The Untyped Network Hypothesis [DRAFT]

Mário Amado Alves, Alípio Jorge, José Paulo Leal
maa@liacc.up.pt

The UNH, semantic models, XML databases, and Mneson

The Untyped Network Hypothesis is the representation of complex information, or data, in an untyped network, or graph. Data are represented either directly in the untyped links or indirectly in certain topologies, or structures.

The following figures exemplify an application of the UNH to a short piece of data, namely the first "book" element of case 1.1 of the XML Query Use Cases (Chamberlin et al. 2003). Fig. 1 reproduces the XML element, Fig. 2 represents the corresponding typed graph, and Fig. 3 shows how this graph may be represented in an untyped network. Later we'll examine more complex cases.

Fig. 1: first book element of Chamberlin et al., case 1.1

```
<book year="1994">
    <title>TCP/IP Illustrated</title>
     <author><last>Stevens</last><first>W.</first></author>
     <publisher>Addison-Wesley</publisher>
    <price> 65.95</price>
</book>
```

Fig. 2: typed graph of Fig. 1

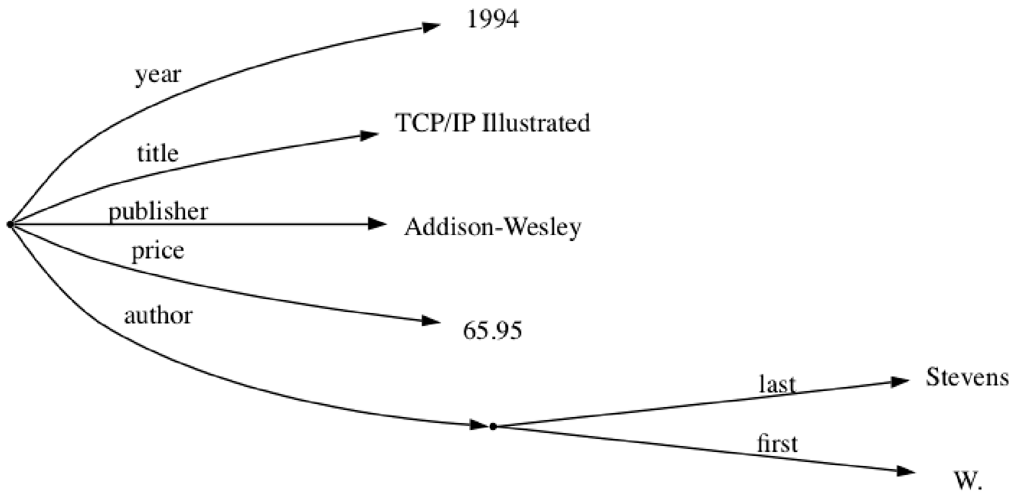
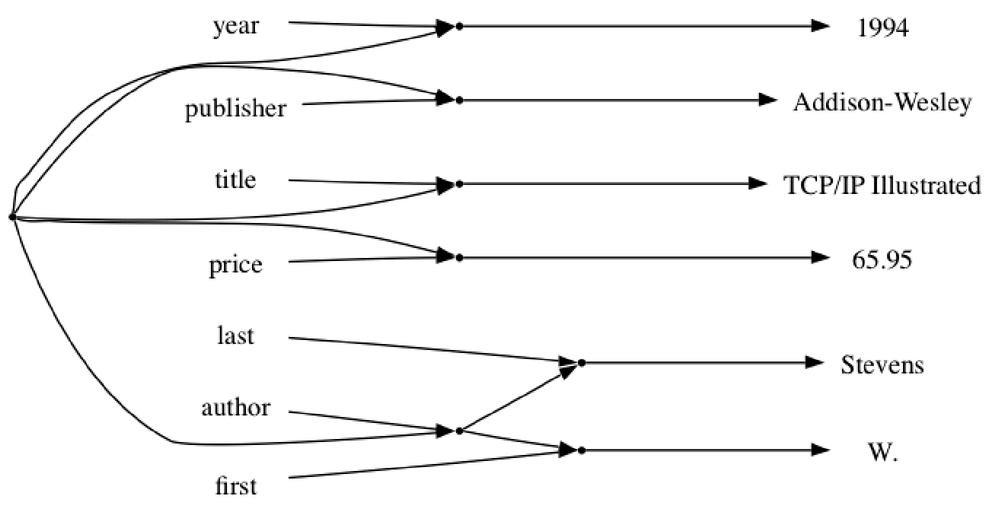


Fig. 3: representation of Fig. 2 in an untyped network



It should be clear by now that "untyped network" means untyped, or unlabeled, links.

