DyNetML: Interchange Format for Rich Social Network Data

Maksim Tsvetovat Jeff Reminga Kathleen M. Carley

February 2004
CMU-ISRI-04-105

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We define a universal data interchange format to enable exchange of rich social network data and improve compatibility of analysis and visualization tools. DyNetML is an XML-derived language that provides means to express rich social network data. DyNetML also provides an extensible facility for linking anthropological, process description and other data with social networks. DyNetML has been implemented and in use by the CASOS group at Carnegie Mellon University as a data interchange format. We have also implemented parsing and conversion software for interoperability with other software packages

This work was supported in part by the DOD, the National Science Foundation under the IGERT program for training and research in CASOS, NSF ITR 1040059 and the Office of Naval Research N00014-02-1-0973. Additional support was provided by CASOS - the center for Computational Analysis of Social and Organizational Systems at Carnegie Mellon University. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the DOD, the Office of Naval Research, the National Science Foundation, or the U.S. government.

Keywords: Social Networks, Data Interchange, Rich Data, XML, DTD

Contents

1 Introduction

Current state of the art in social network data representation presents a fairly bleak picture. Each of analysis and simulation packages uses its own, proprietary and incompatible data format. Some of the file formats do not even have a specification document, making the files unreadable without the software that produced them.

Data formats that were designed for interoperability (such as DL) are rarely expressive enough to fully represent the datasets.

As a result, most researchers are forced to deal with data interchange in a makeshift fashion, at best increasing the workload and at worst resulting in loss of data integrity.

To improve cooperation between researchers and to promote interoperability of software, the community needs to agree on a common data interchange language. In an informal meeting at CASOS 2002, a number of prominent developers and users of social network analysis tools have agreed on cooperating in the development of such an interchange language and to support it when it is available.

This paper presents a proposal for an XML-derived language that addresses requirements for expressivity and compatibility.

1.1 Pitfalls of the Existing Tools

As we mentioned above, the current social network data formats have a number of deficiencies:

Binary files are very difficult to read if exact specification of the file format is not provided. Significant extra efforts are required to keep compatibility with other tools or between versions of the same tool.

Multiple files used for specification of rich data or saving analysis output present a number of problems. First of all, there is a significant potential for data loss due to misplaced or corrupted files (for example, while sent through email). Secondly, a consistent naming scheme for all files and a file catalogue are required to prevent data loss - which requires a certain amount of discipline on the part of the researcher (as these features are not included in the analysis software)

Raw Data file such as binary matrices or edge lists lack the expressiveness required to represent multiple relations between nodes or evolution of social networks over time.

Human-Readable Data in text files or spreadsheets solves the expressivity problem but requires extensive post-processing by hand or with post-processing scripts. However, these programs often represent the weakest link in the software chain (due to hasty design and dependance on outside tools such as Perl or Awk).

1.2 Existing Data formats

The DL format supported by UCINET is a flexible and human readable data format, and it can contain multiple matrices in a single file. However, the matrices in a DL file must be of a single type. For example, a single DL file cannot contain one matrix containing Agent by Agent data, and another with Agent by Knowledge data. Thus to represent an entire Meta-Matrix, multiple DL files must be used, which increases the likelihood of data inconsistencies.

2 Requirements for Data Interchange

In light of the problems outlined above, we proceed to define requirements for a universal data interchange format that would make much easier the task of exchanging rich social network data and improving compatibility of analysis and visualization tools.

1. The data interchange format shall be contained in human-readable text files that are in the same time easily parsable by computers.
2. The data interchange format shall allow an entire dataset, complete with all computed measurements, to be stored in one file
3. The data interchange format shall provide maximum expressive power to its users, allowing:
   - Typed nodes (types may include "person", "resource", "organization", "knowledge", etc)
   - Multiple sets of nodes of the same type (to express multiple units within the company, etc)
   - Multiple typed attributes per node
   - Typed edges
   - Multiple typed attributes per edge
   - Multiple graphs (sets of edges) expressed within the same file
   - Dynamic network data expressed in a single file
4. The data interchange format shall allow developers to extend it in a fashion that will not break existing software
5. The data interchange format shall be flexible enough to be used as both input and output of analysis tools.

3 DyNetML: an XML-Derived Social Network Language

To address the needs of data interchange, we have designed DyNetML: an XML-derived language for expressing rich social network data.

