Towards NetML
Networks Markup Language

Vladimir Batagelj and Andrej Mrvar

International Social Network Conference
London, July 6th - 10th 1995

1. Introduction

NetML is a special 'language' for description of networks and their graphical representations.

A network is usually described by listing its vertices and its lines. To describe its picture we have to provide additional information (positions, shapes and labels of vertices and lines) which is used by presentation programs to display the picture of a given network.

A more compact description can be obtained by two approaches: *factorisation* – common parts of the descriptions are 'moved out' and *proceduralisation* – structure is built from (regular) substructures.

NetML implements all three forms of description. It provides a tool for a complete descriptions of graph and it can also serve as an intermediate description between different formats.

2. Levels of NetML

There are four *levels* of NetML:

- level 0 – complete description;
- level 1 – reduced (factorisation) complete description;
- level 2 – description using operations on structures;
- level 3 – description using user defined operations.

Programs implementing higher levels of NetML can produce lower level descriptions.

The level 0 and level 1 descriptions are primarly intended for presentation programs and graph editors.

At level 2 and level 3 we can include additional information about the structure of a given graph (chemical fragments, construction sequence for inductively defined graphs,...).

3. NetML and SGML

We decided to base the design of NetML on SGML and we got some inspiration from TEI.

SGML is an international standard for the definition of device and system independent methods of representing texts in electronic forms. The structure of the data is expressed by use of named *tags*. They have two forms: start-tag and end-tag pairs, which bracket off a data element – for example

```
<title> Graph drawing </title>
```

and 'stand alone' tags, without the corresponding end-tags. When they are implied by other tags the end-tags can be omitted.

Additional information about a data element can be given by specifying values of tag's *attributes*.

The tags and their attributes are defined in a special DTD (Document Type Definition) file.

NetML is still in the development. By its use we are trying to identify the concepts and problems in describing graphs and their pictures and to get the experience.

4. NetML descriptions

A standard 'envelope' of the description of graphs in NetML has the form

```
<!doctype netml SYSTEM "./netml.dtd">
<netml>
     descriptions of graphs
</netml>
```

Each graph is described inside an element marked by a tag

<graph id=*i* lab=*l* type=*s*>

The type *s* is a combination of: simple, directed, undirected, ordered.

NetML contains also a tag

<network id=*i* lab=*l* type=*s*>

for describing networks (graphs with values on lines and/or vertices). This tag will not be discussed in this paper.

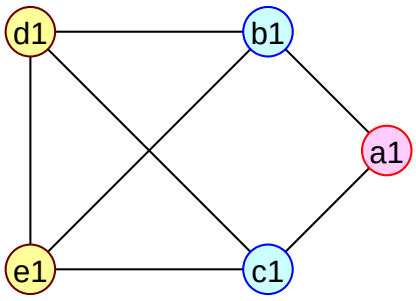
5. Description of Graphs – Basics

A graph consists of vertices and lines

<vertex id=i lab=l rank=n>
<line id=i from=$v1$ to=$v2$ type=t rank=n>

where $t \in \{edge, arc\}$. Sometimes there are vertices/lines of different types (graph and its dual, clusters, halflines, subdivision vertices,...). These situations can be expressed using ranks.

```
<graph id=london1>
 <vertex id=a1> <vertex id=b1>
 <vertex id=c1> <vertex id=d1>
 <vertex id=e1>
 <line id=ac1 from=a1 to=c1 type=edge>
 <line id=ab1 from=a1 to=b1 type=edge>
 <line id=bd1 from=b1 to=d1 type=edge>
 <line id=be1 from=b1 to=e1 type=edge>
 <line id=cd1 from=c1 to=d1 type=edge>
 <line id=ce1 from=c1 to=e1 type=edge>
 <line id=de1 from=d1 to=e1 type=edge>
</graph>
```



6. Description of Graphs – Rules

By introducing some conventions we can shorten such descriptions.

Rule 1. *Defaults*: Values of missing attributes are replaced by current default values.

Rule 2. *Defaults change*: The default value of an element is changed by a tag without the `id` attribute.

Rule 3. *Implicite vertices*: The application of a vertex v in some tag implicitely produces a tag `<vertex id=`v`>`.

Rule 4. *Implicite labels*: When a label attribute is omitted its value is filled by the corresponding `id` value.

Rule 5. *Standard names of vertices*: `v`i – i index of vertex.