The main goal of this research is the betterment of database systems along two axes simultaneously: semantic expressiveness, internal construction.

The complex data requiring representation are a composition of atomic values, attributes, links, lists, records, sets, vectors. This set of structures is deemed sufficient to represent any arbitrarily complex "semantic model" (Rishe 1993; see also RDF, XML).

In the XML strand of literature, this kind of data is often called "semi-structured data." At least for the purposes of the current paper, this designation is misleading, as the complex structures under study are fully defined, there being nothing "semi" about them.

The semantic models strand of research has spawned mainly from observed limitations of the relational data model (Date 19??, section ?, and references thereof). But given the historical weight of the relational model, most proposals have tried to keep it as an element of the theory, and have focused on defining a mapping between the two models (Rishe 1993 and references thereof). This approach forcibly configures a data architecture with two or more levels of logical representation, with the relational model at the base i.e. playing the role of interface with the digital computer representation.

Approaches that depart entirely from the relational model have the potential to eliminate this extra layer in the architecture. From the point of view of software engineering i.e. of the internal construction of the system this is unquestionably a very good point to pursue.

So-called "pure XML databases" constitute a recent example of this 'relational-free' approach. However they reintroduce an extra layer of XML processing. And XML has problems of its own for semantic modelling. Perhaps the most notorious of these problems is the suppliance of two redundant constructs for representing logical attributes, namely element attributes and element containment. More often than not, there is no rationale for the preference of one of these constructs over the other. The XML Query Use Cases (Chamberlin et al.) are full of examples of this. For example the "book" element above (Fig. 1) has "year" as an attribute and "price" as a (sub)element, for no apparent reason other than to show how XQuery can process both kinds.

The Untyped Network Hypothesis takes a fresh look at data representation, eliminating these problems at the root. We are motivated by the observation that the untyped network is a data structure with very good computational properties, and, at the same time, great semantic expressiveness potential. The UNH explores this potential by studying efficient correspondences between the untyped network and the fully fledged structures required for semantic modelling.

In the following sections of this paper we present a development of the UNH as it is being prototyped in the Mneson system (MNESON). We fully present the Mneson data model, which we find a good enough, if not the best, instantiation of the possible UNH-based data models. We also present: the Mneson Calculus, the primitive functional apparatus of the untyped network; Uniformal, a textual language for description of complex structures; and Arrow, a language for direct manipulation of the untyped network. Next we explain why we find Mneson a good instantiation of the UNH, and discuss possible alternatives, diving slightly into implementational elements. We conclude with a view to the future.

Mneson Structures

The Mneson structures are the manner in which information, or data, is represented in Mneson. The Mneson structures consist of

- a base graph configuring simple structures,
- complex structures constructed in terms thereof,
- predefined structure instances representing system entities.

The construction of complex structures is either direct (in terms of simple structures only) or indirect (in terms of other complex structures).

The specified set of structures is judged necessary and sufficient to represent data of arbitrary complexity. This set of structures is called primitive and consists of the base graph, attributes, lists, records, sets, vectors, and operative structures. Each structure is defined in a dedicated section.

We differentiate conceptually between a structure and a concrete realisation, or instantiation, of the structure. However, depending on context, a structure as well as an instantiation of a structure can be called simply by the specific name of the structure. Additionally, the base graph is also called untyped graph, and an instantiation thereof is also called a Mneson graph.

Base graph

Mneson is based on a graph structure with the following characteristics.

The graph is a pair $\langle V, U \rangle$ where V is the set of vertices and U is the set of edges, or links.

Links are untyped (unlabelled), binary and directed. Namely, a link is a pair ⟨source, target⟩ of vertices. The direction of a link is from source to target.

Only linked vertices exist i.e. $\forall x \in V \ \exists y \in V : \langle x, y \rangle \in U \lor \langle y, x \rangle \in U$.

A vertex is either an atomic data value or a valueless vertex. A valueless vertex has an associated serial number that uniquely identifies the vertex. Each vertex is immutable i.e. its value or serial number cannot change.