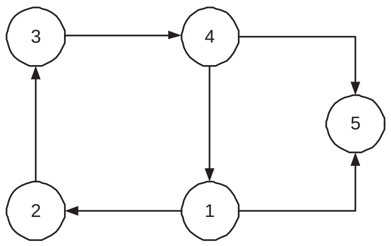


Figure 1: Dynamic Networks in DyNetML

The following simple example illustrates use of DyNetML for representing simple social network datasets (also illustrated on figure 1):

```
<?xml version = "1.0" encoding = "UTF-8"?>

 <DynamicNetwork xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation = "DyNetML.xsd">
    <MetaMatrix>
        <nodes>
            <nodeset id = "actors" type = "agent">
                <node id = "actor1"/>
                <node id = "actor2"/>
                <node id = "actor3"/>
                <node id = "actor4"/>
                <node id = "actor4"/>
                <node id = "actor5"/>
            </nodeset>
        </nodes>
        <networks>
            <graph id = "social_network" sourceType = "agent"
                   targetType = "agent" isDirected = "true">
                <edge source = "actor1" target = "actor2" type = "binary"/>
                <edge source = "actor2" target = "actor3" type = "binary"/>
                <edge source = "actor3" target = "actor4" type = "binary"/>
                <edge source = "actor4" target = "actor1" type = "binary"/>
                <edge source = "actor1" target = "actor5" type = "binary"/>
                <edge source = "actor4" target = "actor5" type = "binary"/>
            </graph>
        </networks>
    </MetaMatrix>
</DynamicNetwork>
```

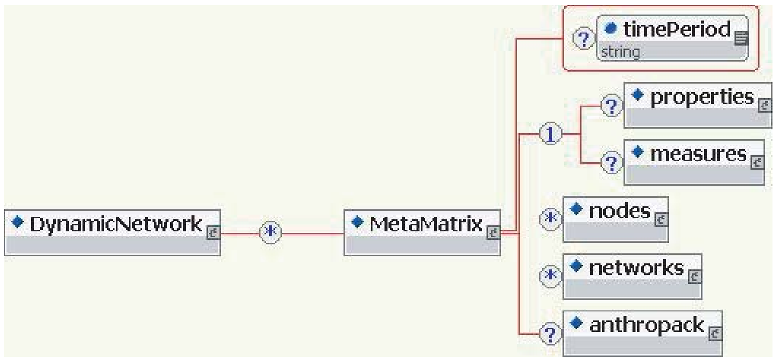


Figure 2: Dynamic Networks in DyNetML

3.1 DyNetML Format Overview

DyNetML represents dynamic network data as sets of time-slices. Each of the time-slices is a descriptive snapshot of the organization at a given time.

Figure 2 shows the top-level hierarchy of DyNetML files. A **Dynamic-Network** element is defined as a sequence of **MetaMatrix** elements, each represening a snapshot of the organization for one time period.

Each of the **MetaMatrix** elements consists of:

- an optional **TimePeriod** attribute that allows clear identification of each timeslice
- a set of **properties** and **measures**, representing data about the whole of the timeslice (see section 6 for a complete definition)
- a **nodes** element, containing one or more nodesets (section 4.1)
- a **networks** element, containing all networks in this timeslice (section 5)
- an **anthropac** element that facilitates linking of network data to anthropological data.

4 Representing Multiple Node and Relation Types

While designed predominantly for dealing with social network data, DyNetML format is shaped as a generalized graph data interchange framework.

DyNetML represents graph as sets of nodes nodes (vertices) and relationships (edges) between them. The node specification allows for detailed specification of each vertex, as well as addition of rich data related to it.

4.1 Specifying Individuals and Nodes

Nodes are organized into **nodesets**, which should be though about as logical groupings of nodes (by type, by affiliation, etc).

Each nodeset has to be identified with a unique **id** attribute (see figures 3 and 4), and a **type** attribute. For more detailed description of node types, see section 4.1.2.

DyNetML allows for an arbitrary number of nodesets (and arbitrary number of nodesets of each type) and an arbitrary number of nodes in each nodeset.

