документа; при стойност *nodesfirst* не е позволено на *node* елемент да присъства след появата на първия *edge* елемент; при стойност *adjacencylist*, декларацията на възел се следва от декларацията на съседните му ребра; при стойност *free* няма строга подредба;

На фиг. 10 се демонстрира информацията от метаданни за парсера от нашия пример:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This file was written by the JAVA GraphML Library.-->
<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/1.0/graphml.xsd">
  <graph id="G" edgedefault="directed"
           parse.nodes="11" parse.edges="12"
           parse.maxindegree="2" parse.maxoutdegree="3"
           parse.nodeids="canonical" parse.edgeids="free"
           parse.order="nodesfirst">
    <node id="n0" parse.indegree="0" parse.outdegree="1"/>
    <node id="n1" parse.indegree="0" parse.outdegree="1"/>
    <node id="n2" parse.indegree="2" parse.outdegree="1"/>
    <node id="n3" parse.indegree="1" parse.outdegree="2"/>
    <node id="n4" parse.indegree="1" parse.outdegree="1"/>
    <node id="n5" parse.indegree="2" parse.outdegree="1"/>
    <node id="n6" parse.indegree="1" parse.outdegree="2"/>
    <node id="n7" parse.indegree="2" parse.outdegree="0"/>
    <node id="n8" parse.indegree="1" parse.outdegree="3"/>
    <node id="n9" parse.indegree="1" parse.outdegree="0"/>
    <node id="n10" parse.indegree="1" parse.outdegree="0"/>
    <edge id="edge0001" source="n0" target="n2"/>
    <edge id="edge0002" source="n1" target="n2"/>
    <edge id="edge0003" source="n2" target="n3"/>
    <edge id="edge0004" source="n3" target="n5"/>
    <edge id="edge0005" source="n3" target="n4"/>
    <edge id="edge0006" source="n4" target="n6"/>
    <edge id="edge0007" source="n6" target="n5"/>
    <edge id="edge0008" source="n5" target="n7"/>
    <edge id="edge0009" source="n6" target="n8"/>
    <edge id="edge0010" source="n8" target="n7"/>
    <edge id="edge0011" source="n8" target="n9"/>
    <edge id="edge0012" source="n8" target="n10"/>
  </graph>
</graphml>
```

Фигура 10: Граф с допълнителна метаданни за парсера. (източник: [2])

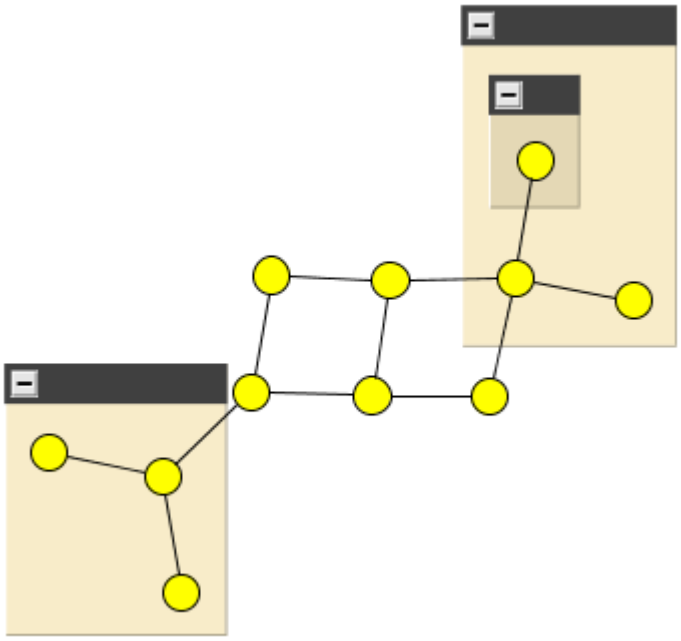
2.3.5 Допълнителни концепции I: вложени графи, хиперграфи и портове

Ще разгледаме някои допълнителни топологични черти на графите, тъй като в някои приложения моделът от предишните секции е прекалено ограничен и не позволява адекватното моделиране на данните. В тази секция ще разширим концепцията като добавим вложени графи, хиперграфи и портове. Тъй като не всички приложения могат да поддържат този разширен синтаксис, в края на всяка подсекция ще бъде обяснено как да се процедира.