There is no further restriction on the base graph structure. Symmetric links are allowed i.e. links $\langle x, x \rangle$. Cycles may occur. Disconnected subgraphs may exist.

Links are also notated with arrows:

$$\langle X, Y\rangle \equiv X \rightarrow Y \equiv \begin{tikzcd} X \arrow[d] \\ Y \end{tikzcd} \equiv Y \leftarrow X$$

Direct attribute

The direct attribute structure captures the common notion of an object, or subject, having a certain property, or attribute.

A direct attribute structure is represented directly as an untyped link S → A in the base graph, where S and A represent the subject and the attribute respectively.

Typed attribute

A typed attribute, or simply attribute, corresponds to the well known concept of database theory, the pair ⟨attribute name, attribute value⟩. For reasons that will become apparent, in Mneson an attribute name is called an attribute type. Furthermore, in order to represent typed attributes in the untyped graph, Mneson introduces the concept of attribute instance.

In the following descriptions, the expressions “target of x”, “source of x”, where x is a vertex, abbreviatedly signifies the vertex situated at the indicated endpoint of a link connecting x, i.e. the vertex y such that ⟨x, y⟩ or ⟨y, x⟩ exists.

(Attribute structure definition)

For each attribute type, there exists a vertex that represents it.

Each target of an attribute type is an attribute instance.

An attribute instance has at most two sources. Of necessity, one source is the attribute type. The other source, if any, is the subject of the attribute.

The targets of an attribute instance, if any, represent the attribute values.

Schematics:

$$\begin{tikzcd} & \text{attribute type} \arrow[d] & \\ \text{subject} \arrow[r] & \text{attribute instance} \arrow[r] & \text{attribute value} \end{tikzcd}$$

(Alternate valueless attribute representation)

Mneson also caters, in a special way, for the useful notion of a valueless attribute, i.e. an attribute that by design never has values. In this case the attribute type can represents the attribute directly, in place of the attribute instance.

That is, the following two structures, where S is a subject and T is an attribute type, are, if X has no targets, alternative representations of the same meaning:

$$\begin{tikzcd} & T \arrow[d] \\ S \arrow[r] & X \end{tikzcd}$$

$S \rightarrow T$

(Attribute type naming)

An attribute type which is a basic value vertex x represents the attribute type named x.

(Typed links)

An attribute equates the well known concept of a typed link, with the following equivalencies of structure and notation:

Attribute	Typed link
subject S	source of the typed link
attribute type T	link type
attribute value V	target of the typed link
$\begin{tikzcd} & T \arrow[d] & \\ S \arrow[r] & X \arrow[r] & V \end{tikzcd}$	$S \xrightarrow{T} V$

Note the typed link notation dispenses with the attribute instance X. When X is not of interest the typed link notation is used in this document, instead of the canonical attribute notation, to denote the equivalent structure.

Sets and Records

(Set structure)

For a vertex that represents a set, its targets represent its elements.

(Record structure)

A record is a set of attribute instances, the components of the record.

Lists

A list of items $X_1, \ldots X_n$ is represented by a chaining of the items via a "next" attribute.

$$X_1 \xrightarrow{\text{next}} X_2 \xrightarrow{\text{next}} \ldots X_n$$

$$X_1 \xrightarrow{\text{next}} X_2 \xrightarrow{\text{next}} \ldots X_n$$

The list is represented by its first element, if any.

Vectors

(Simple vector structure)

A vector is a one dimensional array of integer index.

A vector of index range [First, Last] is represented as a record whose components are of type First to Last.

Overlapping structures

Some structures may overlap. For example: the elements of a set may also be elements of a list, with the resulting combined structure representing an ordered set; two subjects may share the same attribute instance (?).

Mneson Calculus

The Mneson Calculus consists of the operations required to arbitrarily inspect a Mneson graph.

In the following descriptions, letters X, Y denote vertices, an arrow denotes a connection in the untyped graph, the letter S denotes a set of vertices.