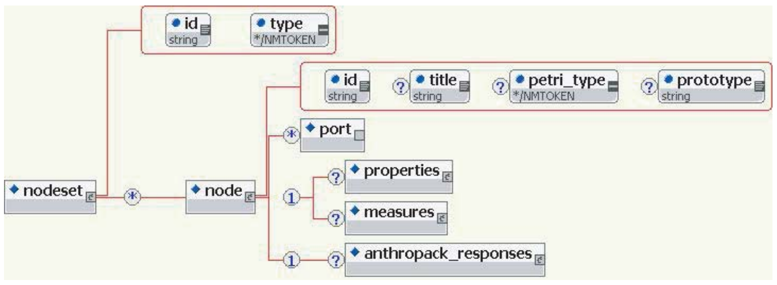


Figure 3: Specification of Vertices in DyNetML

```
<node id="test"
      title="test node"
      prototype="test nodes">
    <port name="in1" port_type="input"/>
    <port name="out1" port_type="output"/>
    <properties>
        <property name="test_property" type="double" value="3.14"/>
    </properties>
    <measures>
        <measure name="test_measure" type="string"
        value="string"/>
    </measures>
</node>
```

Figure 4: Sample specification of a Vertex

A node specification consists of the following (see figure 3):

- **id**: a unique ID *note: it is advisable for ease of searching to use node IDs that do not contain spaces or special characters and are limited in length to 32 characters.*
- **title**: a human-readable title of the vertex (free of restrictions posed on node ID field).

- **prototype**: an optional attribute specifying a subclass of a node. Node prototype can be used to specify additional details about the node, and is essential to the implementation of hypergraphs (section 8).
- Element **port** allows the user to specify inflows and outflows of each node by allowing multiple connection points within each node
- **Properties** and **Measures** elements allow specification of arbitrary rich data for each node
- **Anthropac** element provides a vehicle for connecting anthropological data with social network data (see section 9 for complete specification of anthropological data)

4.1.1 Ports and Multiple Connection Points

In order to implement multiple types of connections within the same graph, and to enable use of graphs as nodes of other graphs, we have implemented a system of ports.

A port can be viewed as a point where an edge attaches to a vertex. Thus, a directed edge connecting a port specified as input to another node's port specified as output represents a resource or information flow across the edge.

Since one can specify multiple input and output ports for every node, it is possible to represent a number of distinct flows along every edge while maintaining clear separation between different types of links.

A port is defined as follows (see figure 3:

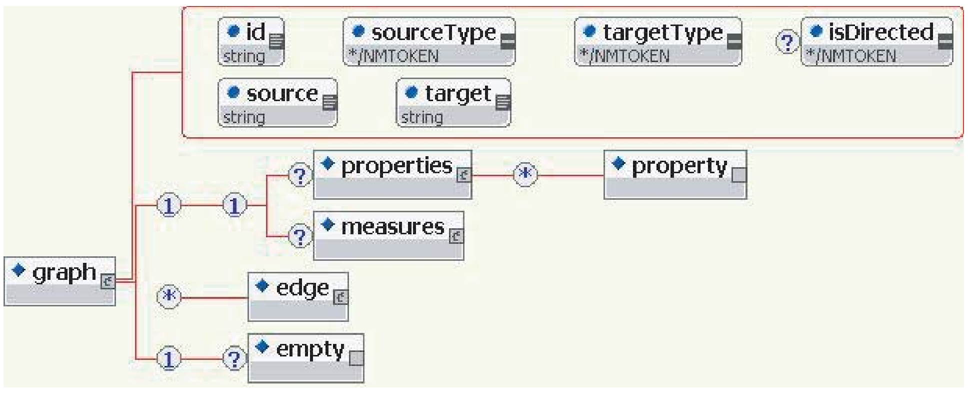
- attribute **name** specifies a port ID that is unique for this node
- attribute **port_type** is a multiple choice, with possible values ”input”, ”output” and ”general”

4.1.2 Node Types in DyNetML

DyNetML has been designed to assist the flow of information between software tools by not only enforcing a consistent structured format upon the data, but also by specifying a constant vocabulary. Since the language has been designed in service of the Social Network Analysis community, we specify a set of standard node types that could be used to express majority of rich social network data.

The standard node types are: **agent, organization, knowledge, resource, task, location**, and **graph**.

While the plugin architecture of DyNetML allows developers to easily add node types, we suggest that to ensure inter-operability of tools using DyNetML one should refrain from expanding the vocabulary unless absolutely required. To provide a more fine-grained node type mechanism, we suggest using the **prototype** attribute of nodes to specify arbitrary subtypes.



```
<graph id="TestGraph"
    source="people"
    sourceType="agent"
    target="resources1"
    targetType="resource"
    isDirected="true">

    <properties> ... </properties>
    <measures> ... </measures>

    <edge ...... />
    .......
</graph>
```

