The Data Projection Model: Toward Auditable Information Systems through Unified Declarations of Operations on Data

Michel Biezunski
402 85th Street #5C
Brooklyn NY 11209
mb@infoloom.com

November 14, 2006

Table of Contents

Abstract

This article proposes a model that enables generalized accountability for information systems.

The Data Projection Model provides a common layer on which any information process can be expressed. Regardless how diverse the origins of information items are, each information item can be expressed as a process.

The Data Projection Model enables audit trails to be created for each such process. In the Data Projection Model[1] a unifying mechanism represents any kind of data as involved in a binary relationship. It is called "projection" by analogy with geometric projection used for representing spatial relationships in a 2-dimensional space.

The Data Projection Model strictly distinguishes the declaration of an operation from the

1 Data Projection is used generally in the context of forecasting. Here the term "projection" refers to spatial projection rather than to temporal projection.

processing performed to yield its result. This model is, by design, limited to the declarative part. For example, applied to arithmetic operations, the Data Projection Model describes the operation “2+3” without mentioning that the result is “5”. It's up to the application to determine whether the result should be expressed as “5”, “5.0”, “five”, or something else. Each operation is decomposed into binary processes, always composed of exactly one operator and two operands. The Data Projection Model enables auditing to take place by collecting the operations declared as being involved in one particular process. Furthermore, because it is possible to represent most data as components of binary operations, the Data Projection Model also enables a great variety of data representations to be aligned and cooperate. The Data Projection Model is based on a construct, called “perspector”, which is described here. This paper also illustrates how the model can be used in a number of contexts, including relational databases, structured markup, and semantic linking.

Introduction

The models used for representing data often contain descriptive and procedural information: validation for example is an example of a procedure to be performed over data corresponding to a given model. These models are powerful and rich with many features that make them operational, but they are eventually limited to certain kinds of operations, and they require their users to accept their respective frameworks . Database models correspond to that category. On the other hand, generic markup languages allowing users to define their own set of structural tags, such as XML, are wide open, in principle. However, once a schema has been defined, data must strictly conform to it, and the multiple islands that are created are quite closed. In the terminology of computer basic operations, information semantics representations have a variety of high-level languages to choose from, but there is no equivalent of assembly language, operating just on “0s” and “1s”. There is a need for models which, instead of covering all properties of the objects, have the minimal characteristics necessary to declare the items contributing to a process rather than focusing on the result yielded by the processing. Many different kinds of data which exist in separate universes can be gathered on a common playground, not to be processed, but to be recorded as existing. The major interest of doing so is to enable auditable information systems to exist, which rely on knowing that certain kinds of data are present, and have been introduced at a given time within the system by a means of a given mechanism (human or automated). Such mechanisms might include the creation and management of semantic information, spreadsheet operations, geospatial relations, database schema creation, database record management, version control, document management. Furthermore, since auditing is no more a certain way of looking at information, this model can be used to define other, customized views.

Formal models have been developed to address these issues, e.g. “Formal Concept

Analysis", based on lattice theory. Models for data representation which enable interoperability between systems based on a common understanding of processing rules and structures include the entity-relationship model, UML, MOF, XML for data and document structures, RDF and Topic Maps for semantic linking, among others.[2] Metadata repositories currently used to create and manage auditable systems rely on proprietary mechanisms. Although standard data representations are becoming more and more pervasive, and although applications exist that are able to keep track of auditable systems, there is no generalized way to present information so that it can be audited. Instead of interoperability, the Data Projection Model enables "interviewability", i.e. ways to view diverse information, because everything has been projected to the equivalent of a flat surface.

Most data representation models rely on the distinction between classes and instances. This is the case for database schemas, taxonomies, ontologies, and XML architectures. Processes act on actual instances depending on what types the instances belong to at the schema level, and processing usually is uniform for a given structural component. Unfortunately, it doesn't address the numerous exceptions that emerge in the real world. In brief, there are things that can't just be considered a case belonging to a pre-defined category. These exceptions can lead to potentially complex systems, and make auditability of the operations and processes performed almost impossible because they fall outside of the scope of the standardized mechanisms. The Data Projection Model allows for auditability of such systems, by providing a common ground for diversity to be expressed, exactly in the same way that cameras are used to project any 3D reality to a 2D surface.