2.3.5.1 Вложени графи

GraphML поддържа графи, в които възлите са йерархично подредени. Йерархията се изразява чрез структурата на GraphML документа. Всеки възел в документа може да има *graph*

елемент, самия който може да съдържа възли, които са в йерархията на съответния възел. Нека да разгледаме примера на фиг. 11 и съответстващия ѝ GraphML документ на фиг. 12.



Фигура 11: Вложен граф. (източник: [2])

```
<?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/1.0/graphml.xsd">
  <graph id="G" edgedefault="undirected">
    <node id="n0"/>
    <node id="n1"/>
    <node id="n2"/>
    <node id="n3"/>
    <node id="n4"/>
    <node id="n5">
        <graph id="n5:" edgedefault="undirected">
          <node id="n5::n0"/>
          <node id="n5::n1"/>
          <node id="n5::n2"/>
          <edge id="e0" source="n5::n0" target="n5::n2"/>
          <edge id="e1" source="n5::n1" target="n5::n2"/>
        </graph>
    </node>
    <node id="n6">
        <graph id="n6:" edgedefault="undirected">
          <node id="n6::n0">
              <graph id="n6::n0:" edgedefault="undirected">
                <node id="n6::n0::n0"/>
               </graph>
          </node>
          <node id="n6::n1"/>
          <node id="n6::n2"/>
          <edge id="e10" source="n6::n1" target="n6::n0::n0"/>
          <edge id="e11" source="n6::n1" target="n6::n2"/>
        </graph>
    </node>
    <edge id="e2" source="n5::n2" target="n0"/>
    <edge id="e3" source="n0" target="n2"/>
    <edge id="e4" source="n0" target="n1"/>
    <edge id="e5" source="n1" target="n3"/>
    <edge id="e6" source="n3" target="n2"/>
    <edge id="e7" source="n2" target="n4"/>
    <edge id="e8" source="n3" target="n6::n1"/>
    <edge id="e9" source="n6::n1" target="n4"/>
  </graph>
</graphml>
```

Фигура 12: GraphML документ с вложени графи. (източник: [2])

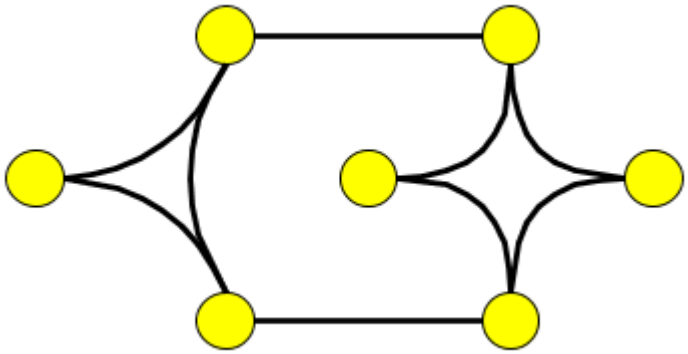
Трябва да се отбележи, че йерархията в представения граф е изразена чрез вложено ограничение, т.е. възел *А* е под възел *Б* в йерархията тогава и само тогава, когато графичната презентация на *А* е изцяло в графичната презентация на *Б.*

Ребрата между два възела във вложен граф трябва да бъдат декларирани в граф, който е предшественик и на двата възела в йерархията. Това е в сила и при нашия пример. Декларацията на ребро между възел *n6::n1* и възел *n4::n0::n0* в граф *n6::n0* би била грешна, докато декларирането му в граф *G* ще бъде правилно.