Figure 5: Specification of Graphs in DyNetML

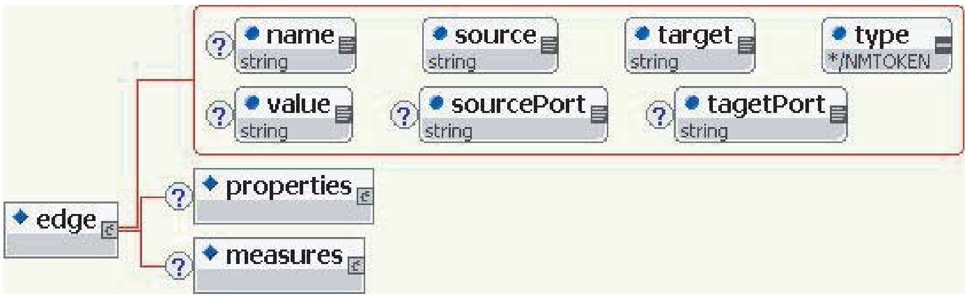
5 Representing Relations in DyNetML

DyNetML format allows the user to specify multiple graphs within a single framework, including graphs that share vertices with other graphs.

An example for use of such system is the case where a number of individuals are engaged in multiple relationship types - such as the formal network, informal advice network, or familial ties network.

Each graph is specified as follows: (see figure 5)

- **id** attribute is the graph's unique ID
- **source** attribute specifies the nodeset from which the source nodes are taken
- **sourceType** attribute specifies the type of nodes contained in the source nodeset



```
<edge source="test1"
      sourcePort="out1"
      target="test2"
      targetPort="in1"
      type="double"
      value="3.14"
      name="testEdge">

      <properties>...</properties>
      <measures>...</measures>
</edge>
```

Figure 6: Specification of Graphs in DyNetML

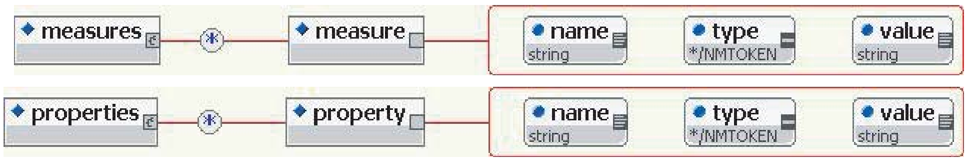
- **target** attribute specifies the nodeset from which the target nodes are taken
- **targetType** attribute specifies the type of nodes contained in the target nodeset
- **isDirected** attribute specifies whether the edges of this graph are directed; the attribute can only take values of ”true” or ”false”

The graph then includes **properties** and **edges** elements (see 6), followed by a set of **edge** elements that comprise the actual graph.

5.1 Edges

Edges (see figure 6) of the graph include the following attributes:

- **source** and **sourcePort** attributes specify the source node and port that an edge originates from. The source node should be a part of the nodeset specified in the **source** attribute of the graph; **source** attribute is required, **sourcePort** is optional if no ports have been defined for the source node.



```
<properties>
 <property name="test" type="double" value="3.14"/>
 <property name="test2" type="string" value="test"/>
</properties>

<measures>
 <measure name="test" type="double" value="3.14"/>
 <measure name="test2" type="string" value="test"/>
</measure>
```

Figure 7: Specification of Properties and Measures

- **target** and **targetPort** attributes specify the source node and port that an edge connects to. The target node should be a part of the nodeset specified in the **target** attribute of the graph; **target** attribute is required, **targetPort** is optional if no ports have been defined for the target node.
- **type** attribute is required for every edge. If the edge is unweighted, the type attribute should be set to "binary"; other acceptable edge value types are "double" and "string"
- **value** attribute specifies the edge weight or value; the type of the value should match the type specified in **type** attribute.
- **name** attribute is an optional string that allows the user to add human-readable title to an edge.

6 Representing Graph, Node and Edge Attributes

One of the important facilities of DyNetML is its ability to attach rich data, or attributes to every element of the structure.

The rich data, specified as **properties** and **measures** can be added to the **MetaMatrix**, **node**, **graph** and **edge** objects.

Properties and **Measures** objects are syntactically similar (see figure 7 and consist of a set of name-value pairs. The main distinction between them is that **Properties** should be thought of as attributes inherent to the subject, such as information obtained from a questionnaire or otherwise known about the subjects.

Measures, on the other hand, are computed by analysis tools and inserted into the dataset during processing.

The guidelines for naming properties and measures are following:

- Names should be descriptive of the nature of data contained within
- Measure names should include the name of the tool that generated them

For example, the measure of Freeman centrality computed by NetStat tool should look as:

```
<measure name="netstat_freeman_centrality" type="double"
value="3.14"/>
```

7 Complex Social Networks in DyNetML

The basic use of DyNetML is specification of rich social network data, including properties and measures attached to objects within the network. DyNetML also allows for an arbitrary number of network superimposed upon each other, and specification of network data over time.