By requiring all auditable data to be represented as binary relations, the Data Projection Model "flattens" the representation. The particular nature of the projection indicates the "perspective". Each operation, once declared within a given perspective, is addressable as such and can be documented: it may contain information about who or what created it, when it was created, etc. The Data Projection Model doesn't define precisely what should be the content of these audit trails but it provides a placeholder for them.

The Data Projection Model contains a single construct which, as will be illustrated, can be used in a wide variety of situations. This construct describes an operation which involves two operands. This basic construct is a building block that can be nested to create complex operations. Technical feasibility is based on the hypothesis that any process involving more that two items (such as n-ary relationships) can be decomposed into a series of binary connections. It is also based on the premise that any data item can be expressed as a

2 About the Entity-Relationship model, see: Peter Pin-Shan Chen, "The Entity-Relationship Model – Toward a Unified View of Data", *ACM Transactions on Database Systems*, Vol. 1, No. 1, March 1976, Pages 9-36; UML (Unified Modeling Language): www.uml.org , MOF (Meta-Object Facility): http://www.omg.org/technology/documents/formal/mof.htm, XML (Extensible Markup Language): http://www.w3.org/XML/ , RDF (Resource Description Framework): http://www.w3.org/RDF/, Topic Maps: www.isotopicmaps.org.

component in a binary relationship. To start with, the mere fact that a piece of data exists is because it has been created at a certain moment in time, through a given mechanism. In other words, data cannot be seen without some perspective, but there are multiple perspectives in which the same data can be seen.

This paper is organized in three parts: the description of the model, examples of representing information structures, and conclusions about how this translates into auditability.

The Data Projection Model

The single construct of the Data Projection Model declares an operation in which two data items participate. We propose to call this construct a *perspector*[3]. A perspector is defined as the aggregator of two data items connected through an operator. A perspector is addressable as such, and can be used within other perspectors. A perspector consists of:

- An *operation* involving two data items (only two, and always two).[4] This is the perspector itself.
- The two data items involved in the operation, called the *x-operand* and the *y-operand*. They can not exchange their roles, since the operation is directional.[5]
- The *operator*, which belongs to a class of operations.
- Each perspector declares a specific operation, not a class of operations. The class of operations is uniquely identified, within a given application context, by the string used to designate the operator. The way operations defined with one given operator are used within a given application is called a *perspective*.[6]

3 The term is used in geometry in a somewhat different sense: a perspector is "the point at which the three lines connecting the vertices of two perspective triangles concur, sometimes also called the perspective center, homology center, or pole." (http://mathworld.wolfram.com/Perspector.-html). Here, a perspector is defined as the aggregator of two data items participating in an operation within a defined perspective.

4 The operation is simply declared. The Data Projection Model does not contain, by design, any indication of how the operation should be performed.

5 If the operation happens to be symmetric, then the *x*-operand and the *y*-operand can be interchanged, but this is a specific case that is defined by applications.

6 For example, an operator called "Naming" could be used in one application to define the process of naming objects. It is possible, and quite likely, that another application would handle the Naming operator differently. For example, one application could consider that names produced by the Naming operation belong to a namespace, i.e. that names are unique. Another application could use this operator without that constraint, therefore allowing the same string to be assigned to different names. What the application does with each operation triggered by a given operator is a matter of perspective, and this is why the notion of perspective is directly related to the operator.

To summarize, these are the building blocks for all applications:

p	perspector
o	operator
x	*x*-operand
y	*y*-operand

Here is the notation we propose to describe a perspector by distinguishing the various roles played by each of its components:

$$p = < x \mid o \mid y >$$ [7]

where *p* is the perspector, *x* is the *x*-operand, *o* is the operator, *y* is the *y*-operand. The "=" sign is used to represents an assignment, "< "and ">" are used to delimit the boundaries of the operation, and the bars "|" are used to separate the *x*-operand, the operator, and the *y*-operand.