Primitive functions

targets (S)
: the (possibly null) set of vertices {Y1, ... Yn} such that $X \rightarrow Yi, X \in S$

sources (S)
: the (possibly null) set of vertices {Y1, ... Yn} such that $X \leftarrow Yi, X \in S$

intersect (S1, ... Sn)
: set intersection, abbreviated S1 ^ S2, or S1 ** S2, for n = 2

subtract (S1, S2)
: set difference; abbreviated S1 - S2

extract (S, N)
: the Nth element of S

set (X1, ... Xn)
: the set {X1, ... Xn}; abbreviated {X1, ... Xn}

count (S)
: the number of vertices of S; abbreviated #S

[Abbreviation for sources, targets?]

Utility functions

extract (S)
: equivalent to extract (S, 1); S must be non null, and should be a singleton

singleton (X)
: equivalent to set (X); abreviated {X}

The relation extract (singleton (X)) = X always holds.

The practical application of the Mneson Calculus requires a mean to inspect a set of vertices. The Mneson Library provides procedure For_Each for this purpose.

Examples

1. Given a subject Sbj and the attribute type Att of a valueful attribute of Sbj, the expression

targets (targets ({Sbj}) ^ targets ({Att}))

represents the (set of) corresponding attribute values.

2. Given an attribute instance Ati of subject Sbj, the expression

extract (sources ({Ati}) - {Sbj})

represents the corresponding attribute type.

Uniformal

Uniformal is a textual language for the description of Mneson structure instances, henceforth simply called structures. For a quick first grasp, below is the representation in Uniformal of the structure in Fig. 2:

```
{year 1994,
 title "TCP/IP Illustrated",
 author (last Stevens, first W.),
 publisher Addison-Wesley,
 price 39.50}
```

It is clear from this example that justa-position represents the typed attribute structure. A typical result of processing a Uniformal text is the creation of the described structure in an open Mneson graph. The manner in which a Mneson graph is open for this purpose is outside the scope of Uniformal. The manner in which the new structure is connected to already existing elements of the open graph is also outside the scope of Uniformal.