The example below is a heavily commented small dataset containing two types of nodes: people and facts, and three networks - friendship, advice and knowledge.

7.1 Example

```
 <?xml version = "1.0" encoding = "UTF-8"?>
 <!DOCTYPE DynamicNetwork SYSTEM "DyNetML.dtd">

 <DynamicNetwork>
 <!-- Define the metamatrix... watch comments in
the XML for explanation of particular features of the format -->

 <MetaMatrix timePeriod = "1">
 <!-- A global measure on the entire metamatrix -->

  <measures>
    <measure name = "global" type = "double" value = "3.14"/>
  </measures>

  <!-- First, we specify the nodes -->

        <nodes>
            <!-- Nodes are broken up into nodesets by type (e.g. agent,
                 knowledge, resource, task, etc) -->

            <nodeset id = "people" type = "agent">

                <!-- This is the simple node with no extended data -->
                <node id = "b"/>

                <!-- This is a more complex node with properties and
                     attached measures -->
                <node id = "a">
                    <!-- This is how to specify internal node properties -->
                    <properties>
                        <property name = "foo" type = "double" value = "3.14"/>
                        <property name = "bar" type = "double" value = "3.14"/>
                    </properties>

                    <!-- This is how to specify node-level measures -->
                    <measures>
                        <!-- Each measure is named and accompanied by
                             type (double|string|binary) -->
                        <measure name = "centrality" type = "double"
                                 value = "3.14"/>
                        <measure name = "betweenness" type = "double"
                                 value = "3.14"/>
                    </measures>
                </node>
            </nodeset>
            <!-- Another nodeset -->

            <nodeset id = "facts" type = "knowledge">
                <node id = "a1"/>
                <node id = "a2" title = "boss"/>
            </nodeset>
        </nodes>
        <!-- Now we specify the graphs that comprise the metamatrix -->

        <networks>

<!--
    NOTE: source and target of each edge should be a valid node;
    however it's up to the software developer to ensure that - or
    to check consistency in any code that imports this data
-->

            <!-- A very simple graph -->
            <graph id = "friendship" source="people" sourceType = "agent"
```

```
                    target="people" targetType = "agent">
                <edge source = "a" target = "b" type = "binary"/>
                <edge source = "b" target = "a" type = "binary"/>
            </graph>

            <!-- A graph with some graph-level measures -->
            <graph id = "advice" source="people" sourceType = "agent"
                    target="people" targetType = "agent">
                <measures>
                    <!-- Just like node-level measures; nothing new here -->
                    <measure name = "degree" type = "double" value = "3.14159"/>
                    <measure name = "foo" type = "double" value = "3.14159"/>
                    <measure name = "bar" type = "double" value = "3.14159"/>
                </measures>
                <edge source = "a" target = "b" type = "binary"/>
            </graph>
            <graph id = "knowledgeNetwork" isDirected = "true" source="people"
                    sourceType = "agent" source="facts" targetType = "knowledge">
                <edge source = "a" target = "1" type = "string" value = "foobar"/>
                <edge source = "b" target = "2" type = "double" value = "3.14159"/>
            </graph>
        </networks>
    </MetaMatrix>
</DynamicNetwork>
```

8 Hypergraphs for Representation of Complex Social Networks

DyNetML allows users to specify complex multi-level networks by hierarchically encapsulating networks as nodes of other networks. Example on figure 8 illustrates a network that contains three organizations. Each of the organizations, in turn, contains several interconnected nodes.

For the purpose of this example, let us assume that the organizations are identical in structure. Thus, it is only necessary to define the structure once.

Each of the organizations is then defined as an instance of the structure graph, through the use of **prototype** attribute.

The organizations are then connected into a network by a different graph, connecting the organizations' input and output ports.

```
<?xml version = "1.0" encoding = "UTF-8"?>
 <DynamicNetwork xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation = "DyNetML.xsd">
    <MetaMatrix>
        <nodes>
```

