Archives of Control Sciences
Volume 12(XLVIII), 2002
No. 4, *pages 411–431*

Granulating XML information

ERNESTO DAMIANI and RAJIV KHOSLA

The *eXtensible Mark-up Language (XML)* is the standard mark-up language for representing, exchanging and publishing information on the Web. The XML data model, called *Infoset*, represents XML documents as *multi-sorted graphs*, including nodes belonging to a variety of types. XML-based formats are increasingly used as languages for interoperability and agents' communication on the Internet, raising the need for techniques capable to extract and organize heterogeneous XML messages and data while tolerating variations in their internal structure. This paper presents a technique for organizing well-formed XML information items around user provided *graph patterns*. Our approach is based on a *graph granulation* technique that allows agents to extract XML data at different levels of detail, using XML graphs' edges as a hint to semantic relation between nodes. The design and implementation of a software tool for XML data granulation are also discussed.

Key words: XML information granulation, multi-sorted graphs, graph granulation, multiple levels of detail

1. Introduction

The *eXtensible Mark-up Language (XML)* has achieved a remarkable success as a general format for encoding and exchanging information in a number of application fields. XML-based languages are increasingly being used for interoperabilty of distributed applications, as well as within agents' communication protocols, where XML is fostering a new degree of interoperability [14]. XML-based syntax is now the solution of choice in several application domains, including:

- *Brokerage Applications* A wealth of XML information (e.g. about available Web services) is made available via the Web or in special purpose repositories. Brokerage applications need to select the information more closely related to the user interests.

E. Damiani is with Università di Milano, Dipartimento di Tecnologie dell'Informazione, e-mail: edamiani@crema.unimi.it. R. Khosla is with LaTrobe University, Computer Science Department, e-mail: khosla@cs.latrobe.edu.au.

The authors wish to thank Letizia Tanca for common work on flexible XML querying. Thanks are also due to Monica Attolini and Gian Carlo Macchi of Siemens for useful discussions on processing XML messages, and to Luca Bargigia and Alex Redolfi for their work on ASN.1 to XML mapping.

Received 15.11.2001, revised 25.04.2002.

- *Message Management* In network administration systems, huge amounts of XML messages cross administration domain borders, where message structure may change while retaining commonalities in tag repertoire and vocabulary.

- *Web-mining systems* The term "Web mining system" indicates a wide class of services collecting domain or market-place specific information from the Web. Such information is then organized and presented to users in order to allow them to keep under control the information flow about their field of interest, possibly applying data-mining techniques.

While a common data model for messaging and information exchange is highly desirable, it is widely recognized that using XML syntax does not in itself guarantee that information items provided by heterogeneous sources will be easy to understand and process, as each source is likely to use its own XML structure and vocabulary. Standard schemata defining XML-based languages for special purposes may well alleviate this problem, but when such schemata are not available or simply too many to be taken into account (e.g. when new information is discovered), even a modest degree of variability of the information structure makes processing XML data a very difficult task. Building on the approach to flexible XML searching presented in [10] and [9], in this paper we address a scenario where:

- Each processing agent's input information flow relies on a common semi-structured data model, namely XML.

- The flow is composed by a (possibly huge) amount of tagged information items coming from a number of different sources, perhaps sharing a common vocabulary of tags, but employing different (and, *a priori*, unknown) mark-up structures.

Figure 2 summarizes our scenario, showing a processing node or agent whose input flow comes from multiple datasources, some of them previously unknown to the agent itself. Data sources shown in Figure 2 may be other processing agents, Web-based sources or even network devices capable of generating an XML message flow. This situation is rather different from (and, in our opinion, complementary to) the one where agents use granulation to deal with uncertainty on *quantitative* data whose format is well-known [19] as here we deal with uncertainty on the information structure rather than on its content. Nevertheless, like its quantitative counterpart, structure-related uncertainty should be reconciled or tolerated by means of flexible and approximate techniques.

The approach adopted in this paper does not assume processing agents to be fully aware of the syntax employed by the datasources; rather, we only rely on datasources complying with XML format and on the basic XML data model being based on graphs. We use *fuzzy graph granulation* techniques [20], tailored to XML, in order to provide agents with a coarser view of XML information items which may look distinct at a

```
<?xml version="1.0" encoding="UTF-8"?>
<car>
   <maker> Honda </maker>
   <model serialcode = “12303B”>
      <modelname> Civic </modelname>
      <year> 2001 </year>
      <description>
         A popular compact car .
      </description>
   </model>
   <plant>
      <address> Osaka, Japan</address>
   </plant>
</car>
```

Figure 1. A well-formed XML document

finer level of detail[1]. Our technique allows for computing membership to XML granules, seen as a basis for organizing the agents' information space. The paper is organized as follows: in Section 2 a concise introduction to the XML data model is provided, while in Section 3 we first outline our approach in a crisp setting and then, in Section 3.3, we show how the XML graph-based data model can be fuzzified by means of suitable weigthing techniques. In Section 4 our graph granulation technique is introduced, while Section 5 deals with fuzzy graph-matching of XML data based on our approach. Section 5.1 outlines the software architecture of our prototype system, while a sample execution is shown and briefly commented upon in Section 6. Finally, in Section 7 we draw the conclusions.

2. A concise introduction to XML

Generally speaking, an XML dataset is composed of a sequence of nested elements, each delimited by a pair of start and end tags (e.g., `<tag>` and `</tag>`). XML documents can be broadly classified into two categories: *well-formed* and *valid*. An XML document is *well-formed* if it obeys the basic syntax of XML (e.g., non-empty tags must be properly nested and each non-empty start tag must have the corresponding end tag). A sample well-formed XML document that will be used throughout the paper is shown in Figure 1, while its Infoset tree is represented in Figure 3.

[1]Though we shall not elaborate on data semantics is this paper, we remark that coarse, semi-coarse and fine granulation of a message can position the same information differently with respect to a given *problem solving ontology*. The interested reader may consult [15].