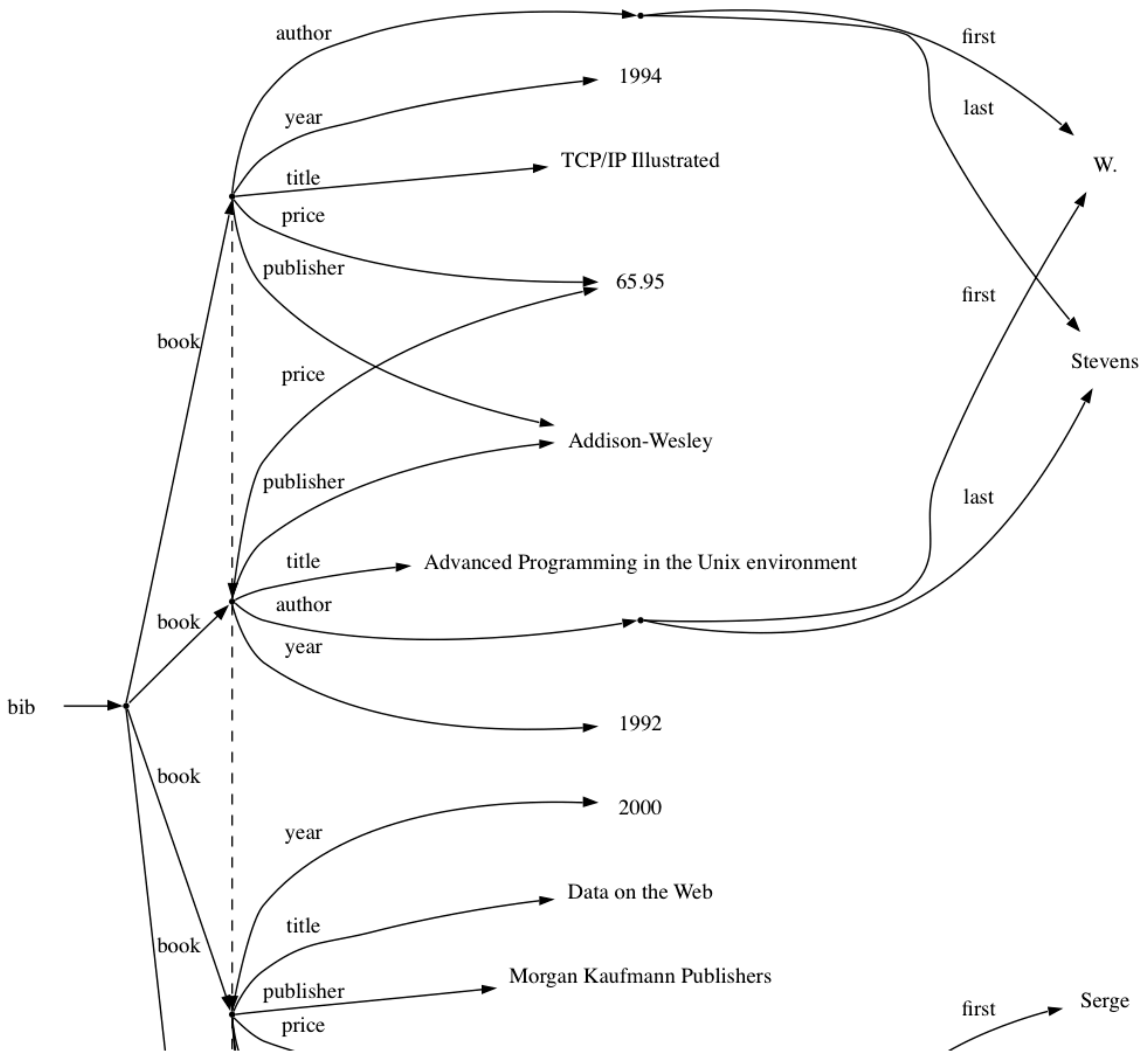
Complete Example

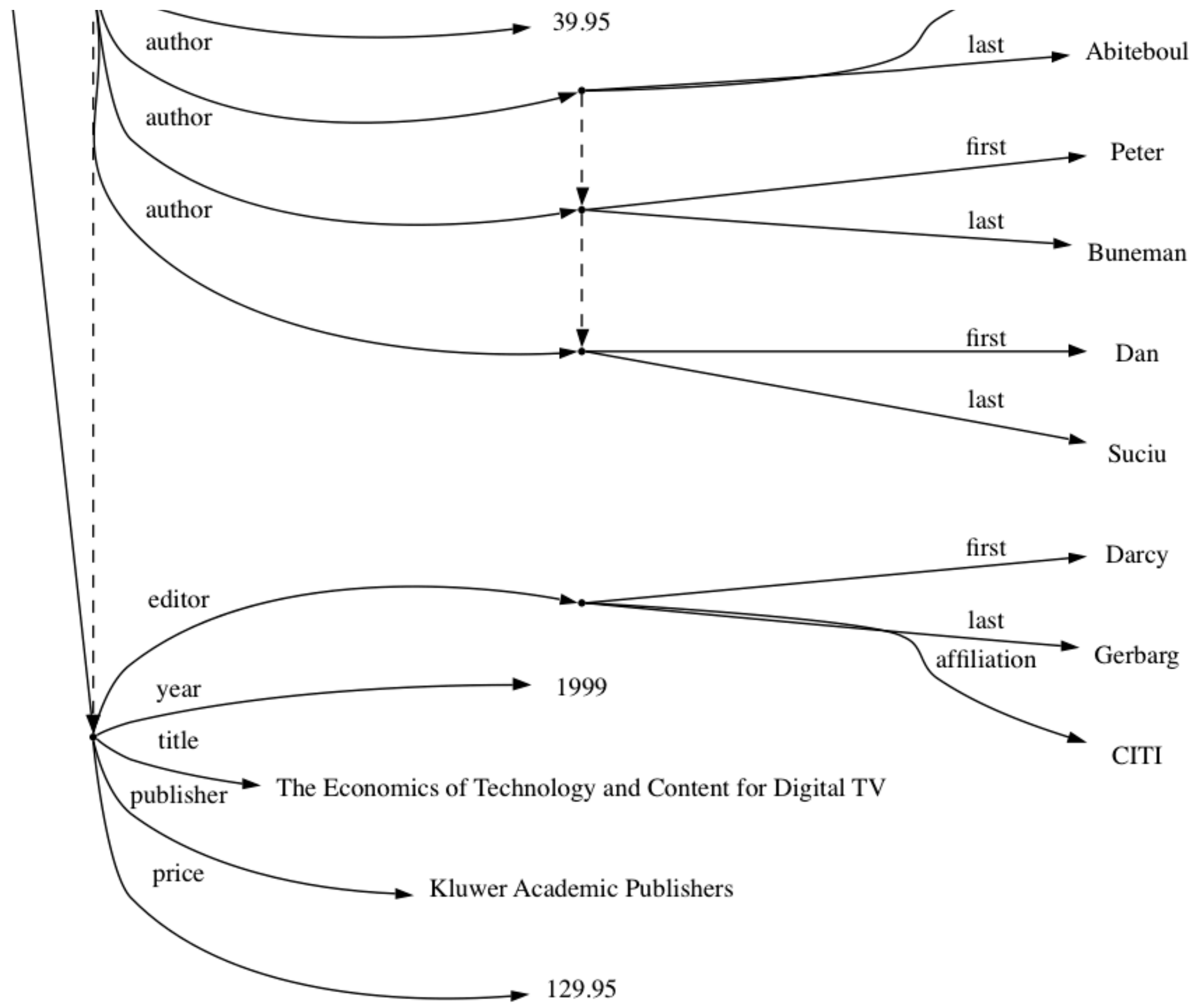
XML Query Use Case 1.1 (Data)

```
<bib>
    <book year="1994">
        <title>TCP/IP Illustrated</title>
         <author><last>Stevens</last><first>W.</first></author>
         <publisher>Addison-Wesley</publisher>
        <price> 65.95</price>
    </book>
    <book year="1992">
        <title>Advanced Programming in the Unix environment</title>
         <author><last>Stevens</last><first>W.</first></author>
         <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
    </book>
    <book year="2000">
        <title>Data on the Web</title>
         <author><last>Abiteboul</last><first>Serge</first></author>
```