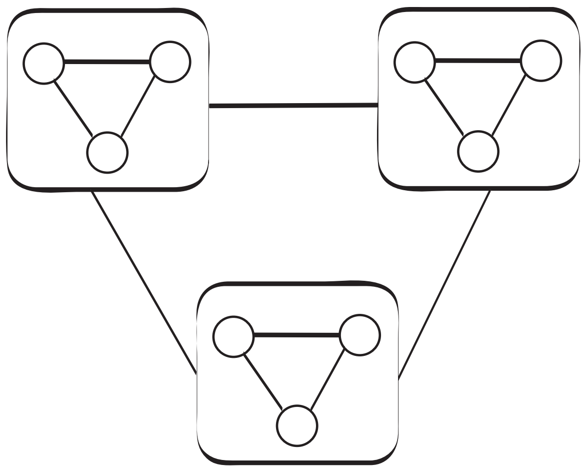


Figure 8: Using graphs as nodes of other graphs

```
        <nodeset id = "units" type = "agent">
            <node id = "a"/>
            <node id = "b"/>
            <node id = "c"/>
        </nodeset>
        <nodeset id = "instance" type = "graph">
            <node id = "triangle1" prototype = "triangle">
                <port name = "in" port_type = "input"/>
                <port name = "out" port_type = "output"/>
            </node>
            <node id = "triangle2" prototype = "triangle">
                <port name = "in" port_type = "input"/>
                <port name = "out" port_type = "output"/>
            </node>
            <node id = "triangle3" prototype = "triangle">
                <port name = "in" port_type = "input"/>
                <port name = "out" port_type = "output"/>
            </node>
        </nodeset>
    </nodes>
    <networks>
        <graph id = "triangle" sourceType = "petri" targetType =
               "petri" isDirected = "true">
            <edge source = "a" target = "b" type = "binary"/>
            <edge source = "b" target = "c" type = "binary"/>
            <edge source = "c" target = "a" type = "binary"/>
        </graph>
```

```
            <graph id = "hypertriangle" sourceType = "graph"
                   targetType = "graph" isDirected = "true">
               <edge source = "triangle1" target = "triangle2"
                     type = "binary" sourcePort = "out" tagetPort = "in"/>
               <edge source = "triangle2" target = "triangle3"
                     type = "binary" sourcePort = "out" tagetPort = "in"/>
               <edge source = "triangle3" target = "triangle1"
                     type = "binary" sourcePort = "out" tagetPort = "in"/>
               <edge source = "triangle2" target = "triangle1"
                     type = "binary" sourcePort = "out" tagetPort = "in"/>
               <edge source = "triangle3" target = "triangle2"
                     type = "binary" sourcePort = "out" tagetPort = "in"/>
               <edge source = "triangle1" target = "triangle3"
                     type = "binary" sourcePort = "out" tagetPort = "in"/>
            </graph>
        </networks>
    </MetaMatrix>
</DynamicNetwork>
```

9 Linking Anthropological Data with Social Network Data

ANTHROPAC is a program for collecting and analyzing data on cultural domains. The program helps collect and analyze structured qualitative and quantitative data including freelists, pilesorts, triads, paired comparisons, and ratings. ANTHROPAC's analytical tools include techniques that are unique to Anthropology, such as consensus analysis, as well as standard multivariate tools such as multiple regression, factor analysis, cluster analysis, multidimensional scaling and correspondence analysis. In addition, the program provides a wide variety of data manipulation and transformation tools, plus a full-featured matrix algebra language.

In order to facilitate simultaneous collection of anthropological and social network data, we introduce a representation of Anthropac data as a part of DyNetML, allowing anthropological data can be attached to any of the nodes in the network.

9.1 Specification

Anthropac data consists of a definition of a set of questions that comprise a survey, and a set of responses to these questions attached to nodes.

Thus, Anthropac data is divided into two parts: question definition and responses.

Questions are specified in an optional **anthropac** section within the definition of a metamatrix:

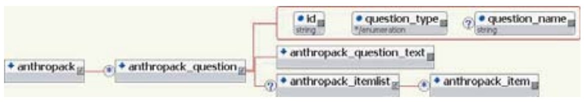


Figure 9: Anthropac Question Format

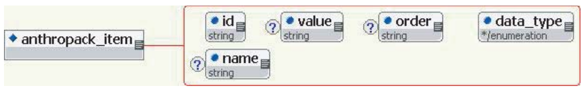


Figure 10: Anthropac Response Option (anthropac_item) Format

```
.....
 <MetaMatrix>
    ....
    <anthropac>
        <anthropac_question .../>
        ....
    </anthropac>
 </MetaMatrix>
```

9.1.1 Anthropac Question Format

Anthropac questions (see figure 9) are specified as following:

- **id** attribute: a unique question ID (can be numeric or string)
- **question_name** attribute: a human-readable question name
- **question_type** attribute: a string specifying the type of question in this element. Allowed question types are: **bio, frame, freelist, multiple_choice, pair, triad, pile** and **text**
- a text element **anthropac_question_text** containing the text of the question
- an **anthropac_item_list** element containing a set of response options contained in **anthropac_item** elements (not necessary for specification of non-multiple-choice questions).