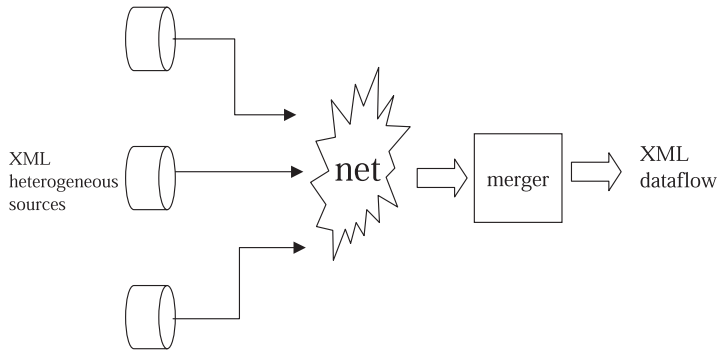

Figure 2. A multi-source XML data flow

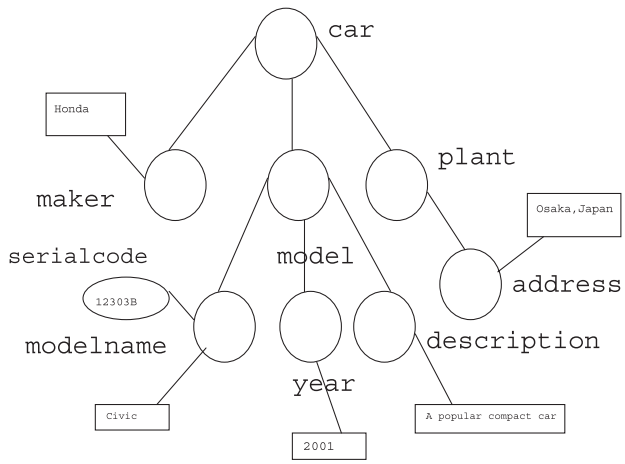

Figure 3. The Infoset tree for the document of Figure 1

The document displayed in Figure 3 contains *elements*, i.e. tags, enclosing other tags and/or textual content; some of those elements (e.g., `<model>`) also have *attributes* (in our case, `serialcode`) holding additional information [2]. Attribute values can be strings, numbers or even *Universal Resource Identifiers* (URI) pointing to other documents or external resources. `ID`s are special attributes whose values are guaranteed to be unique throughout a document; such values may be referenced by other special attributes called `IDREF`s, thus creating *implicit links* between different elements of the same document.

Well-formed XML documents are also *valid* if they conform to a proper *Document Type Definition*(DTD) or XML Schema . A DTD is a file (external, included directly

[2]The Infoset contains several other node types, such as *comments*, *entities* and *notations*. We do not deal with such types in this paper as they are not relevant to our discussion.

```
<!ELEMENT address (#PCDATA)>
<!ELEMENT car (maker, model, plant)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT maker (#PCDATA)>
<!ELEMENT model (modelname, year, description)>
<!ATTLIST model serialcode ID >
<!ELEMENT modelname (#PCDATA)>
<!ELEMENT plant (address)>
<!ELEMENT year (#PCDATA)>
```

Figure 4. A sample DTD

in the XML document, or both) which contains a formal definition of a particular type of XML documents. A sample DTD for the document of Figure 1 is reported in Figure 4.Indeed, DTDs include declarations for all *elements* (i.e. tags) and *attributes*, that will appear in XML documents. Element declarations in DTDs (line 1-5 and 7 of Figure 4) state what names can be used for element types, where they may occur, how each element relates to the others, and wh at attributes and sub-elements each element may have. In turn, attribute declarations (line 6 of Figure 4) in DTDs specify the attributes of each element, indicating their name, type, and, possibly, default value.

Note that special attributes whose type is `ID`(shown, again, on line 6 of Figure 4) are used to uniquely identify elements. Their counterparts are `IDREF` attributes: the `IDREF` associated with an element contains the same value as the `ID` attribute of another element, allowing to create implicit links between elements within a document. Note also that, due to the semi-structured nature of XML data, it is possible (and, indeed, frequent) that two documents complying with the same DTD exhibit a different structure. In fact, some elements in the DTD can be optional and other elements can be included in an XML document zero, one, or multiple times. While substantially longer than a DTD, an XML Schema definition carries much more information, inasmuch it allows the document designer to define XML data structures as reusable *simple and complex types* and then to declare XML elements as variables belonging to those types. A sample schema for the document of Figure 1 is reported in Figure 5. XML Schema is currently the solution of choice for defining XML-based formats for information interchange on the Internet, while DTDs are still widely used by the document management community.

An XML Schema consists of simple and complex *type definitions* and of *element* and *attribute declarations*, indicating the type each XML element or attribute belongs to . Type definitions may be *unnamed*, i.e. given within element declarations, as in Figure 5, or named in order to support reusability. The *validation* or syntax-checking procedure is performed by a *XML validating parser* program and involves a well-formed XML document and a DTD or XML Schema: if the XML document is *valid* with respect to the DTD or Schema, the parser usually produces a memory representation of the document