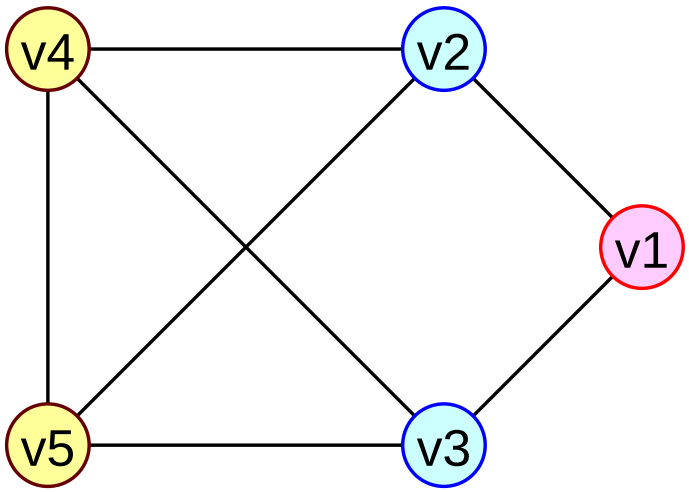
Rule 6. *Implicite names of lines*:

$a.b.i$ – i-th arc from a to b; $a.b = a.b.1$

a-$b.i$ – i-th edge between a and b;
a-$b.i = b$-$a.i$; a-$b = a$-$b.1$

7. Example – Rules

```
<graph id=london1a>
 <line type=edge> <line from=v1 to=v2>
 <line from=v1 to=v3> <line from=v2 to=v4>
 <line id=v2-v5> <line id=v3-v4>
 <line id=v3-v5> <line id=v4-v5>
</graph>
```



8. Description of Graphs – Stars

One of the most frequent ways to describe a graph structure is by the use of *stars*.

<star at=*v*> *lines* </star>
<order at=*v* start=*l*1 next=*l*2>

In some applications the order of lines in the stars is important (rotations in graph embeddings, order of sons in search trees,...). The order is determined implicitely by the construction or explicitely by the order tag. Order is represented by a circular list of lines.

```
<graph id=london1b type="simple,undirected,ordered">
 <star at=a1> <line to=c1> <line to=b1>
 <star at=b1> <line to=a1> <line to=e1> <line to=d1>
 <star at=c1> <line to=a1> <line to=d1> <line to=e1>
 <star at=d1> <line to=b1> <line to=e1> <line to=c1>
 <star at=e1> <line to=b1> <line to=c1> <line to=d1>
</graph>
```

9. Description of Graphs Transformations

Vertex transformations

<vcpy id=v od=u>	add to vertex v the star in u;
<vidt id=v od=u>	identify vertices v and u;
	add to v the star in u; delete u
<vren id=v od=u>	rename vertex u to v;
<vdel id=v rank=r>	delete vertex v;
	delete vertices of rank r;
<vdlp id=v rank=r>	delete loops in vertex v;
	delete loops in vertices of rank r.

Line transformations

<lren id=l od=k>	rename line k to l;
<lrev id=l>	reverse the direction of line l;
<ldel id=l rank=r>	delete line l;
	delete all lines with rank r;
<ldlp rank=r>	delete all loops with rank r;
<ldiv id=l vd=v	subdivide the line l producing
li=p lt=q>	a new vertex v and lines p and q;
<join id=l li=p	join two incident lines p and q
lt=q>	producing a new line l.

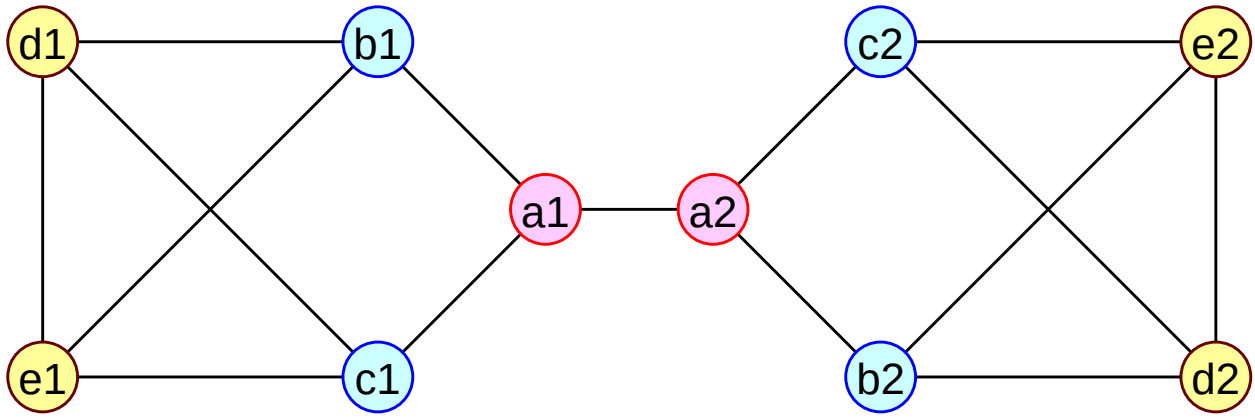
10. Combining Graphs

We can also describe a graph by combining some existing graph. This can be done by a `get` tag:

`<get id=`$g1$ `od=`$g2$ `ids=`s`>`

where $s \in \{old, new\}$. Vertex and lines ids are local to a graph. Each application of a `get` tag produces a new copy of a referenced graph. If `ids=old` the original ids are preserved; if `ids=new` all ids are automatically replaced by new ids. The user can also explicitely rename (some) ids inside the body of the `get` tag.

