Semantic Heterogeneity Among Document Encoding Schemes

Final Report for NIST Federal Assistance Contract 60NANB0D0115

David Durand and Paul Caton
Scholarly Technology Group
Brown University
{dgd, paul}@stg.brown.edu
January 15, 2002

About this document

This document reports on research conducted at the Scholarly Technology Group (STG), Brown University, in 2001; Dr. Allen Renear, Principal Investigator. A revised presentation of these results is being submitted for publication to *Markup Languages: Theory and Practice*, MIT Press. This paper will also be published as a technical report on the STG web site, and we expect to present work deriving from it at conferences such as Extreme Markup Languages and the Joint International Conference of the Association for Literary and Linguistic Computing & the Association for Computers and the Humanities.

Acknowledgements: This research was made possible by support from the National Institute for Standards and Technology [60NANB0D0115]. We gratefully acknowledge the special help of Dr. Victor McCrary, Chief, Convergent Information Systems Division.

The Brown University Scholarly Technology Group (STG) is an applied research and development group specializing in electronic document design and markup systems.

Contact: Elli Mylonas, Associate Director
Scholarly Technology Group
Brown University, Box 1841
Providence, RI 02912
v: 401-863-7231 f: 401-863-9313 e: elli_mylonas@brown.edu
w: http://www.stg.brown.edu

Table of Contents

Executive summary

Semantic heterogeneity in document encoding systems is a serious obstacle to the interoperability required to create a critical mass of content for the electronic publishing industry. This is a problem which persists even after a common syntax (e.g. XML) has been adopted, and sometimes even when common vocabularies are used. To address this problem the Scholarly Technology Group (STG) at Brown University undertook a project, supported by the National Institute of Standards and Technology, to develop a framework for analyzing the fundamental features of semantic heterogeneity in document encoding systems. This is a first step in developing tools and techniques for resolving the difficulties semantic heterogeneity creates for the electronic publishing industry in general, and for the ebook industry in particular. The project draws on ongoing related work at STG, particularly in the XHub project, an XSLT-based system for creating OEB ebooks from multiple XML vocabularies, and in other encoding projects, such as the Women Writers Project.

This paper contributes to the understanding of this problem at two levels of abstraction. At a high level of abstraction, we present and justify a theoretical framework within which semantic heterogeneity can be analyzed operationally in terms of its effect on the ability to construct transformations between encoding schemes. This theoretical framework justifies the common practice of document conversion, and provides clear meanings for a number of undefined terms in common use by practitioners. The practical consequence of this theoretical framework is to simplify the analysis of the effects of semantic heterogeneity on the unbounded set of uses for electronic documents. Whether one is concerned with cross-server metadata search, natural language analysis, screen layout, or typesetting, document transformation represents a well-understood and defined framework for analyzing the problem.

At a lower level of abstraction, and closer to practice, we propose a preliminary taxonomy of specific textual problems that create difficulties in the creation of transformations, and relate that taxonomy to some of the work already done in database theory. This taxonomy is derived from examples drawn from the examination of several DTDs for scholarly texts, ranging from the Text Encoding Initiative's DTD for the description of humanities texts, to the W3C's HTML 4.01 (a subset of which forms the core of the Open eBook DTD), to a major publisher's DTD for a scientific encyclopedia. While we have found many papers discussing tree transformation from a technical perspective, and advancing technological proposals for defining document transformations, we have not found discussions of the kinds of document encoding problem that make transformations necessary. We hope that our preliminary taxonomy and examples are a foundation for understanding the representation and schema problems that arise in multi-DTD environments.

At this project's inception we anticipated the need for a principled comparison of DTDs and an analysis of their differences and the resulting problems. A surprising result, though in hindsight it seems obvious, is that document transformation can be regarded as more than an *ad hoc* tool for solving practical text encoding problems. In fact, document transformation provides a unified perspective that subsumes a variety of seemingly disparate concerns. Throughout, we attempt to provide some pointers into the larger literature on databases, specific document conversion languages, and web technologies.

In the future we intend to examine the specifics of document transformation languages with respect to our provisional taxonomy of types of semantic heterogeneity, in confidence that document

transformation is a fundamental model for addressing the still incompletely understood problem of semantic heterogeneity in documents. Document transformation is fundamental to text processing, not just a convenient tool.

Researchers

Dr. David Durand, Chief Scientist, Scholarly Technology Group, and Adjunct Associate Professor of Computer Science, Brown University. An internationally recognized expert in SGML, XML, and HyTime, David Durand has been a leading participant in the text encoding world since 1990. As an invited expert, he worked with the W3C in the creation of XML, and has been a member of the W3C working groups that developed the related standards XPointer, XLink, and XML Base. His research brings together interests and expertise in SGML/XML, the Text Encoding Initiative (TEI), the W3C, linguistic analysis, and computer-supported collaborative work (CSCW), which was the focus of his doctoral research. His extensive and influential publications include a guide to HyTime, the standard for multimedia markup in SGML, which he co-authored with Steven DeRose. David Durand has worked for the past ten years in a variety of SGML consulting and research positions, including seven years at Dynamic Diagrams (three years as CTO), a Providence-based company specializing in SGML publication software and consultancy.