За приложения, които не могат да боравят с вложени графи, решението е да се игнорират възли, които не се съдържат в най-горния в йерархията граф и също възли, чиито крайни точки не са в същия граф.

2.3.5.2 Хиперграф

Хиперграфите съдържат хиперребра, които са обобщение на ребрата, в смисъл че не представляват връзка между две крайни точки – те изразяват отношението между произволен брой такива. Хиперребрата се декларират с елемента *hyperedge* в GraphML. За всяка крайна точка от хиперреброто то съдържа елемент *endpoint* в себе си, който от своя страна трябва да има XML атрибут *node*, съдържащ идентификатора на възел в документа. Ще разгледаме примера от фиг. 13 и 14. На него е показан граф с две хиперребра и две обикновени ребра. Хиперребрата са илюстрирани със свързващи дъги, докато обикновените ребра с прави линии. Забележете, че ребрата могат да бъдат определени чрез елемент *edge* или елемент *hyperedge*, съдържащ два *endpoint* елемента.



Фигура 13: Граф с хиперребра. (източник: [2])

```
<?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/1.0/graphml.xsd">
  <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 14: GraphML документ с хиперребра. (източник: [2])

```
<?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/1.0/graphml.xsd">
  <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 15: Документ с портове. (източник: [2])

Подобно на обикновените ребра, хиперребрата и крайните точки също могат да имат XML атрибут *id*, който дефинира уникален идентификатор за съответния елемент.

2.3.5.3 Портове

Всеки възел може да специфицира различни логически крайни точки, към които ребрата и хиперребрата да се прикачат. Всяка логическа крайна точка се нарича *порт.* По аналогия може да разгледаме примера с една дънна платка като граф, чиповете като възли и свързващите ги жици като ребра. Тогава пиновете на конекторите на всеки чип съответстват на портовете на един възел.

Портовете на всеки възел се декларират чрез *port* елемента като деца на съответните *node* елементи. Забележете, че порт елементите могат да бъдат вложени, т.е. може да съдържат *port* елементи в себе си. Всеки *port* елемент трябва да има XML атрибут *name,* който идентифицира този порт. Елементът *edge* има незадължителни XML атрибути *sourceport* и *targetport*, чрез които да специфицира портовете на възела източник и съответно възела цел. Аналогично, елементът *endpoint* има незадължителен XML атрибут *port.* Използването на портове и тези атрибути можем да видим на фиг. 15.

2.3.6 Допълнителни концепции II: разширение на GraphML

GraphML е проектиран да бъде разширяем. Чрез GraphML топологията на един граф и простите атрибути на неговите елементи могат да бъдат сериализирани. За съхранението на по-сложни приложни данни се налага разширението на GraphML. В тази секция ще разгледаме различните възможните за тази цел.

Разширенията на GraphML трябва да бъдат дефинирани от XML Schema. Схемата, която дефинира разширението, може да произлиза от GraphML Schema документи чрез използването на стандартен механизъм, подобен на този при XHTML.

2.3.6.1 Добавяне на XML атрибут към GraphML елементи

В повечето случаи допълнителна информация може (и би трябвало) да бъде прикрепена към GraphML елементите чрез GraphML атрибути. Това гарантира четимостта за други GraphML парсери. Въпреки това, понякога е по-удобно да се използват XML атрибути. Да предположим, че имаме парсер, който знае XLink атрибутът *href* и го интерпретира коректно като URL. Допълнително, нека искаме да съхраним граф, чиито възли моделират WWW страници, в GraphML. За да асоцииране възел към страницата, която моделира, трябва да добавим URL на съответната страница като *xlink:href* атрибут в елемента *node.*

```
    ...
<node id="n0" xlink:href="http://graphml.graphdrawing.org"/>
    ...