In the Data Projection Model, any *x*-operand, any *y*-operand, any operator, and any perspector is always represented by a character string. The character strings define a stringspace: if two *x*-operands are represented by the same string, they are the same operand. The same holds for *y*-operand, operator, and consequently for the perspector itself. The namespace will be called here a "stringspace", because strings do not represent names unless they are explicitely declared to do so. [8]

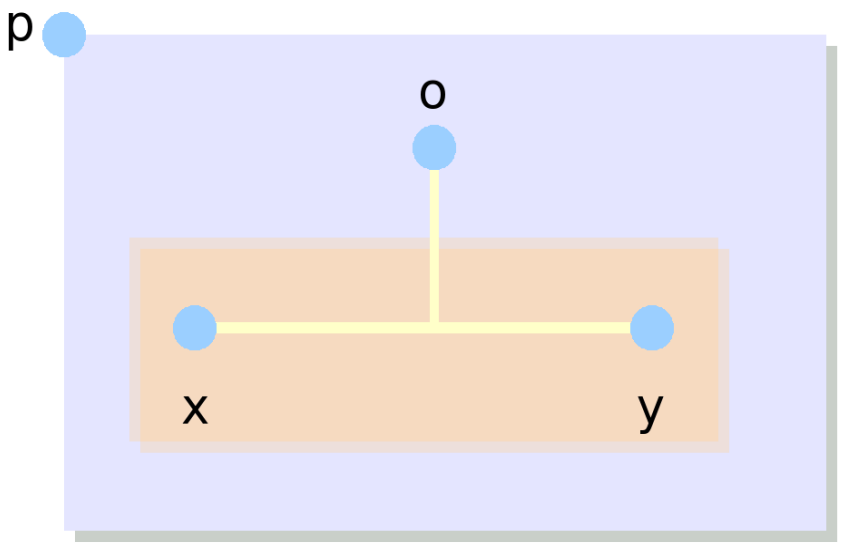
A perspector *p* can be used as a component in another perspector *p'* where it can serve either as an *x*-operand, a *y*-operator, or possibly, if the application permits, even as an operator.[9] There is no limit to the nested levels in which a perspector can be used. Application designers are free to constrain the nesting levels to a number they consider appropriate. The applications may also include provisions to otherwise constrain how perspectors can be used.

---

7 The use of angle brackets in a perspector is made in reference to Dirac's notation for quantum states in quantum mechanics. See for example http://en.wikipedia.org/wiki/Bra-ket_notation. Alternative notations are possible. The perspector `p1 = < 2 | + | 3 >` can be written in XML: `<p id="p1"><x>2</x><o>+</o><y>3</y></p>`,
where `p` is the element type for a perspector, `x` for the *x*-operand, `o` for the operator, and `y` for the *y*-operand. In this article, I will use the Dirac notation, since it is easier to read for a human eye.

8 In such a string space, rdf:about and html:title, for example, would be part of the same unique space. The fact that these two items correspond to different namespaces in the XML sense is irrelevant here.

9 Experience will show whether the ability of a perspector to be used as an operator in another perspector should be allowed, or if it should be prohibited.



p = < x | o | y >

perspector — x operand — operator — y operand [1]

A perspector is purely declarative. It doesn't yield the result which would be produced by actually performing the operation. That is left to applications to define. It may be desirable for high-level cross-industry applications to standardize the expected performance of a number of operations, but this is out of scope of the Data Projection Model. It is of the same nature as the implementation, for example, of the addition operation in various programming languages: 2+3 will (hopefully!) always yield 5 or something very similar, but the way the addition is implemented internally in every language is left to the environment designers to decide. Thus, in:

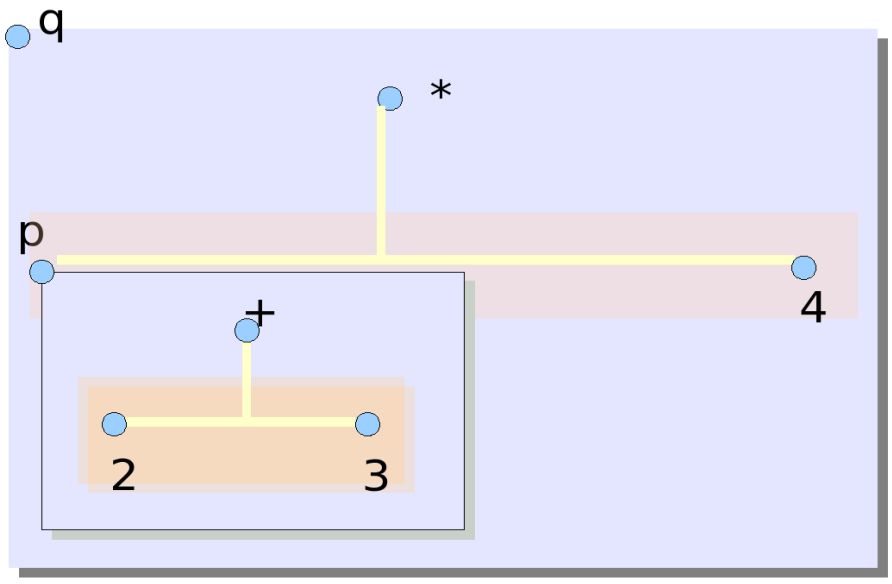
```
p = < 2 | + | 3 >
```

in which "+" represents the addition operator, *p* does not represent the value 5^{10}. It merely represents the operation of addition being applied to the operands 2 and 3. Perspectors can be used as operands for other perspectors. For example,

```
q = < p | * | 4 >
```

where "*" represents the multiplication operator, might be implemented to yield 20, but itself merely describes the fact that the perspector *p* is used as an x-operand for the perspector *q*.

10 Some programming languages will treat it as a float, others as an integer, and the displayed result may actually differ, even if the mathematic value is identical.



$$q1 = < p \mid o \mid y >$$
$$q2 = < x \mid o \mid p >$$
$$q3 = < x \mid p \mid y >$$

are valid constructs, as far as the Data Projection Model is concerned.[11]
Examples

Example 1. Names

Naming is often overlooked as an operation. In this example, we will distinguish between an identifier, which we consider unique within a system, and the name of an object. We want to allow an object to have several names in addition to the identifier by which it is uniquely addressed in our system. In this example, an object *is associated with* a name by means of the operation "Naming". If the object to be named occupies the *y*-operand, then the *x*-operand is considered to be the name. In other words, a name is the opposite operand to the object, here reference by its system identifier, being named via a "Naming"[12] operator.

$$p_1 = < \text{Washington} \mid \text{Naming} \mid \text{X893e3348347} >$$

11 When arithmetic operators are used, the perspector notation resembles traditional algebraic notation (except for parentheses becoming angle brackets and the addition of vertical bars).

12 We could have decided to use the opposite order. But of course, once we have adopted a convention, we need to use it consistently throughout an application. The choice of the word "Naming" for such an operator is arbitrary and is only valid for this example. Application designers are free to use the string they want for the operation describing naming.

[The first president of the United States][13]

p_2 = < Washington | Naming | YyER89384-3490439 >

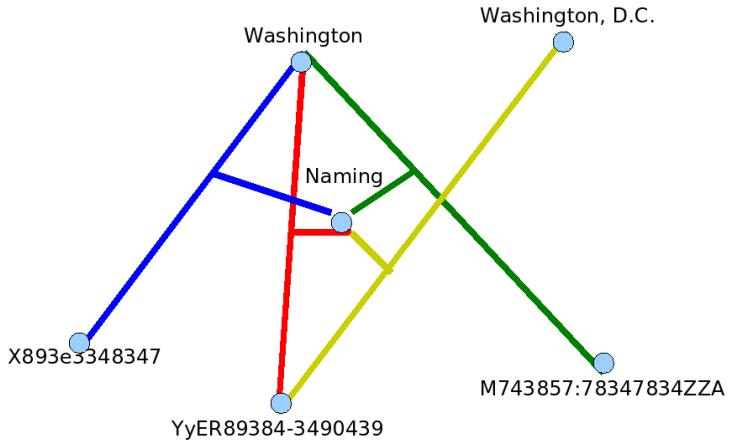
[The capital of the United States]

p_3 = < Washington | Naming | M743857:78347834ZZA >

[The West Coast state above Oregon]

p_4= < Washington, D.C. | Naming | YyER89384-3490439 >

[The capital of the United States]



8

In this example, the string "Washington" is used in three naming operations, and the three perspectors p_1, p_2, and p_3 that describe them are different. The x-operand, i.e., the string "Washington", is unique in the "stringspace" containing the operands, the operators and the perspectors. But, as shown in this example, it may be used to name data items that are quite different. The fourth perspector p_4 has the same y-operand as the second one, p_2. As a result, the application should be able to tell that the data item for the "The capital of the United States" has been associated with two different names.