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema" elementFormDefault="qualified">
   <xsd:element name="address" type="xsd:string"/>
   <xsd:element name="car">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="maker"/>
            <xsd:element name="model" type="modelType"/>
            <xsd:element name="plant" type="plantType"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="description" type="xsd:string"/>
   <xsd:element name="maker" type="xsd:string"/>
      <xsd:complexType name="modelType">
         <xsd:sequence>
            <xsd:element ref="modelname"/>
            <xsd:element ref="year"/>
            <xsd:element ref="description"/>
         </xsd:sequence>
   <xsd:attribute name="serialcode" use="required">
      <xsd:simpleType>
         <xsd:restriction base="xsd:binary">
         <xsd:encoding value="hex"/>
         </xsd:restriction>
      </xsd:simpleType>
   </xsd:attribute>
   </xsd:complexType>
      <xsd:element name="modelname" type="xsd:string"/>
   <xsd:complexType name="plantType">
      <xsd:sequence>
         <xsd:element ref="address"/>
      </xsd:sequence>
   </xsd:complexType>
      <xsd:element name="year" type="xsd:short"/>
</xsd:schema>
```

Figure 5. A sample XML Schema

according to a lower-level data model, such as the *Document Object Model* (DOM) Level 3 [23] standard.

3. Granulating XML graphs

As we have seen, both XML DTD and Schemata contain the full specification of the structure of XML documents, while the design style encouraged by XML Schemata supports expressing at least part of the document semantics via type and element definitions. However, much information exchange in distributed systems relies on *well formed* XML documents, for which no schema or DTD is available. In other cases, processing agents may receive XML messages whose schemata or DTDs are unknown in advance, but specified via pointers inside the messages themselves. Of course, in principle all available schemata could be fetched and agents might embark in the attempt of understanding and processing the incoming XML documents' content on the basis of on their associated metadata. Here, we take a different and complementary approach, processing the input flow of Figure 2 in order to compute fuzzy granules. Intuitively, given an heterogenous XML dataflow and a set of reference graphs (e.g. expressing disjoint message types), we would like to compute a (gradually) coarser view of each message (represented as an XML fragment) and use it to decide how much the message resembles each

reference graphs at each given level of detail. Loosely speaking, each reference graph can be seen as a "structural prototype" in our granulation [1], as the degree of matching between references and imput graphs has the semantics of a fuzzy membership function at a chosen level of detail k. However, as we shall see, our granulation is not a fuzzy partition, since it is easy to see that the total membership value of a single item (again, at a chosen level of detail k) w.r.t. all granules will not amount to one. For the sake of clarity, we shall start by an informal presentation of the main idea in a crisp setting, postponing our fuzzy formulation to Section 4.

3.1. Notation

Following [10], throughout this paper we shall represent XML documents as *labeled graphs* $G = (V, E, L, f, g)$, whose node set V comprises both nodes representing tags and nodes representing text or multimedia content and attributes. Arcs belonging to $E \subseteq V \times V$ are labeled by a function $f : E \rightarrow L$, where $L =$\{e-contains, a-contains,xlink,id-idref\} is a set of *relation labels* which respectively represent XML tag and attribute inclusion, hypertext links, and `ID-IDREF` implicit links. Note also that in this representation graphs are *directed*. Another function $g : V \rightarrow I^*$ (where I^* is the set of strings built over a suitable alphabet I) represents the information content (`PCDATA` in Figure 4) associated to an element or attribute. It is easy to see that a sub-graph representing (element and attribute) element containment only is always a tree, where leaf nodes represent content and attribute values, while internal nodes correspond to XML tags.

3.2. Dealing with differences in structure

The graph-based representation of XML information suggests the idea of granulating the information they contain based on the topology of their graphs. Granulation on vague graphs has been discussed since long, for instance in the context of multi-resolution data handling. Generally speaking, applying a granulation to a graph G is an operation that produces another graph with a coarser structure where some subgraphs collapse into nodes, representing the data modelled by G at a lower level of detail [20]. This operation may be formally described [17] as an extension of the usual partitioning of the set of nodes V of G, with the additional constraint than nodes in each partition block must preserve the graph connectivity, i.e. belong to a subgraph of G; note that nodes can be shared between different blocks. However, graphs representing XML documents require some additional caution, as in XML graphs nodes are multi-sorted and the relation between the structure and the semantics of nodes' content must be taken into account. For instance, the document of Figure 8, while carrying more or less the same semantics of the document in Figure 1, is represented by a fairly different graph (Figure 7). In order to factor out "irrelevant" differences between XML graphs' structures, we need to be able to progressively blur the distinction between the different graphs that can be built over a given set of XML nodes. A simple tool to obtain this result is the well-known *k-closure* operator C_k, which takes a graph G as input and produces a graph G' where an edge is