```

Фигура 16: node елемент сочещ към URL . (източник: [2])

За да добавим XML атрибути към GraphML елементи, се налага да разширим GraphML. Това разширение може да бъде дефинирано чрез XML Schema. Документът graphml+xlink.xsd показва как *href* атрибутът е добавен към *node*. Добре е да се отбележи, че до този момент няма официална

Schema дефинирана за XLink; за валидирането на следния файл трябва да се уверите, че атрибутът *schemaLocation* сочи към предходна версия, напр. xlink.xsd.

```
<?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/1.0/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 17: Разширение на GraphML - атрибути. (източник: [2])

Частите на документа от фиг. 17 имат следната функция: документът graphml+xlink.xsd има *schema* елемент за свой корен. *targetNamespace="http://graphml.graphdrawing.org/xmlns"* посочва, че езикът, дефиниран от текущия документ, е GraphML. Следващите три реда определят пространството от имена по подразбиране и префиксите за пространството от имена на XLink и XMLSchema. Атрибутите *elementFormDefault* и *attributeFormDefault* не са значими за текущия пример. *<xs:import namespace="http://www.w3.org/1999/xlink" schemaLocation="xlink.xsd"/>* дава достъп до пространството от имена на XLink, намиращо се на файл *xlink.xsd*. *<xs:redefine schemaLocation="http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">* определя файлът, (част от) който е предефинирана. Групата от атрибути *node.extra.attrib* е включена в списъка от атрибути

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
            xmlns:xlink="http://www.w3.org/1999/xlink"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
                                graphml+xlink.xsd">
  <graph edgedefault="directed">
    <node id="n0" xlink:href="http://graphml.graphdrawing.org"/>
    <node id="n1" />
    <edge source="n0" target="n1"/>
  </graph>
</graphml>
```

Фигура 18: GraphML документ с допълнителни XML атрибути. (източник: [2])

на *node*. След предефиниция тази атрибутна група включва старото си съдържание плюс още един атрибут – *xlink:href* – който не е задължителен.

Освен *node.extra.attrib* има и съответстващи на всички главни GraphML елементи групи от атрибути. На фиг. 18 можем да видим пример за документ, който е валиден спрямо схемата graphml+xlink.xsd.

2.3.6.2 Добавяне на сложни типове

Структурираното съдържание може да бъде добавено в *data* елемент. Например, потребител може да иска да съхрани снимки за възлите под формата на SVG (фиг. 19).

```
    ...
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>
    ...
```

Фигура 19: Елемент node *и графичното му представяне. (източник: [2])*

За да добавим структурирани данни към GraphML елементи, трябва да разширим GraphML. Това разширение може да бъде дефинирано чрез XML Schema. Документът graphml+svg.xsd показва как SVG елементи са добавени към съдържанието на *data.* Добре е да се отбележи, че до този момент няма официална Schema дефиниция за SVG. За валидация следния файл, трябва да сме сигурни, че атрибутът *schemaLocation* сочи към предходна версия, например SVG.xsd.