13 This perspector illustrates the fact that it is just one perspective, because someone else may have preferred to use the string "George Washington" instead. The fact that multiple perspectives exist will never be eliminated. What we can do is to keep a record of the origin of each perspector -- via another perspector -- so that we can trace back where everything comes from.

Example 2. The Entity-Relationship Model

The Data Projection Model can be used to provide a unified view comprising both schemas and instances. Here is a way to express entities and relationships.[14]

First, every entity and relationship in the schema is declared as such:

```
< Employee | Is a | Entity >
< Department | Is a | Entity >
< Project | Is a | Entity >
< Dependent | Is a | Entity >

< Project-Worker | Is a | Relationship >
< Dept-Emp | Is a | Relationship >
< Emp-Dep | Is a | Relationship >
< Proj-Manager | Is a | Relationship >
```

In the Data Projection Model, the entities can be used as *x*-operands or *y*-operands, and the relationships can be used as operators. Thus a unified representation is provided to describe both the schema and the data. Note that in most applications it will not be necessary to explicitly declare entities and relationships, since operands (*x*- or *y*-) may be inferred to be entities, while operators may be inferred to be relationships. In that case, the eight first lines of this example could be omitted, making the entity-relationship model easily expressed using the Data Projection Model.

```
< Employee_1 | Is a | Employee >
< Department_1 | Is a | Department >
< Project_1 | Is a | Project >
< Dependent_1 | Is a | Dependent >
```

and the relationships can be used as operators:

```
< Employee_1 | Project-Worker | Project1 >
< Department_1 | Dept-Emp | Employee1 >
< Employee_1 | Emp-Dep | Dependent1 >
```

14 This example is extracted from the seminal article by Peter Pin-Shan Chen, "The Entity-Relationship Model – Toward a Unified View of Data", *ACM Transactions on Database Systems*, Vol. 1, No. 1, March 1976, Pages 9-36.

```
< Employee_1 | Proj-Manager | Project1 >
```

Naturally, all these perspectors can be output automatically from any relational database system.

Example 3. Representing the structure of XML documents.

XML Schemas, DTDs, or RELAXNG specifications are various ways to express the rules to which a document must conform, i.e. the list of elements, attributes that are allowed and how they are related to each other. These rules also can be represented using the Data Projection model. Here is an example of a possible description of a (small) part of the structure of XHTML documents[15]:

```
< html | contains | head >
< html | contains | body >
< html | namespace | http://www.w3.org/2002/06/xhtml2 >
< html | attributes | common >
< common | contains | Core >
< common | contains | I18N >
< common | contains | Events >
```

etc.

Flattening the expressions in each of the specifications mentioned above, by limiting the expression to binary relationships, is also a process that can be automated.

Example 4: Representating an XML Document Instance

A document instance can be represented using a similar method. Let's suppose we want to describe the following instance (stored in a document called “mydoc.xml”). Here is a description with perspectors that takes into account the Document Object Model specification[16].

```
<book>
    <title>New Perspectives on Information Management</title>
```

15 Again, this is only an example. There are multiple ways to describe the structure of HTML documents. For example, it may be preferable to avoid using groups such as “Common” and instead extensively describe all pairs of relations one by one. It may make sense for the industry to agree on one representation, but each implementation environment may have reasons to prefer one way rather than another.