added (provided it is not already there) between each pair of nodes connected via a *path* of length k (with $k \geqslant 1$) in the original graph.

We adopt a tailored version of this operator that only takes into account *admissible paths*, defined as follows: given a partition of the labels' set L, a path is admissible if all its arcs belong to the same partition block. The trivial partition $L = \{\{e-contains\}, \{a-contains\}, \{xlink\}, \{id-idref\}\}$ admits only paths whose nodes belong to the same Infoset type, while other partitions can be chosen to eliminate at will the distinction between arcs whose labels are distinct in the Infoset, but carry similar semantics in the data flow under consideration. [3]. It is easy to see that $C_1(G) = G$ for all graphs; by convention, we shall denote by $C_\infty(G)$ the (tailored) *transitive closure* of a labeled graph G. Figure 6 shows a graph and its transitive closure $C_\infty(G)$. When computing a relation between a user-provided reference XML graph G and a candidate XML graph G' from the input flow, applying the C_k operator to G' for increasing values of k provides a raw-and-fast (though by no means precise) technique for progressively blurring the distinction between structures that provide similar information but are nested differently [4]. Indeed, as we shall see in detail later, the time complexity (worst-case) of computing C_k (as well as $C_\infty(G)$) is well-known to be $O(\mid V \mid\mid E \mid)$, and several heuristics may be adopted to make the computation of C_k faster. Note that here the value of parameter k corresponds to the *level of detail* of the view on candidate graphs. Once the closure of the XML information item's graph has been computed at the desired level of coarseness, its membership w.r.t. each reference graph is computed via a suitable *matching relation*. Intuitively, relations between G and $C_k(G')$ can be classified according to their strictness [5].

- *Graph Isomorphisms* Matching between document subgraphs is a function φ as above, which in this case is required to be a one-to-one mapping.
- *Graph Embeddings* Matching between subgraphs in the input flow and the reference graph is defined as a function φ associating the reference graph'a nodes to the input flow nodes in such a way that edges and labels are preserved.
- *Graph Simulations* Matching between subgraphs in the input flow and the reference graph depends on the number of paths spanning the same nodes they have in common.

Whichever matching is chosen, the rationale of the procedure described above is comparing each reference graph against each candidate graph in the input flow, after extending the latter in order to by-pass links and intermediate elements which are not "relevant" from the receiver's point of view. However, simply applying the closure operator

[3] For the sake of simplicity, in the sequel we shall refer to the trivial partition only.

[4] This technique may be seen as a generalization to computing path expressions including *wildcards*.

[5] Of course, more sophisticated distance measures can be defined taking into account the fact that nodes may belong to different XML *lexical categories*, e.g. elements in the query graph may correspond to attributes in the document and viceversa.

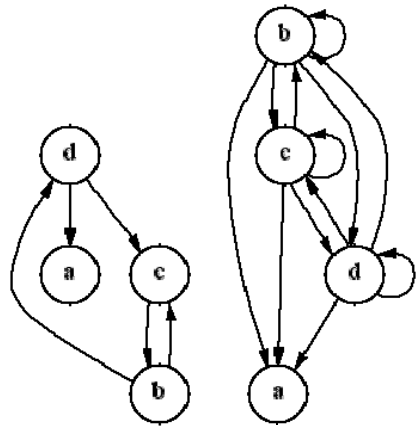

Figure 6. A graph and its transitive closure $C_\infty(G)$

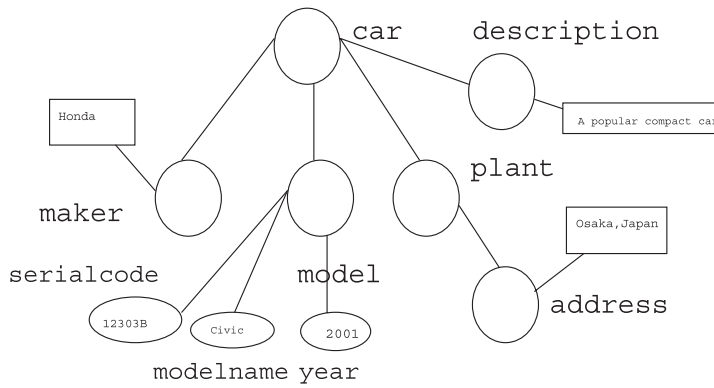

Figure 7. The Infoset tree for the document of Figure 8

C_k to crisp graph candidates does not guarantee that the extension will be performed in a sensible way: our assumption that an edge between nodes suggests a semantic relation between them is clearly weakened when new edges are added as the result of applying the closure. In the next Section, we shall see how fuzzy techniques can be used to achieve this goal.

3.3. Weighting XML information

As outlined in Section 3.2, before matching a candidate graph to the set of references, we need a way to filter the closure result, trimming the closure graph to retain only "promising" closure edges. To this end, we shall first evaluate the *importance* of well-formed XML information at the granularity of XML elements, relying on *fuzzy weights* to express the *relative importance* [3] of information at the granularity of XML elements and attributes. Low weight values correspond to a negligible amount of information,