```
<?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/1.0/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 20: Разширение на GraphML - структурирани данни. (източник: [2])

На фиг. 20 е показана Schema, подобна на примера в подточка 2.3.6.1. Първо е нужно да се направят декларациите на пространствата от имена. След това се въвежда пространството от имена на SVG. Накрая, сложният тип *data-extension.type*, който е база на съдържанието в *data* елемента, се разширява от SVG елемента *svg*. Чрез Schema документа graphml+svg.xsd може да се валидира GraphML документа на фиг. 21.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
         xmlns:svg="http://www.w3.org/2000/svg"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
                             graphml+svg.xsd">
  <key id="k0" for="node">
    <default>
      <svg:svg width="5cm" height="4cm" version="1.1">
        <svg:desc>Default graphical representation for nodes
        </svg:desc>
        <svg:rect x="0.5cm" y="0.5cm" width="2cm" height="1cm"/>
      </svg:svg>
    </default>
  </key>
  <key id="k1" for="edge">
    <desc>Graphical representation for edges
    </desc>
  </key>
  <graph edgedefault="directed">
    <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>
    <node id="n1" />
    <edge source="n0" target="n1">
      <data key="k1">
        <svg:svg width="12cm" height="4cm" viewBox="0 0 1200 400">
          <svg:line x1="100" y1="300" x2="300" y2="100"
           stroke-width="5"  />
        </svg:svg>
      </data>
    </edge>
  </graph>
</graphml>
```

Фигура 21: GraphML документ включващ SVG данни. (източник: [2])

Забележете, че възелът с идентификатор *n1* приема подразбиращото се графично представяне, въведени в *key k0.* Примера от фиг. 21 показва също така ползата от XML пространствата от имена: има два различни *desc* елемента – един в пространството от имена на GraphML и един в това на SVG. Възможните конфликти, породени от идентичността на имената на елементи от различни XML езици, са разрешени чрез различни пространства от имена.

2.4 Ограничения при използването на GraphML

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 други два XML-базирани формата за представяне на графи – GXL [4], създаден с цел позволяване на интероперативност между софтуерни реинженерни инструменти и компоненти (парсери, анализатори, визуализатори), и DyNetML [5], чиято идея е представянето на сложни социално-мрежови графи чрез използването на *MetaMatrix*, позволяващо определянето на връзките между агенти, задачи, ресурси, знания и т.н.

3.1 Критерии за сравнение

Ще сравним горепосочените формати с GraphML по следните критерии:

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

3.2 Сравнение с GXL

Таблица 1: Сравнение на GraphML с GXL

Критерий	GraphML	GXL
Четимост на документа	Еднакво лесен за четене както от машини, така и от хора.	Гъвкав език, предоставящ интероперативност между различни граф-базирани инструменти.
Описание на елементите	Ребрата и възлите се описват с XML тагове (*node*, *edge*). Поддържа използването на атрибути.	Ребрата и възлите се описват с XML тагове (*node*, *edge*). Поддържа използването на атрибути.
Организация на документа	Всеки GraphML документ започва с *graphml* елемент. Всеки граф се определя от *graph* елемент. Няма определена подредба на елементите.	Всеки GXL документ започва с *gxl* елемент. Всеки граф се определя от *graph* елемент. Няма определена подредба на елементите.
Поддържани типове	Булеви стойности (boolean), цели числа (int, long), числа с плаваща запетая (float, double), низове (string).	Булеви стойности (bool), цели числа (int), числа с плаваща запетая (float), низове (string), енумератори (enum), сложни/композитни типове (seq, set, bag, tup), локатори (locator).
Достъпност	Безплатен.	Безплатен.

3.3 Сравнение с DyNetML

Таблица 2: Сравнение на GraphML с DyNetML

Критерий	GraphML	DyNetML
Четимост на документа	Еднакво лесен за четене както от машини, така и от хора.	Еднакво лесен за четене както от машини, така и от хора.
Описание на елементите	Ребрата и възлите се описват с XML тагове (*node*, *edge*). Поддържа използването на атрибути.	Ребрата и възлите се описват с XML тагове. Поддържа използването на атрибути. Възлите имат допълнително типове.

Организация на документа	Всеки GraphML документ започва с *graphml* елемент. Всеки граф се определя от *graph* елемент. Няма определена подредба на елементите.	Всеки DyNetML документ започва с *DynamicNetwork* елемент, в който присъства елемент *MetaMatrix*, съдържащ списък *nodes* от множества от възли дефинирани чрез *nodeset,* и списък от образуваните графи *networks*, съдържащ *graph* елементи с информация за неговите ребра.
Поддържани типове	Булеви стойности (boolean), цели числа (int, long), числа с плаваща запетая (float, double), низове (string).	Булеви стойности (boolean), числа с плаваща запетая (double), низове (string), празни стойности (void).