16 World Wide Web Consortium, *REC-DOM-Level-1-19981001*: http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/level-one-core.html

```
</book>
```

Let's suppose now that the document has been parsed, is now available as nodes in a graph, and is readable using a DOM-based engine:

```
< mydoc.xml | fileName | node0 >
< node0 | NodeType | DOCUMENT_NODE >
< node0 | firstChild | node1 >
< node1 | NodeType | ELEMENT_NODE >
< node1 | tagName | book >
< node1  | firstChild | node2 >
< node2 | tagName | title >
< node2 | firstChild | node3 >
< node3 | NodeType | TEXT_NODE >
< node3 | Text | New Perspectives on Information Management >
```

The two preceding examples could be concatenated so that smooth navigation would be allowed from the document type structure to the document instance and vice-versa. This uninterrupted or unified navigation strategy may be useful for auditing.

Example 5: RDF

Representing an RDF[17] application with the Data Projection Model is straightforward, because RDF is based on triples, and the Data Projection Model is as well. It is always possible to consider the following equivalences:

subject	x-operand
object	y-operand
predicate	operator

There is a one-to-one correspondence between subject and x-operand, object and y-operand, predicate and operator.[18] In addition – and this is the major difference between

17 Resource Description Framework, W3C Recommendation, February 2004. Dave Beckett, ed. : http://www.w3.org/RDF/ .

18 The reason that the RDF terminology has not been simply used as-is is to avoid the assumption that the x-operand necessarily has something to do with a subject, y-operand with an object, and operator with a predicate. The actual semantics of what they are is entirely dependent on the context. But the major difference between the Data Projection Model and RDF is the fact that the operation or perspector itself is always addressable, while in RDF it takes a reification process to make it addressable.

RDF and this model -- every perspector exists as such is always addressable. In RDF terms, this means that the triple will always be reified. Therefore, anything that can be expressed using RDF triples can be expressed using the Data Projection Model.

The N3 syntax for RDF[19] shows that the mapping is straightforward. An expression such as:

```
<#pat><#knows><#jo>
```

can be translated as:

```
< #pat | #knows | #jo >
```

RDFS can also be translated directly using the same constructs:

:Person is a class.

```
< :Person | rdf:type | rdfs:Class >
```

The Data Projection Model can be seen as a way to use RDF triples to represent practically anything. The advantage is the fact that there is a unified representation to describe any connection between two information items, whatever they are.

In a way, the Data Projection Model could be seen as an alternate expression for the core of RDF, without all the bells and whistles. It facilitates the usability of RDF because of its simplicity, but the price is to abandon the ability to describe any specific processing and reasoning that goes with the information. Another layer is necessary to do that. In that sense, the Data Projection Model is to RDF what XML is to formatting. It doesn't contain anything having to do with actual processing. In the context of RDF, processing means validation towards schemas expressed in RDFS, or interpretation of some specific RDF structures such as rdf:Description.

The Data Projection Model makes no differentiation between information expressed in XML, RDF or relational databases. The interpretation of the information depends on what the application needs. This approach can be most efficient in cases of hybrid applications which accommodate various or multiple notations.

Example 6: Topic Maps

Topic Maps is a data representation where a subject of conversation is represented by a proxy, usually called a “topic”. This proxy has a number of properties, either predefined (in the current version of the standard, as well as in the proposed Topic Maps Data Model) or user-defined, in the proposed Topic Maps Reference Model.[20]. Here is an example of a topic T1 named “New York”. One occurrence of the topic is indicated, and relations to one other

19 This example comes from a tutorial given in 2003 by Tim Berners-Lee, Dan Connolly, And Sandro Hawke. http://www.w3.org/2000/10/swap/Primer

20 ISO/IEC 13250: Topic Maps. Information on the standard and on the ongoing work can be found at http://www.isotopicmaps.org

topic are indicated.

```
< T1 | Naming | New York >
< T1 | Occurs | http://www.nyc.gov >
< T1 | Has_Borough | T2 >
< T2 | Naming | Brooklyn >
```

Example 7: Mathematical and Logical Operations

Another example of a domain where the perspectors can be used is mathematical and logical operations[21]:

addition	< *x* | + | *y* >
subtraction	< *x* | - | *y* >
multiplication	< *x* | * | *y* >
division	< *x* | / | *y* >
logical and	< *x* | and | *y* >
logical or	< *x* | or | *y* >

To summarize, the Data Projection Model introduces a construct "perspector", notated "*p* = < *x* | *o* | *y* >", that is able to declare anything that can be expressed as a relationship between two information items represented as character strings.