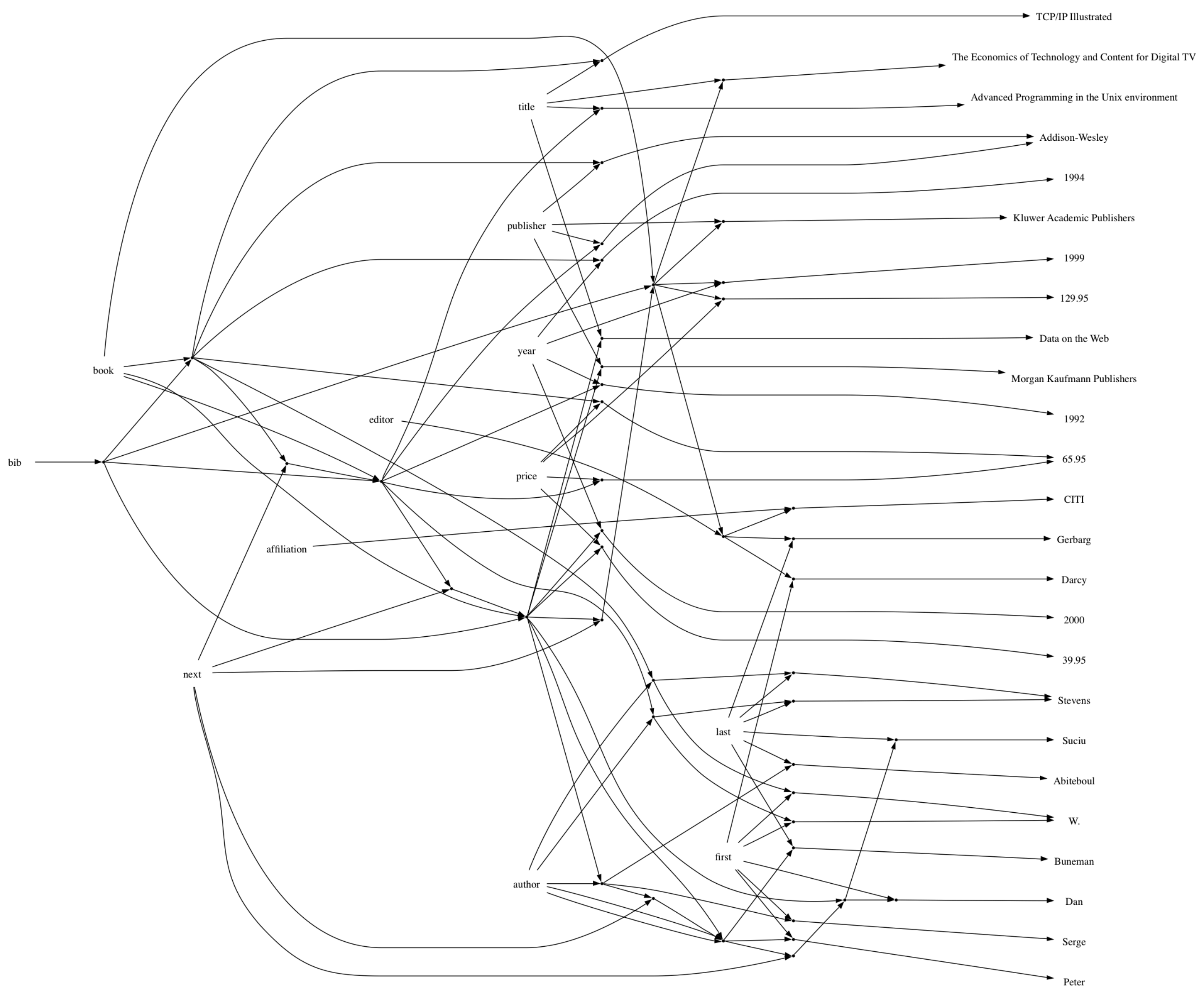
```
          <author><last>Buneman</last><first>Peter</first></author>
          <author><last>Suciu</last><first>Dan</first></author>
          <publisher>Morgan Kaufmann Publishers</publisher>
         <price>39.95</price>
    </book>
    <book year="1999">
         <title>The Economics of Technology and Content for Digital TV</title>
        <editor>
                <last>Gerbarg</last><first>Darcy</first>
                 <affiliation>CITI</affiliation>
        </editor>
            <publisher>Kluwer Academic Publishers</publisher>
         <price>129.95</price>
    </book>
</bib>
```