11. Example – Combining Graphs




```
<graph id=london2>
 <get od=london1b>
 <get od=london1b>
  <vren id=a2 od=a1> <vren id=b2 od=b1>
  <vren id=c2 od=c1> <vren id=d2 od=d1>
  <vren id=e2 od=e1>
 </get>
 <line id=a1-a2>
 <order at=a1 first=a1-a2 second=a1-c1>
 <order at=a2 first=a2-b2 second=a2-a1>
</graph>
```

12. Description of Pictures

Pictures are described inside the tag

`<picture id=`*i* `lab=`*l* `type=`*s* `dim=`*n* `space=`*i*1`>`

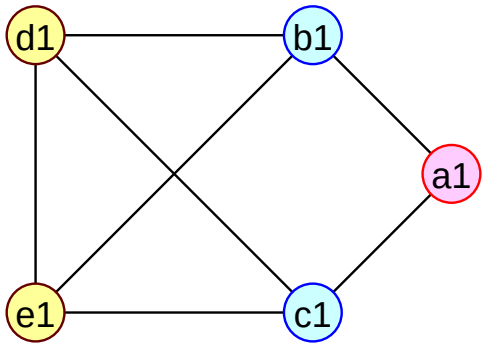
The order of drawing is implicitely determined by the description. An explicite setting of drawing order is possible by the use of priority attribute `prior`. Graphical elements (`shape, arrow, strip, label, draw, fill`) are put in the drawing list which is displayed when the description is completed.

Picture is linked to *points* in 2, 3 or higher dimensional *space*. Besides points the space can contain also *directions* (position + direction).

`<space id=`*i* `dim=`*n*`>` *points + dirs* `</space>`
`<point id=`*i*`>` *coords* `</point>`
`<dir id=`*i* `org=`*p*1 `vec=`*p*2`>` *coords coords* `</dir>`

The attributes set of `vertex` tag is extended by a `pos` attribute linking a given vertex to a point in the space.