The Data Projection Model does not establish the rules by which the data are actually processed, nor does it validate the results of any processing. The purpose of the Data Projection Model is not to provide a replacement for any procedural language. In other words, it is not aimed at being the Esperanto for information technologies. Instead, the purpose of the Data Projection Model, as elementary as it seems, is to declare a map of all connections that may be activated, and to provide and maintain information about each individual connection.

21 The following is only a partial list of the operation types that can be declared. The purpose of this example is merely to show the range of applicability of the Data Projection Model.

Toward Auditable Information Models

The Data Projection Model encourages application developers and users to disclose the binary relationships that they are implicitly using, and make them available for interchange, interoperability, or auditing purposes. These relationships may come from a variety of sources, and many applications should be able to output them in the perspector notation. Since this is a very generic notation, systems using standard technologies such as XML or relational databases should be able to "export as perspectors" with a default configuration, once the feature is implemented for each type of information system. For specific applications, customized filters will be possible. This is similar to the export functions for XML or HTML in word processors, that can either use a standard, ready-to-go filter or let advanced users customize their own output.

However, not all data items must be handled this way. It is possible to choose, among a diversity of data items, which are those which deserve being treated as binary relationships, and therefore made auditable, and which should remain just isolated data, usually treated simply as property values for other data items.

The distinction between what needs to be handled as a full-fledge perspector and what remains a simple property is a domain that opens new modeling capabilities.

Auditability therefore is not some magical feature that would necessarily make everything transparent, no matter what. It is instead the result of a design decision as to whether particular data items should be made available for auditing.

Auditing is actually a specific rendering of information. The same techniques can be used to provide navigable information networks. The difference between the Data Projection Model and other approaches is its ability to provide multiple perspectives on the same data items.

Audit trails might be built of perspectors with operators such as Date of Creation, Date of Last Access, Creation Mechanism, Documentation, Links, etc.

Because of its basic character, the Data Projection Model is virtually already implemented in many products and systems, in particular those which use XML, or relational databases. Even if this is not the case, the disclosure of perspectors should not be a very difficult technical problem, provided access to internal information is permitted. The major problem users may encounter is the astronomical number of possible perspectors. Some work is likely to be needed to decide for each application what should be disclosed as perspectors. Furthermore, a whole series of new applications are conceivable that would be able to yield various kinds of useful information using perspectors as sources.

Future developments

Pre-defined operators in Perspectors

A number of applications could benefit by defining a fixed list of perspectors, of great general value, which could lead to useful applications of the Data Projection Model in the same way that HTML, considered as an instantiation of XML, is widely used and implemented inside the Web browsers. The list that follows is purely indicative, and it is left to the industry to define those that would make the most sense for general implementation. Types of perspectors of general value include:

- Arithmetic operations: Addition, Subtraction, Multiplication, Division, Percentage, Equality, Greater Than, Less Than, Greater Than or Equal, Less Than or Equal, etc.
- Logical operations: and, or, equal, exclusive or, negation, existence, null, etc.
- Set operations: belongs to, union, intersection, difference, empty set, etc.
- String operations: equality, containment, starts with, ends with, is null, concatenation, etc.
- Graph operations: node, edge, relation, etc.
- XML tree operators: root, child, parent, sibling, node type, generic identifier, attributes, attribute name, attribute value, content, processing instruction, comment, document type, etc.
- XML Schema operators: document type, encoding, version, root element, attribute declared value, etc.
- RDF and RDF-Schema operators: subject, object, predicate, about, etc.
- Dublin Core operators: author, title, contributor, coverage, date, description, etc. (More at http://dublincore.org/documents/dcmi-terms/)
- MARC operators: See http://www.loc.gov/marc/marcdocz.html for more information.
- Topic Maps operators: name, occurrence, association, scope, etc.
- Relational Database operators: field, record, field name, etc.
- Spreadsheet operators: average, sum, block, belongs to, cell number, etc.
- Time operators: hour, minute, second, universal time coordinates, time zone, etc.
- Etc.