```
<?xml version="1.0" encoding="UTF-8"?>
<car>
   <description>
      A popular compact car.
   </description>
   <maker> Honda </maker>
   <model serialcode="12303B" modelname="Civic" year="2001">
   </model>
   <plant address="Osaka" country="Japan"/ >
</car>
```

Figure 8. A semantically equivalent, but topologically different XML document

while a value of 1 means that the information provided by the element (including its position in the document graph) is extremely important according to the document's source. Other than that, the semantics of our weights is only defined in relation to other weights in the same input flow. Needless to say, we would like the computation of such weights to be carried out automatically.

3.3.1. Topology-based weighting

Two main approaches can be used to compute automatically an estimate of elements' importance: *tag-related*and *structure-related* document weighting [10]. Tag-related weighting labels *the nodes* of an XML graph with their relative importance, by means of a function $w_{node} : V \rightarrow [0, 1]$. [6]. Tag-related weights can be specified by the document designer or computed automatically taking into account the frequency profile of the set of terms composing the elements' content in the target document. However, tag-related weighting techniques are not suitable to the on-line scenario of Figure 1. Therefore, we exploit a quick-and-dirty structure-related technique, that weighs *the arcs* of an XML graph using topological parameters related to the position of XML elements and attributes, giving a fuzzy graph in a natural way. Namely, we compute a function $w_{arc} : E \rightarrow [0, 1]$ estimating the importance of the arc based on its *nesting*, i.e. the length of the path to the terminal element of the arc from the document root node, and *fan-out*, i.e. the number of elements/attributes directly contained in the terminal element of the arc under consideration. Though in this paper we shall deal with input flow weighting only, our techniques are readily extendable to take into account weighted reference graphs as well.

The choice between the weighting models described above is bound to be application and even dataset-specific [9]. We shall comment on a particular weighting technique selection in Section 6. In both the above models, our weights are constant and do not

[6]Note however that, being weights associated to nodes rather than arcs, the resulting graph is not a fuzzy graph in the classical sense [17]

depend upon the content/value of the XML element or attribute involved; this is indeed a drawback which limits the semantics of weights. For instance, an XML element such as <PRICE> could be considered important only when its content lies inside a given range of values (for a detailed discussion on this subject see [12]). In principle [11], this problem could be solved introducing an additional dependency between weights and the content/value of the corresponding element/attribute. Checking this additional dependency is however bound to be computationally very expensive.

3.3.2. Weight computation

We are now ready to outline the actual computation of the fuzzy weights. Using the structure-related technique, weighting is performed by means of a function $w_{arc} : E \rightarrow [0, 1]$ to weigh each arc $(v_i, v_j) \in V$ of the target document graph G. The basic requirement for w_{arc} is to be fast enough for on-line computation, while capturing at least some of the document semantics. The most straightforward choice is to rely on the fact (by no means certain, nevertheless well attested in other fields: see for instance [2] for object-oriented data) that outer elements (such as <car> in Figure 1) will tend to be more general than the inner ones, and therefore will carry most of the document semantics. A simple function based on this assumption is the *normalized nesting level*, that takes into account the generality of defined as follows:

$$w_{arc}(v_i, v_j) = \frac{d_{max} - l}{d_{max}} \tag{1}$$

where d_{max} is the length of the longest path starting from a root node and l is the distance from v_j to the root. This function establishes a simple inverse relation between arcs importance and their nesting level. Of course, not all nodes at the same nesting level should be treated equally. The nesting level can be aggregated with another function associating to each arc (v_i, v_j) in G the *normalized cardinality* of the sub-tree G' (obtained taking inclusion arcs only into account) whose root is v_j, namely

$$w_{arc}(v_i, v_j) = \frac{\mid G'(v_j) \mid}{\mid G \mid} \tag{2}$$

Note that in this case $w(v_i, v_j)$ does not depend at all on v_i. Also, arcs from element to attribute nodes enjoy no special status and are weighted as the others. This weighting function is non-monotonic w.r.t distance from root and estimates each arc's importance via the size of the subtree rooted in its final node.

Applying the above weighting procedure to the well-formed document in Figure 1, and using the function of Eq.1, we obtain Table 1.

4. Fuzzy closure-based granulation

Once the weighting is completed, the *fuzzy closure* FC_k of the fuzzy labelled graph is computed. As we discussed earlier, computing graph-theoretical closure entails inser-

Table 1. Structure-related weights for the sample document in Figure 1

n_i	n_j	w
car	maker	2/3
car	model	2/3
model	serialcode	1/3
model	modelname	1/3
model	year	1/3
model	description	1/3
car	plant	2/3
plant	address	1/3

ting a new arc between two nodes if they are connected via a path of any length of arcs in the original graph. The weight of each closure arc $P = v_0, ..., v_n$ in $C - G$ is computed as the reciprocal of the ρ-length [17], of the underlying path, defined as follows[7]:

$$\rho(P) = \Sigma_{i=1}^{n} \frac{1}{w(v_{i-1}, v_i)} \tag{3}$$

If $n = 0$, we define $\rho(P) = 0$. For instance, with reference to Figure 3, the path P between the `<car>` and `<description>` nodes gets $\rho(P) = 3 + \frac{3}{2} = \frac{9}{2}$ and its final weight is $\frac{2}{9}$ or 0.22. It can be shown that if G is an undirected tree, then ρ is a suitable definition of distance [8] as Eq.3 defines a *metric* on the node-set V. Namely we get for all nodes x,y, z:

$$\rho(x, y) = 0 \Leftrightarrow x = y \tag{4}$$

$$\rho(x, y) = \rho(y, x) \tag{5}$$

$$\rho(x, z) \leqslant \rho(x, y) + \rho(y, z) \tag{6}$$

In our case, since XML trees are directed, we are only interested in Eq. 4 and Eq.6, that hold respectively by definition of ρ and of path concatenation.

4.1. Extracting granule members

We are now ready to outline our granulation procedure, which relies on the following procedure:

1. Weight each candidate graph G' according to structure-related or tag-related techniques described in Section 3.3. Note that should the input flow contain multiple copies of the same document, our weighting lends itself to caching quite well, as weights on input information items can be computed once for all (in most cases, at the cost of a visit to the document tree).

[7] Note that, in the crisp setting, the ρ-length is simply the length of the path, as all the weights evaluate to 1.

[8] If G is a graph, then one needs to take the minimum of $\rho(v_i, v_j)$ over all possible paths connecting v_i to v_j in order to get a metric [17].

2. Compute the fuzzy closure FC_k of G' (for a given value of k) using the reciprocal ρ-length to weight the closure arcs. This operation is clearly dominated by incidence matrix multiplication, its worst case complexity is $O(\mid V \mid\mid E \mid)$ while its average-case complexity lies in between $O(n^2)$ and $O(n^3)$ where n is the cardinality of the node-set V of the target document graph. Again, graph closure can be pre-computed once for all and cached for future requests.

3. Perform a *cut* operation on $FC_k(G')$ using a threshold (this operation gives a new, tailored target graph TG'). The cut operation simply deletes the closure arcs whose weight is below a user-provided threshold α, and is linear in the cardinality of the edge-set of $C - G$.

4. Compute a fuzzy similarity matching between the subgraphs TG' of the tailored document and the reference graph G, according to selected type of matching. This operation coincides with the usual query execution procedure of pattern-based query languages, and its complexity can be exponential or polynomial w.r.t the cardinality of the node-set V of the target document graph [8], depending on the allowed topology for queries and documents [7].

The first steps of the above procedure are reasonably fast (as document weights and closure can be pre-computed, required on-line operation consists in a sequence of one-step lookups) and does not depend on the formal definition of weights. The last step coincides with standard pattern matching in the query execution of XML query languages [6], and it clearly dominates the other steps. Experience with graph-based query languages for the WWW [8] has shown that, in the case of graph embedding, its worst-case complexity turns out to be polynomial in $\mid V \mid$ for simple tree-shaped queries. Therefore, the total worst case complexity of our procedure does not exceed $O(n^4)$.

5. Fuzzy matching algorithm

In this section we shall outline the fuzzy matching algorithm [18], used for computing the candidate degree of matching with respect to the reference. Regardless of the notion of graph matching that is employed (Section 3.2), our procedure consists of three steps:

1. Given the reference $G = (V, E, f)$ locate a matching subgraph $G' = (V', E', f')$ in the extended document graph (using, for instance, crisp depth first search), such that that there is a mapping $\varphi : V \to V'$ preserving arcs and arc labels. A procedure `FindMatch` is used, according to the desired type of matching[9].

[9] In our current implementation, this procedure also ensures that if values are specified on terminal nodes in the reference graph, they also must appear as content labels of the corresponding nodes in the input document graph.

2. Compute the *granule membership function* $\mu_{G'}$ via standard aggregation as follows:

$$\mu'_G = \wedge_{(v_i,v_j)\in E} w_{arc}(\varphi(v_i), \varphi(v_j)) = T(w_{arc}(\varphi(v_i), \varphi(v_j), ...) \qquad (7)$$

where T is a standard triangular t-norm [16] such as the minimum. Eq.7 suggests three basic comments: (i) Different choices of the t-norm for weights aggregation will result in different matching values. However, here it is not the absolute value of membership that counts, but its relative value w.r.t. other input graphs of the same flow. (ii) In the second part of Eq.7, we straightforwardly use t-norm associativity to compute the conjunction T over all edges $\varphi(v_i), \varphi(v_j)$ that appear in the (closed) document graph and correspond to edges v_i, v_j in the query graph. (iii) Function 7 plays the same role as the objective function in standard fuzzy graph matching algorithms [13], expressing the degree of membership of a candidate subgraph in the result set as a conjunction of the weights on corresponding arcs.

3. Add the subgraph to the granule of G' with membership μ'_G.

Intuitively, delecting the type of t-norm to be used for combining weights means deciding if and how a low weight on an intermediate element should affect the importance of a nested high-weight element. This can be a very difficult problem, as the right choice may depend on the dataset or even on the single data instance at hand. There are some cases in which the t-norm of the minimum best fits the context, other cases in which it is more reasonable to use the product or the Lukasiewicz t-norm. Often, it is convenient to use a family of t-norms indexed by a tunable parameter. In general, however, it is guessing the right context, or better the knowledge associated to it from some background of preliminary knowledge, that leads to the right t-norm for a given application. This context-based technique was discussed in some detail in [10].

5.1. Software architecture

We shall now describe the architecture of a software system based on our approach. It is composed of two main modules, the `Pattern Locator` and the `Smusher`, corresponding to operations at two different levels of granularity. The `Pattern Locator` module is the core of our design. First, it parses and pre-processes the input tree. Then, it uses a `FindMatch` function to look for fragments of the target document having a *topological similarity* with the reference graph. The `Smusher` is a service module, which is called by the `FindMatch` function of the Locator to perform XML *node smushing*, i.e. to evaluate similarity between elementary granules of information (such as XML nodes with their content and attributes) and create result nodes more suitable for user output. The final result of a `Pattern Locator` execution is a granule of XML fragments, ordered according to their membership value; this list is sent to a final `Post-Processor` module that organizes it in a catalog suitable for user consultation or further processing. Figure 9 depicts our architectural design.

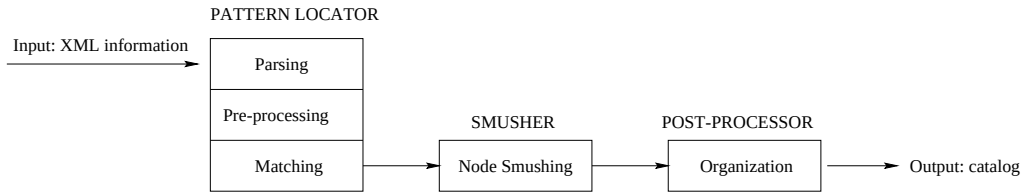

Figure 9. The Architectural Design