Достъпност	Безплатен.	Безплатен.

4 Примери на използване

Своето приложение GraphML намира в описанието на графи. GraphML файловия формат (с разширение *.graphml) е взаимния труд на общността в сферата на изобразяването на графи за дефиниране на всеобщ формат за обмен на данни между различни системи.

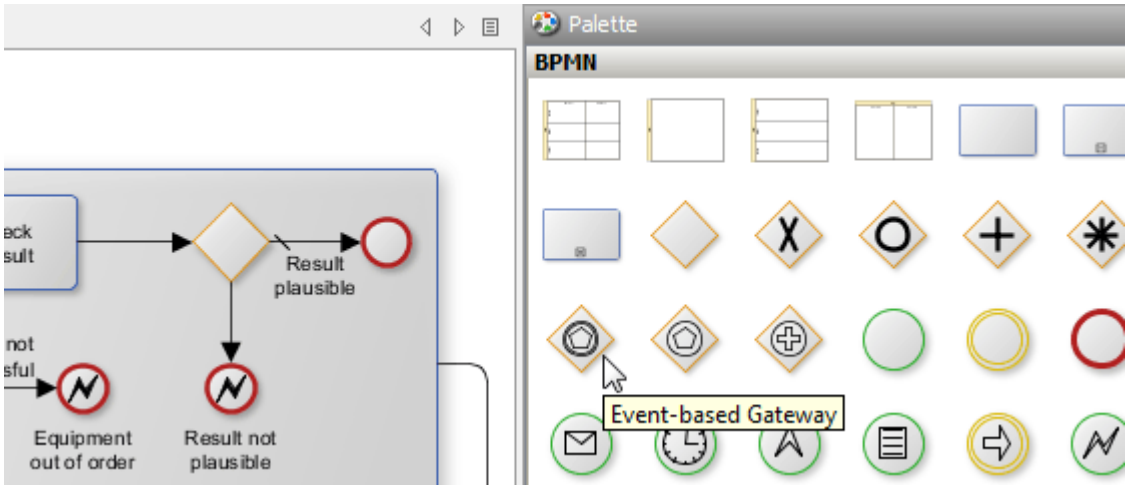
4.1 yEd

Основен пример за употребата на GraphML са продуктите на *yWorks* (източник: [6]) и по конкретно тяхното мощно десктоп приложение *yEd,* което може да бъде използвано за бързо и ефективно генериране на висококачествени диаграми. Приложението работи на всички Windows, Unix/Linux и Mac OS версии и поддържа импортиране и експортиране на файлове в GraphML формат.

Чрез *yEd* създаването на диаграми става като детска игра, благодарение на изобилието си от мощни инструменти и интуитивен потребителски интерфейс. Системата поддържа екстензивна палета от готови елементи, включващи:

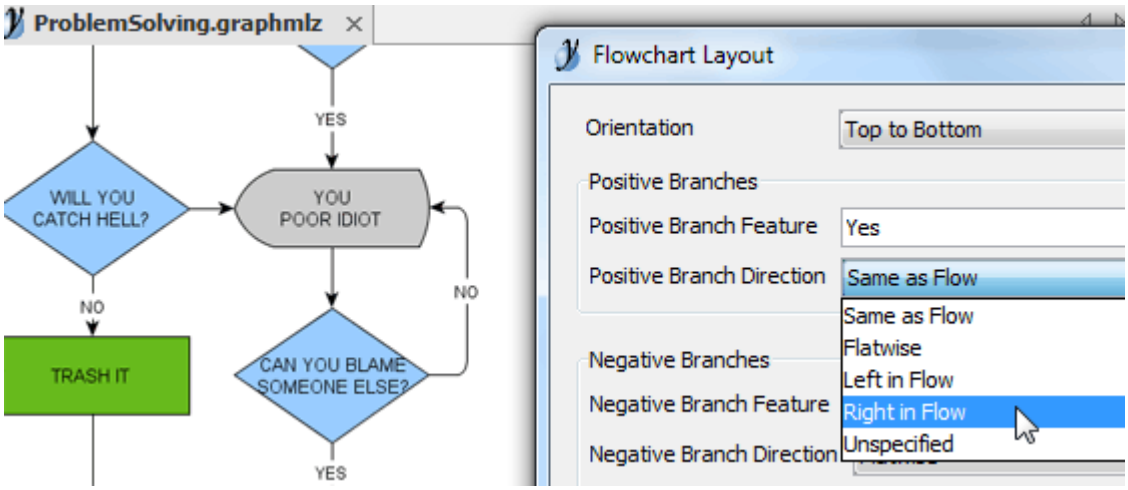
- разнообразие от възлови форми и типове на ребра;
- групиране на възли;

- UML елементи за класови диаграми;



Фигура 22: yEd палети. (източник: [6])

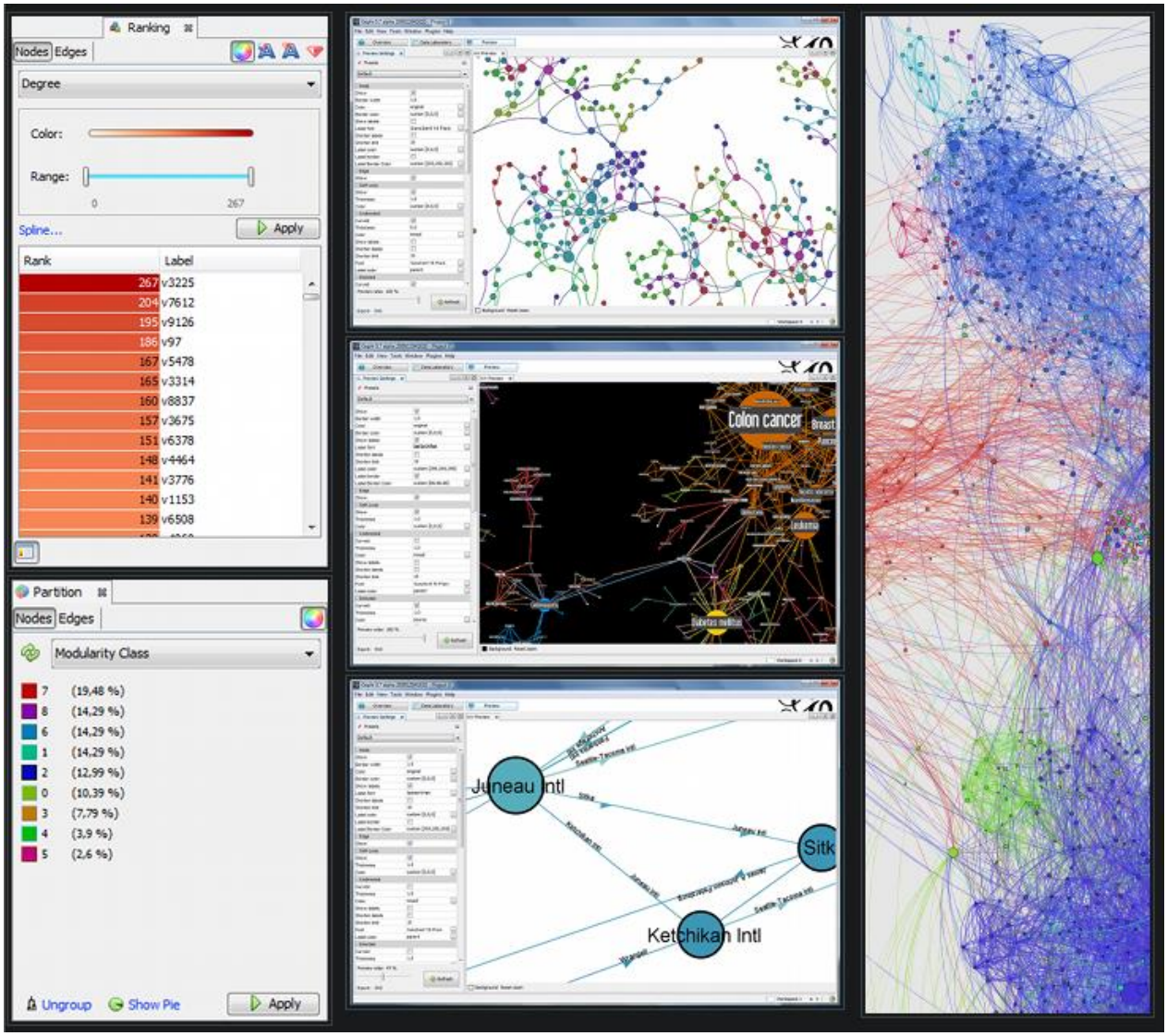
- избор на най различни облици (Layout) и много други;



Фигура 23: yEd избор на Layout. (източник: [6])

4.2 Gephi