Rules

Whether a language for representing rules with perspectors should be expressed in a standard notation or should be left up to applications to decide is still an open question. Wide adoption of perspectors might lead to another common layer for expressing rules. It is possible that one of the existing rules language, or one which is currently being elaborated, can be used within the context of perspectors, therefore avoiding the need to restart from scratch. For example, it should be possible to use XSLT to express rules. If perspectors are expressed using an XML notation, then existing tools should prove to be of great help.

References

- Ee-Peng Lim, Vladimir Cherkassky: "Semantic Networks and Associative Databases: Two Approaches to Knowledge Representation and Reasoning", *IEEE Intelligent Systems*, August 1992 (Vol. 7, No. 4) pp. 31-40. DOI Bookmark: http://doi.ieeecomputersociety.org/10.1109/64.153462
- Peter Pin-Shan Chen, "The Entity-Relationship Model – Toward a Unified View of Data", *ACM Transactions on Database Systems*, Vol. 1, No. 1, March 1976, Pages 9-36.
- UML (Unified Modeling Language): www.uml.org
- MOF (Meta-Object Facility): http://www.omg.org/technology/documents/formal/mof.htm
- XML (Extensible Markup Language): http://www.w3.org/XML/
- RDF (Resource Description Framework): http://www.w3.org/RDF/
- Formal Concept Analysis: "Applied Lattice Theory: Formal Concept Analysis", Bernhard Ganter, Rudolf Wille, http://www.math.tu-dresden.de/~ganter/fba.html.
- Topic Maps: www.isotopicmaps.org
- World Wide Web Consortium, *REC-DOM-Level-1-19981001*: http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/level-one-core.html
- Steven Newcomb and Patrick Durusau, see ongoing work on the Topic Maps Reference Model at: http://www.isotopicmaps.org
- The Uni-Level Description: A Uniform Framework for Representing Information in Multiple Data Models, Shawn Bowers and Lois Delcambre, Proc. of the 22nd International Conference on Conceptual Modeling (ER 2003), , October 2003, best paper nominee.
- ftp://ftp.cs.rutgers.edu/pub/borgida/cml-features.pdf December 1983, Features of Languages for the Development of Information Systems at the Conceptual Level, Alexander Borgida, Department of Computer Science, Rutgers University
- J. R. Abrial. Data semantics, in *Database Management Systems*, J.W. Klimbie and K. L. Koffeman, eds., North Holland, New York, 1974.
- D Shipman, *The Functional Data Model and the Data Language DAPLEX,* ACM TODS, Vol.6, No 1, March 1981, pp. 140-173
- Frequently Asked Questions for comp.lang.functional, Edited by Graham Hutton, University of Nottingham, Version of November 2002 (*no longer being updated*)
- Binary Relations Approach, Andrew Girow, Copyright © 1996 Andrew Girow. All Rights Reserved.

Annex: Perspector Algebra

Once information processes have been disclosed and decomposed as perspectors, what is left is an ocean of binary relations, all looking similar, and in vast numbers. It would be useless to have all this information available without ways to rebuild it and view it in ways that are most relevant to the particular perspective(s) needed. Views result from queries performed on these information items. Queries can be used to collect information items that are seen in a given perspective.

Here is a proposed syntax for query on perspectors:

List of perspectors

< | *o* | >	All operations with one given operator *o*
< *x* | *o* | >	All operations performed with operator *o* on operand in position *x*
< | *o* | y >	All operations performed with operator o on operand in position *y*
< *o* | *xy* >	All operations performed with operator o on operand in either position *x* or *y*
< *x* | *o* | *y* >	All operations performed with operator *o* on operand *x* and operand *y*
< *xy* >	All operations performed with any operator on operand in either position *x* or *y*
< *x* | | >	All operations performed with any operator on operand in position *x*
< | | *y* >	All operations performed with any operator on operand in position *y*
< *x* | | *y* >	All operations performed with any operator on *x* and *y*
< | | >	Everything
< >	Nothing

Lists of strings

< *x* | * | >	List of all operators that have *x* as their x-operand.
< | * | *y* >	List of all operators that have *y* as their y-operand.
< * | *o* | >	List of all x-operands that have *o* as their operator
< | *o* | * >	List of all y-operands that have *o* as their operator
< *x* | * | *y* >	List of all operators that have *x* as their x-operand and *y* as their y-operand