The `Pattern Locator`'s operation relies on the following procedure:

1. Parse the reference, obtaining a standard DOM tree.

2. Parse each input XML document, resulting in a weighted, extended DOM tree where nodes are `ExtendedNode` objects. Arc weights are computed once for all, at the cost of a visit to the document tree.

3. Compute the closure FC_k (for a user-selected value of k) by visiting the extended DOM tree and calling the `Closure` method. The `Conj` function passed to `Closure` computes a fuzzy aggregation of the arc weights; choice of `Conj` is dataset dependent and can be done by taking user feedback into account [10]. Again, extended DOM closure can be pre-computed once for all and cached for future requests.

4. Perform a visit of the closure tree eliminating from each `Arcset` the arcs whose weight does not reach a user-defined threshold (this operation gives a new, tailored extended DOM representation of the target document). Thresholding simply deletes the closure arcs whose weight lies below a user-provided threshold α, and costs an additional visit to the document tree.

5. Pass the reference and the candidate document's DOM trees to the `FindMatch` function to evaluate the *similarity matching* between the subtrees of the tailored target document and the pattern tree. In the current implementation, `FindMatch` uses the services of the `Smusher` module to evaluate similarity between nodes' content.

6. A sample application

In this section we briefly present the results of a sample execution of our system in a practical application setting, related to distributed systems administration agents. Our scenario requires on-line granulation of a flow of XML data encoding requests and status messages coming from different network devices. For the sake of simplicity, the XML flow presented here is composed by simplified requests and event notifications,

produced by a real-world XML-based environment for network administration. The flow contains only 12 messages, whose graph structures are to be granulated according to two reference patterns, representing respectively a local and a remote request for a service. Note that:

- *outliers*, i.e. messages that are NOT service requests, may be interspersed in the flow. While we could introduce additional patterns to classify them, here we are interested in filtering them out.
- No XML Schema or DTD information for incoming messages is available at the receiving end.
- There is no clear distinction in the input flow between local and remote requests other than the fact that (hopefully) the typical local requests' structure resembles the first pattern, while remote requests are "more like" the second one.

Even a simple application like this one involved several choices and parameters' settings, the rationale for which is briefly summarized below.

- Selected coarseness level was $k = 2$ and graph embedding was used to compute fuzzy matching throughout the session. The rationale for these choices is simple: first of all, the coarseness level is roughly equivalent to a "zoom out" parameter, and therefore depends on the total depth of the incoming XML data trees, which in our case are simple three level ones, as well as on the search patterns (see Appendix). Graph embedding does not require the cardinality of nodes in the pattern to exactly match the one in the target XML trees, and is less precise than isomorphism; however, it usually increases recall.
- Bi-directional closure and min t-norm aggregation were used for this trial execution. Generally speaking, the bi-directional closure increases recall as it captures the relation between siblings as well as the father-child one; in our case, however, both types of closure give the same result. The min t-norm aggregation is the standard conservative choice.
- Pure topology-based weighting was performed, as a certain difference in structure between local and remote request messages was present in most cases. Would a difference in cardinality be more apparent in this application (for instance, remote request messages showing a higher number of leaf nodes than local ones) a mixed or pure fan-out based weighting technique could have been chosen in lieu of the topology-based one.

The parameter selection interface of our experimental environment is shown in Figure 10 Figure 11 shows the effect of a low pruning threshold (namely $\alpha = 0.1$). The system detects three granules A, B and C, composed respectively of documents carry resemblance to the first or the second pattern only, and to both. Namely C contains 8 candidate messages, while A and B carry two messages each. One message for each pair in A and

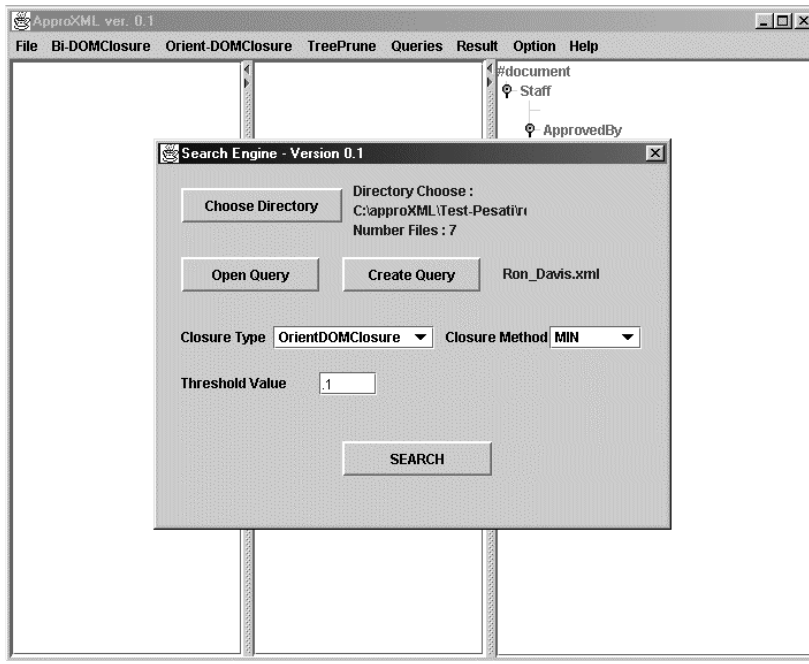

Figure 10. Selecting the closure and t-norm to be used for granulation

B is however clearly an outlier, as its nonzero membership value is very low; indeed, those messages turn out not to be naming service requests at all, and therefore the distinction remote vs. local does not apply to them. The application of a higher threshold level of $\alpha = 0.4$ (Figure 12) reduced the size of granules A and B accordingly, while the application of higher thresholds does not change the granules' content. Granule C contains message whose structure is considered compatible with both being local or remote; content-based check need to be used to resolve the structural ambiguity, though membership values provide an indication, suggesting an order for executing it that could limit the waiting time for real service requests.

7. Conclusions

We presented a technique tailored to organize XML information provided by heterogeneous application sources. Our current application field is using the structure of messages rather than their associated metadata in order to guess the purpose of XML messages generated by a number of heterogeneous and *a priori* unknown sources and

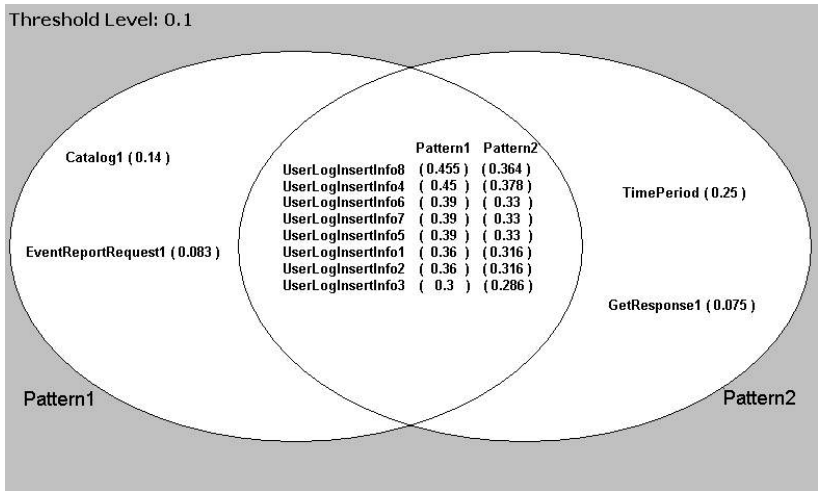

Figure 11. Results for low threshold $\alpha = 0.1$

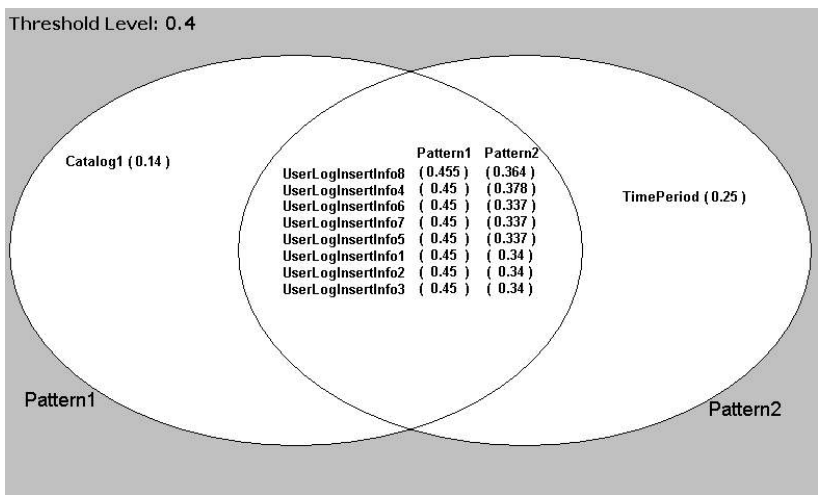

Figure 12. Results for threshold $\alpha = 0.4$

received by network administration agents. However, we believe granulating XML information has a number of other applications, including identifying new Web-based products and services in the framework of Web mining systems. We are currently working on extending our granulation system capabilities in this direction.

Appendix: reference patterns

In this Appendix we report the reference patterns (Figure 13 and Figure 14) that were used to compute the granules of Section 6. Nodes' content, while included for clarity in the figures, is not really relevant to our structure-based granulation.