Mneson graph representations follow. A "next" attribute type is introduced to preserve the linear order of the "book" elements as they appear in the XML source, and also to link multiple "author" elements. The "next" attribute is represented as dashed lines in the typed link view, and normally in the base graph.





Untyped network (base view)



Representation in Arrow:

```
bib
 °(book
    (year 1994,
     title "TCP/IP Illustrated",
     author (last Stevens, first W.),
     publisher "Addison-Wesley",
     price 65.95),
   book
    (year 1992,
     title "Advanced Programming in the Unix environment",
     author (last Stevens, first W.),
     publisher "Addison-Wesley",
     price 65.95),
   book
    (year 2000,
     title "Data on the Web",
     °(author (last Abiteboul, first Serge),
       author (last Buneman, first Peter),
       author (last Suciu, first Dan)),
     publisher "Morgan Kaufmann Publishers",
     price 39.95),
   book
    (year 1999,
     title "The Economics of Technology and Content for Digital TV",
     editor (last "Gerbarg", first "Darcy", affiliation "CITI"),
     publisher "Kluwer Academic Publishers",
     price 129.95))
```

Mneson construction

Mneson represents the base graph as an ordered set of links. Naturally, an order relationship is defined for links. As a link is simply a pair (source, target) of vertices, given an order relation for vertices, an order relation for links is simply the composite order. So an order for vertices is defined, as follows. Vertex order [...] Link order is defined simply as the composite order:

(s1, t1) < (s2, t2) if and only if s1 < s2, or else if s1 = s2 then t1 < t2.

Actually package Mneson.Order provides composite order for tuples of any arity. Mneson link order is realised as an instantiation on Mneson.Order for binary links. The relation above is simply a customized expression of that instantiation.

The logical ordered set of links is realised as two ordered set containers from the AI302 library: a direct set with all the links in their natural order, and an inverse set with all links in their reversed order i.e. a set of links (target, source). This is so to provide inverse navigation, i.e. going from a target to the sources, with the same computational complexity.

The time complexity of AI302 sets is no worse than log (n) for search, i.e. for location of a given element (link), whit n = the size of the set. Location of all targets of a given source X is realised using the provided operation Lower_Bound, which has the same complexity as above. Lower_Bound locates the roof of a given element value, i.e. the lowest element higher than the given. So the Lower_Bound search of (X, Front_Vertex) yields the first link with X as source, if one exists. Retrieving the remaining targets is simply a matter of going up the set up until the source of the retrieved link is greater than X (or the end of set is reached). This operation has constant and negligible time complexity.