anthropac_item elements (see figure 10) contain:

- a required **id** attribute: an answer option ID that is unique for this question (can be string or numeric)
- a required **data_type** attribute, specifying the type of value (**double, string or boolean**)
- an optional **name** attribute, containing a human readable description of the answer option

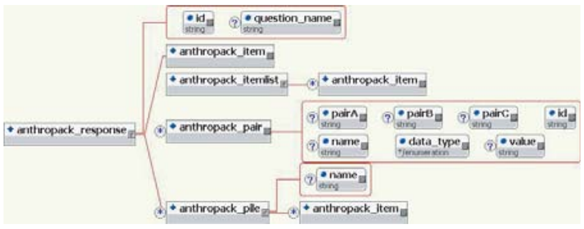


Figure 11: Anthropac Response Format

- an optional **value** attribute: contains the respondent's answer to the question or a choice option for question specification
- an optional **order** attribute containing the sequence number of this response option within the question

9.1.2 Responses

Responses to the questions in an Anthropac questionnaire are specified as an optional part of a **node** element:

```
....
<nodeset id="test" type="agent">
   <node id="1" ..../>
       ....
       <anthropac_responses>
           <antropac_response .../>
           ....
       </anthropac_reponses>
   </node>
</nodeset>
```

Anthropac question response (see figure 11 are defined as follows:

- required **id** and **question_name** attributes link an anthropac_response item to the question that it's answering
- a single **anthropac_item** element (see above) is used to specify a simple answer to the question
- a **anthropac_itemlist** element contains a set of **anthropac_item** elements for more complex questions such as freelists
- optional **anthropac_pair** and **anthropac_pile** items pertain to a pair, triad and pilesort question types which will be discussed below.

9.1.3 Biographical Questions

Biographical questions ask a for a number of simple items such as name, age or gender of the respondent.

Question

```
<anthropack_question id = "1" question_type = "bio"
                     question_name="Biographical Data Question">
    <anthropack_question_text>
        This is the question text for a bio question
    </anthropack_question_text>
    <anthropack_itemlist>
        <anthropack_item id = "gender" data_type = "string"/>
        <anthropack_item id = "age" data_type = "double"/>
    </anthropack_itemlist>
</anthropack_question>
```

Answer Format

```
<anthropack_response id = "1"
    question_name = "Biographical Data Question">
    <anthropack_itemlist>
        <anthropack_item id = "1" data_type = "string" value = "Male"/>
        <anthropack_item id = "2" data_type = "double" value = "29"/>
    </anthropack_itemlist>
</anthropack_response>
```

9.1.4 Frame Substitution Question

Question Format

```
<anthropack_question id = "2" question_type = "frame">
    <anthropack_question_text>
        Frame Substitution Quesiton Text
    </anthropack_question_text>
    <anthropack_itemlist>
        <anthropack_item id = "1" value = "1" data_type = "string"/>
        <anthropack_item id = "1" value = "2" data_type = "string"/>
    </anthropack_itemlist>
</anthropack_question>
```

Answer Format

```
<anthropack_response id = "2"
    question_name = "Frame Substitution Question">
    <anthropack_itemlist>
    <anthropack_item id = "1" data_type = "string" value = "1" name = "foo"/>
    <anthropack_item id = "6" data_type = "string" value = "2" name = "bar"/>
    </anthropack_itemlist>
</anthropack_response>
```

9.1.5 Free-list Question

Free-lists allow respondents to answer a question with a list of items.

Question

```
<anthropack_question id = "3" question_type = "freelist">
    <anthropack_question_text>
        This is a freelist question
    </anthropack_question_text>
</anthropack_question>
```

Answer Format

```
<anthropack_response id = "3"
    question_name = "This is a freelist question">
    <anthropack_itemlist>
        <anthropack_item id = "1" data_type = "string" order = "1" value = "red"/>
        <anthropack_item id = "2" data_type = "string" order = "2" value = "blue"/>
        <anthropack_item id = "3" data_type = "string" order = "3" value = "green"/>
    </anthropack_itemlist>
</anthropack_response>
```

9.1.6 Text Question

Text questions allow respondents to answer the question in form of an essay or a short answer.