13. Example – Picture



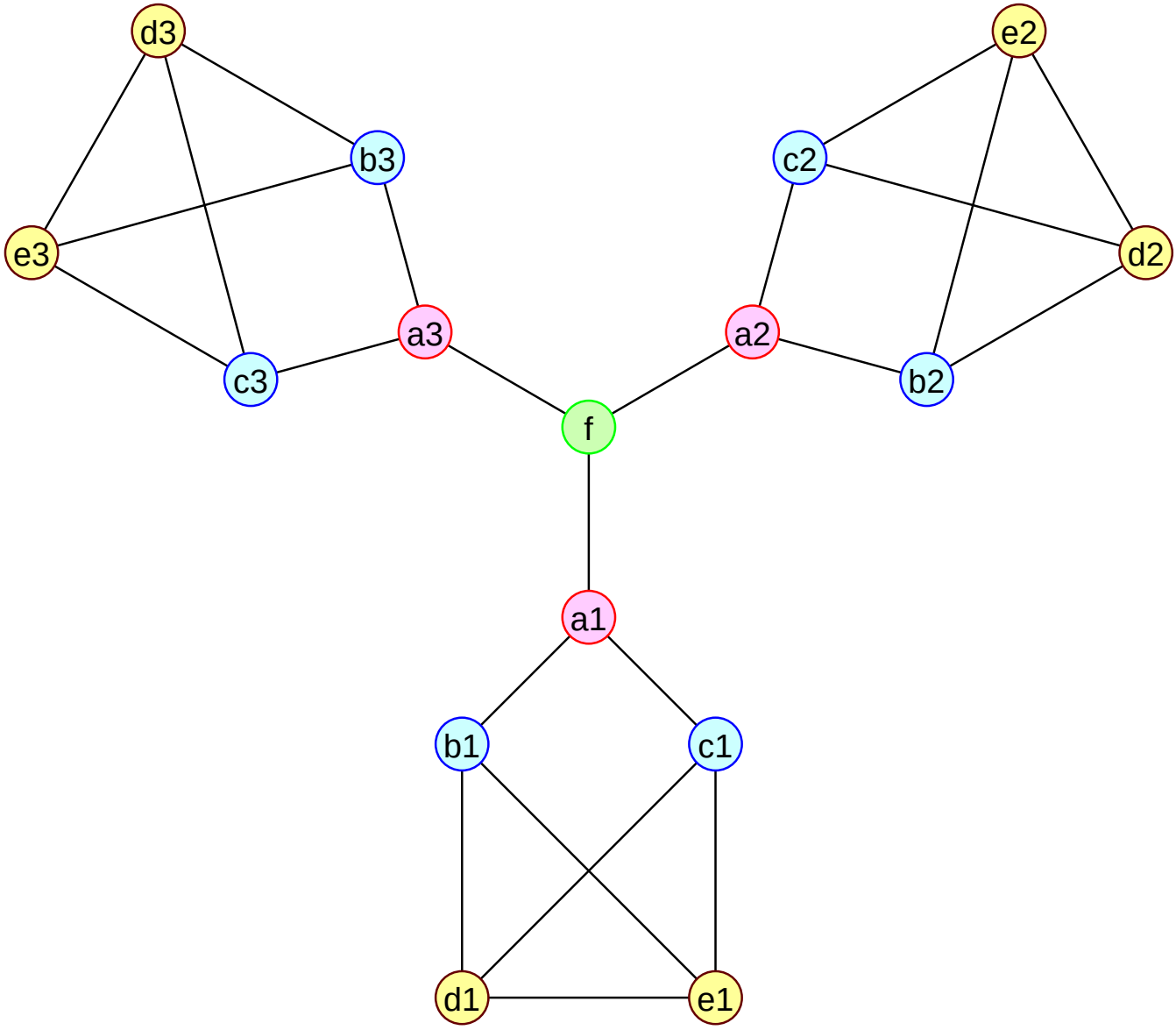

```
<picture id=londonp dim=2>
 <get od=london1b>
 <point id=pa> 400 200  <vertex id=a1 pos=pa>
 <point id=pb> 300 300  <vertex id=b1 pos=pb>
 <point id=pc> 300 100  <vertex id=c1 pos=pc>
 <point id=pd> 100 300  <vertex id=d1 pos=pd>
 <point id=pe> 100 100  <vertex id=e1 pos=pe>
</picture>
```

The tag

<trot id=p dirs=s dirt=f>

translates the picture p ($s.org \rightarrow f.org$) and rotates (and expands) the translated picture ($s.vec \rightarrow f.vec$).

14. Y Graph



15. Description of Y Graph

```
<picture id=londonp3>
 <get id=pic1 od=londonp>
  <point id=org0> 400 200 <point id=org1> 500 400
  <point id=toN>    0   1 <point id=toW> -1 0
  <dir id=N org=org0 vec=toN> <dir id=W org=org1 vec=toW>
  <trot id=pic1 dirs=N dirt=W>
 </get>
 <get id=pic2 od=pic1>
  <vren id=a2 od=a1> <vren id=b2 od=b1> <vren id=c2 od=c1>
  <vren id=d2 od=d1> <vren id=e2 od=e1>
  <point id=center> 500 550
  <point id=toE>    1   0  <point id=d120> -0.5 0.866
  <dir id=cE org=center vec=toE>
  <dir id=cd120 org=center vec=d120>
  <trot id=pic2 dirs=cE dirt=cd120>
 </get>
 <get id=pic3 od=pic2>
  <vren id=a3 od=a2> <vren id=b3 od=b2> <vren id=c3 od=c2>
  <vren id=d3 od=d2> <vren id=e3 od=e2>
  <trot id=pic3 dirs=cE dirt=cd120>
 </get>
 <vertex id=f pos=center>
 <line id=f-a1> <line id=f-a2> <line id=f-a3>
</picture>
```

16. Description of Pictures of Graphs

There are other tags. The position and properties of labels are determined by tags

<style id=*i* dir=*d* font=*f* fsize=*n* fcolor=*c*>
<label id=*i* pos=*d* txt=*t* sty=*s*>

The definitions of basic graphic elements (shapes, arrows, curves,...) are also supported by the corresponding tags and are usually prepared on special resource files (flowchart elements, electrical circuit elements,...).

We assign a curve to a line by the use of the additional ci attribute <line id=*i* ci=*c*> To allow precise position of the endpoints of a curve we provide shapes with special directions – *hooks*.

Curves are additional objects in the space. They can be composed of several *segments*. They are drawn when they are referenced by a line tag or by tags <draw> and <fill>

17. Standard Data Structure