So finding all targets of a given vertex has no worse than log (n) time complexity.

The real killer is intersection, namely locating a certain target of a given vertex, when this certain target is defined as the target or source of another given vertex. For example locate the attribute instance of a given record R and attribute type T. For this Mneson employs an algorithm whose precise time complexity is still under study but that can be shown to have an upper bound better than linear on the size of the set. The relevant excerpt from Mneson.Base follows:

```
procedure For_Each_Common_Target
  (Source_1, Source_2 : Vertex; Process : Process_Vertex)
is
   use Link_Sets;
   Next1 : Cursor_Type := Lower_Bound (Links, (Source_1, Front_Vertex));
   Next2 : Cursor_Type := Lower_Bound (Links, (Source_2, Front_Vertex));
   Back1 : Cursor_Type := Lower_Bound (Links, (Source_1, Back_Vertex));
   Back2 : Cursor_Type := Lower_Bound (Links, (Source_2, Back_Vertex));
   Link1, Link2 : Link_Type;
   Tgt1, Tgt2 : Vertex;
begin
   while Next1 /= Null_Cursor
   and Next1 /= Back1
   and Next2 /= Null_Cursor
```

```
      and Next2 /= Back2 loop
         Link1 := Element (Next1);
         Link2 := Element (Next2);
         exit when Link1 (1) /= Source_1 or Link2 (1) /= Source_2;
         Tgt1 := Link1 (2);
         Tgt2 := Link2 (2);
         if Tgt1 = Tgt2 then
            Process (Tgt1);
            Next1 := Succ (Next1);
            Next2 := Succ (Next2);
         elsif Tgt1 < Tgt2 then
            Next1 := Lower_Bound (Links, (Source_1, Tgt2));
         elsif Tgt2 < Tgt1 then
            Next2 := Lower_Bound (Links, (Source_2, Tgt1));
         end if;
      end loop;
   end;
```

This algorithm is similar to the one used in the Intersection operation of AI302 sets (but was invented independently). AI302 is based on the STL, so it is the result of at least three minds, which is a strong indication that is corresponds to the state-of-the-art in optimised implementation of set intersection.

With this set of optimisation techniques, Mneson garantees good enough time efficiency, if not the best possible, for most common database operations. It is our plan to test this assertion by experimental comparison with at least a pure XML database system, over a subset of the XML Query Use Cases.

Other non-relational approaches in the past

[include this section? where?]

Former research on non-relational database systems materialised as the Lore and TSIMNIS systems from Stanford (see Lee et al. 2001 and references thereof). These systems also had problems of their own. Notable, their conception of the primitive elements--vertices and links--was not general enough for arbitrary semantic modelling. Anyway, for whatever problem, their development was abandoned in favour of XML, so now they sustain the same problems as noticed for XML.

Another relational-free approach relates to programming languages, object-oriented or otherwise. In the past, persistent programming languages were researched. The unwritten rationale being that general purpose programming languages are the right tool to represent all system entities, including data, at the base logical level (i.e. the logical level closest to the physical representation). The addition of persistence to the language would therefore eliminate the need for an external database component, relational or otherwise. The database is in the language.

Recently, with large main memory becoming a common and inexpensive asset, so-called "prevalent" systems have been proposed e.g. Prevayler (for Java). For some reason persistent languages failed to pick up. Perhaps prevalent systems will. Anyway, the assumption that the database should be in the language is hard to support. The converse stance, programs in the database, seems more rational, since programs are data.

References

Camberlin et al. 2003
XML Query Use Cases : W3C Working Draft 12 November 2003 / Don Chamberlin [...] (editors).

In: W3C

Lee et al. 2001
: Designing Semistructured Databases: A Conceptual Approach / Mong Li Lee ; Sin Yeung Lee ; Tok Wang Ling ; Gillian Dobbie ; Leonid A. Kalinichenko. -- 12-22 p. In: Lecture Notes in Computer Science, 2113 (2001)

RDF
: Resource Description Framework. In: W3C

Rishe 1993
: A Methodology and Tool for Top-down Relational Database Design / Naphtali Rishe. - 259-291 p. In: Data and Knowledge Engineering, 10 (1993)

W3C
: World Wide Web Consortium. http://www.w3c.org

XML
: Extensible Markup Language. In: W3C