Question

```
<anthropack_question id = "4" question_type = "text">
    <anthropack_question_text>
        Freeform/essay question text
    </anthropack_question_text>
</anthropack_question>
```

Answer Format

```
<anthropack_response id = "4">
    <anthropack_item id = "1" data_type = "void">
        long-winded open-ended answer
    </anthropack_item>
```

```
</anthropack_response>
\subsubsection{Multiple Choice Question}

\paragraph{Question}

\begin{verbatim}
<anthropack_question id = "5" question_type = "multiple_choice">
    <anthropack_question_text>
        Select box and multiple-choice are very similar
    </anthropack_question_text>
    <anthropack_itemlist>
        <anthropack_item id = "1" data_type = "void">Choice 1</anthropack_item>
        <anthropack_item id = "2" data_type = "void">Choice 2</anthropack_item>
    </anthropack_itemlist>
</anthropack_question>
```

Answer Format

```
<anthropack_response id = "5">
    <anthropack_item id = "1" data_type = "void"/>
</anthropack_response>
```

9.1.7 Pair and Triad Comparison Question

Pair Comparison questions allow the respondent to pick items from the list and sort them into matching pairs

Question Format

```
<anthropack_question id = "6" question_type = "pair">
    <anthropack_question_text>
        This is a pair comparison question
    </anthropack_question_text>
    <anthropack_itemlist>
        <anthropack_item id = "1" data_type = "void">Item Name 1</anthropack_item>
        <anthropack_item id = "2" data_type = "void">Item Name 2</anthropack_item>
        <anthropack_item id = "3" data_type = "void">Item Name 3</anthropack_item>
    </anthropack_itemlist>
</anthropack_question>
```

Answer Format

```
<anthropack_response id = "6">
    <anthropack_pair id = "1"
        pairA = "1" pairB = "2" name = "Pair 1"
        data_type = "double" value = "2"/>
    <anthropack_pair id = "2"
```

```
        pairA = "2" pairB = "3" name = "Pair 2"
        data_type = "double" value = "3"/>
</anthropack_response>
```

9.1.8 Pilesort Question

Pile Sort questions allow respondents to sort items into groups by different characteristics.

Question

```
<anthropack_question id = "7" question_type = "pile">
    <anthropack_question_text>
        This is a pilesort question
    </anthropack_question_text>
    <anthropack_itemlist>
        <anthropack_item id = "1" data_type = "void">Item Name 1</anthropack_item>
        <anthropack_item id = "2" data_type = "void">Item Name 2</anthropack_item>
        <anthropack_item id = "3" data_type = "void">Item Name 3</anthropack_item>
    </anthropack_itemlist>
</anthropack_question>
```

Answer Format

```
<anthropack_response id = "7">
    <anthropack_pile name = "Foo">
        <anthropack_item id = "3" data_type = "void">Item Name 3</anthropack_item>
    </anthropack_pile>
    <anthropack_pile name = "bar">
        <anthropack_item id = "1" data_type = "void">Item Name 1</anthropack_item>
        <anthropack_item id = "2" data_type = "void">Item Name 2</anthropack_item>
    </anthropack_pile>
</anthropack_response>
```

10 Parsing DyNetML

We have implemented parsers for DyNetML in C++ and Java, integrating them throughout the social network analysis and simulation tools developed at Carnegie Mellon University, including ORA, Construct, DyNet, OrgAhead and NetWatch. We have also developed visualization and editor tools that read and generate DyNetML, as well allow for conversion between DyNetML and other data formats such as UCINET DL, Pajec and raw text files.

DyNetML files can be parsed with any XML parser, such as Xerxes (www.apache.org) or MSXML (available from Microsoft).

For new implementations, we recommend that large DyNetML files be parsed by a SAX as opposed to a DOM parser - which is much more efficient when dealing with large nested hierarchies.

11 Availability

DyNetML definition as a DTD or XSD Schema, as well as examples of DyNetML files and some datasets in DyNetML format can be downloaded from:

`http://www.casos.cs.cmu.edu/dynetml`

We also make available converters from and to UCINET DL format. Other converters are also currently under development.

12 Future Directions

DyNetML allows researchers to freely combine anthropological data, social network data and process descriptions, in a robust and readable XML-derived format. DyNetML combines human readability of text files with standardized and rigorous parsing mechanism - thus allowing consistent interpretation of data by different software tools. While DyNetML files are verbose, they compress very well, and in compressed form take barely more space then matrix or other plain-text files with equivalent amount of data.

We are currently working to extend DyNetML to include representation of Petri Nets for process description and Bayesian Networks. Under implementation also is a more compact format that retains expressive power of DyNetML but allows easier specification of large networks.