Друг добър пример за използване на GraphML е в безплатния софтуерния пакет с отворен код за мрежов анализ и визуализация, написан на Java в платформата NetBeans, наречен Gephi (източник: [7]). Технологията предоставя ергономичен интерфейс (т.е. няма нужда от опит по програмиране), високоскоростен вграден двигател за рендериране, персонализация чрез плъгини и най-важното – поддържа файлови формати GraphML.



Фигура 24: Инструменти на Dephi. (източник: [7])

5 Добри практики и методи за използване

- Поставяне на *edge* елементите в най-високото ниво на йерархията на възлите, за които въвеждат отношения.
- Добавяне на основно съдържание след предефиниране на атрибутните групи, защото може да има повече от една Schema дефиниция, разширяваща същата група от атрибути.
- Приложения, използващи GraphML документа, които не могат да обработват хиперграфи, трябва да имат дефиниран заобиколен път, например:
    - игнориране на хиперребрата.
- Приложения, изпозващи GraphML документа, които не могат обработват вложени графи, трябва да имат дефиниран заобиколен път, например:
    - игнориране на възли, които не се съдържат в графа от най-високото ниво в йерархията;
    - игнориране на възли, които нямат две крайни точки в графа от най-високото ниво в йерархията.
- Приложения, използващи GraphML документа, които не могат да обработват графи с портове, трябва да имат дефиниран заобиколен път, например:
    - игнориране на портовете.

6 Заключение и очаквано бъдещо развитие

GraphML е XML-базиран файлов формат за графи. Той е резултат от сборният труд на общността по изобразяване на графи да се създаде всеобщ формат за обмяна да структурирана като граф информация. Използва синтаксис базиран на XML и поддържа целия обхват от възможни граф структури, включващи насочени, ненасочени, смесени графи, хиперграфи и атрибути, специфични за отделните приложения.

Развитието на семантичния Уеб предполага, че XML и базираните на него езици ще намират все по-широко приложение. GraphML форматът, благодарение на лесния си за разбиране и удобен за използване както от хора, така и от програми, език, ще намери дълбоко приложение в набиращите популярност услуги на големи системи и Уеб-базирани системи, използващи теорията на графите в своята логика.

7 Разпределение на работата

Работата по текущото есе е съвместния труд на всички участници в екипа. Всеки търси материали по темата и след това намереното се обсъжда взаимно, споделят се мнения и накрая се взима решение.

8 Използвани литературни източници

1. Graph Markup Language (GraphML)

2. GraphML Primer (graphdrawing.org)
3. GraphML Specification (graphdrawing.org)
4. GXL - Graph eXchange Language (uni-koblenz.de)
5. CASOS (cmu.edu)
6. yWorks
7. Gephi - The Open Graph Viz Platform