```
<?xml version="1.0" encoding="UTF-8"?>
  <Header>
    <ServiceName>
      <UserName>ROOT</UserName>
    </ServiceName>
  <Packages>Admin</Packages>
    <Description>GetNodeName</Description>
    <FuncAttributeValue>
    <GuiAlarmStatus>19AEFh</GuiAlarmStatus>
    </FuncAttributeValue>
  < / Header>
```

Figure 13. The header of the first pattern

```
<?xml version="1.0" encoding="UTF-8"?>
  <Header>
    <ServiceName >
      <UserName>ROOT</UserName>
      <GlobalForm>someURI</GlobalForm>
    < /ServiceName>
    <TimePeriod>
      <Unit type="seconds">12</Unit>
    </TimePeriod>
    <Description>GetNodeName</Description>
    <FuncAttributeValue>
      <GuiAlarmStatus>2733Bh</GuiAlarmStatus>
    </FuncAttributeValue>
  </Header>
```

Figure 14. The header of the second pattern

References

[1] J. BALDWIN and T. MARTIN: Fuzzy Modelling in an Intelligent Data Browser. *Proc. FUZZ-IEEE*, Yokohama, Japan, (1995), 1171-1176.

[2] G. BORDOGNA, D. LUCARELLA and G. PASI: A Fuzzy Object Oriented Data Model. *IEEE Int. Conf. on Fuzzy Systems*, **1** (1994), 313-317.

[3] P. BOSC: On the Primitivity of the Division of Fuzzy Relations. *Soft Computing*, **2**(2), (1998).

[4] B. BOUCHON-MEUNIER, M. RIFQI and S. BOTHOREL: Towards General Measures of Comparison of Objects. *Fuzzy Sets and Systems*, **84** (1996).

[5] D. BOX, A. LAM and D. SKINNARD: XML: Beyond Markup. DevelopMentor Series, Addison-Wesley, 2001.

[6] S. CERI and A. BONIFATI: A Comparison of Four XML Query Languages. *SIGMOD Record*, **29**(1), (2000).

[7] R. COHEN, G. DI BATTISTA, A. KANEVSKY and R. TAMASSIA: Reinventing the Wheel: An Optimal Data Structure for Connectivity Queries. *Proc. ACM-TOC Symp. on the Theory of Computing*, S.Diego, USA, (1993).

[8] S. COMAI, E. DAMIANI, R. POSENATO and L. TANCA: A Schema-Based Approach to Modeling and Querying WWW Data. In H. Cristiansen, (Ed)., *Proc. of Flexible Query Answering Systems*, Roskilde, Denmark, *Lecture Notes in Artificial Intelligence* **1495** Springer, (1998).

[9] E. DAMIANI and L. TANCA: Blind Queries to XML Data. *Proc. DEXA*, London, UK, (2000). *Lecture Notes in Computer Science*, **1873** Springer, (2000), 345-356.

[10] E. DAMIANI, L. TANCA and F. ARCELLI FONTANA: Fuzzy XML Queries via Context-based Choice of Aggregations. *Kybernetika*, **4**(16), (2000).

[11] D. DUBOIS, F. ESTEVA, P. GARCIA, L. GODO, R. LOPEZ DE MANTARAS and H. PRADE: Fuzzy Set Modelling in Case-Based Reasoning. *Int. J. of Intelligent Systems*, **13**(1), (1998).

[12] D. DUBOIS, H. PRADE and F. SEDES: Fuzzy Logic Techniques in Multimedia Database Querying: A Preliminary Investigations of the Potentials. In R. Meersman, Z. Tari and S. Stevens, (Eds), *Database Semantics: Semantic Issues in Multimedia Systems*, Kluwer Academic Publisher, 1999.

[13] S. GOLD and A. RANGAJARAN: A Graduated Assignment Algorithm for Graph Matching. *IEEE Trans. on on Pattern Analysis amd Machine Intelligence*, **18**(2), (1996), 104-118.

[14] S. HAUSTEIN and S. LUEDECKE: Towards Information Agent Interoperability. In M. Klusch, L. Kerschberg (Eds), *Cooperative Information Agents*, Lecture Notes in Artificial Intelligence **1860** (2000), 208-219.

[15] R. KHOSLA, I. SETHI and E. DAMIANI: Intelligent Multimedia Multi-Agent Systems. Kluwer Academic Publisher, 2001.

[16] G. KLIR and T. FOLGER: Fuzzy Sets, Uncertainty and Information. Prentice Hall, 1988.

[17] J. MORDESON and P. NAIR: Fuzzy Graphs and Hypergraphs. Studies in Fuzziness and Soft Computing, Physica-Verlag, 2000.

[18] K. NOMOT: A Document Retrieval System Based on Citations Using Fuzzy Graphs. *Fuzzy Sets and Systems*, **38** (1990), 207-222.

[19] W. PEDRYCZ and G. VUKOVICH: Intelligent Agents in Granular Worlds. In V. Loia, S. Sessa, (Eds), *Soft Computing Agents*, Studies in Fuzziness and Soft Computing, Physica-Verlag, 2002.

[20] J.G. STELL: Granulation for Graphs. In C. Freksa and D.M. Mark (Eds), *Spatial Information Theory. Cognitive and Computational Foundations of Geographic Information Science. Lecture Notes in Computer Science*, **1661** Springer, (1999), 417-432.

[21] W3C. Extensible Stylesheet Language (XSL) Version 1.0. October 2000. http://www.w3C.org/TR/xsl/

[22] W3C. Extensible Markup Language (XML) 1.0. W3C Recommendation, Feb. 1998. http://www.w3C.org/TR/REC-xml/

[23] W3C. Document Object Model Level 3 (DOM) W3C Working Draft, Jan. 2002 http://www.w3.org/TR/2002/WD-DOM-Level-3-Core-20020114/

[24] W3C. XSL Transformations (XSLT) Version 1.0 W3C Recommendation 16 November 1999. http://www.w3.org/TR/xslt