Paul Caton, Project Analyst, Scholarly Technology Group, and Electronic Publications Editor, Women Writers Project, Brown University. A specialist in client/server XML publishing, Paul Caton works extensively with CSS, XSLT, and the TEI and other TEI-related element sets. In 2000 he was a member of the DTD Working Group for the Early English Books Online project. He is completing a doctoral dissertation that focuses on problems in the theory and practice of text encoding. He has published and given conference presentations on markup theory, document processing, and other encoding-related issues. Paul Caton has worked for the last five years at the Women Writers Project and the Scholarly Technology Group at Brown University.

1 Introduction

This paper considers the problem of semantic heterogeneity in content markup systems such as SGML/XML's document type definitions (DTDs) or newer systems like XML Schemas or Relax NG. While our discussion is framed in terms of XML alone, the issues discussed here are the same for all content markup systems. We have based significant parts of our model on the analysis of semantic heterogeneity in databases. However, XML documents, schemas, and processing differ from database data, schemas, and processing in a variety of significant ways. In this introduction we create a framework within which the practical problems posed by semantic heterogeneity in XML schemas (of all sorts) can be examined, and within which potential solutions to these problems can be judged.

1.1 Semantics

The term "semantics" is often confusing, because it has so many senses. We consider four distinct senses that are relevant to markup languages. One sense is *operational semantics* as used in the description of programming languages. An operational semantics for a language is defined by what a sentence in that

language will do, or what a particular family of programs will do with that sentence as input. For relational databases, the operational semantics of SQL defines many aspects of database system behavior. *Denotational semantics* is a precise mathematical definition of the objects and relations of a language in which each sentence of the language names, or denotes, a mathematical object, such as a function. The relational algebra is the denotational semantics of relational database queries and tables, describing the structure and query operations of an ideal database as a mathematical system. The third meaning of semantics is its loose ordinary language sense, in which the semantics of a statement is its "meaning." We call this sense *natural semantics*. The natural semantics of a relational database is the relationship between its tables and fields and the real world entities that it models. Defining natural semantics and database schemas to represent them is the task of database designers and these semantics are documented at levels of formality ranging from prose descriptions to entity-relation diagrams. When people say that an XML document represents the meaning of document features, rather than their formatting, they are generally appealing to some form of natural semantics. This latter notion, because of its inherent vagueness, has been the subject of many attempts at closer definition by the application of logical methods and axioms, and we will use the term *logicist semantics* to refer to formal models that attempt to represent the natural semantics of some external domain. Logicist semantics, strictly speaking, is a variety of denotational semantics, but is distinguished by using logical representations that include constants that refer to non-mathematical primitive entities like people, part numbers, and paragraphs. The tools of logicist semantics are essentially the tools of philosophy and the logicist paradigm in AI.

The semantic web is largely concerned with enabling the collaborative creation of a logicist semantics of the contents of web-accessible documents and data. Thus, the semantic web is creating logicist semantic models for biological information, for commerce, and many other subject domains that people document in web pages. In this paper, we are not directly concerned with the semantics of what documents are about, but rather with the semantics of their structural components. It is clear that most documents share a variety of common structural features, from chapters, lists and illustrations, to lemmas, theorems, and paragraphs. While many documents share these kinds of common features, the particular sets of features and their interrelationships vary greatly from document to document, and from document type to document type. Despite many attempts there has been little progress in creating a useful language that can describe the structures of all documents faithfully. The issue of semantic incompatibility between document schemas such as DTDs arises from the structure of the documents, and not their contents. To some extent the distinction between structure and contents is a difference in emphasis, as document structures are obviously constrained by the need to accurately represent their contents. There is an obvious interaction and dependency between the description of the contents of a document and the structure of that document as represented by its decomposition into an XML element structure.

1.2 XML semantics and document transformation

The biggest and most obvious difference between databases and XML documents is the form of the data itself. Most modern databases are based on the relational model. A relation is commonly figured as a table where the table columns represent the relation domains and each table row represents one *n*-tuple of the relation. Object or data identity is not a primitive notion in databases at all, but is synthesized, when needed, by the imposition of uniqueness and key constraints on the values stored in particular columns, and by the modeling notions of entities and relations.

In an XML document, by contrast, object identity for XML elements is an essential component of document representations, albeit one usually unmarked by an explicit key. Even more importantly, identity in a document is associated with the placement of each datum in a particular order (and hierarchical relationship) with respect to other data in that document. While the elements and attributes defined in a DTD or other schema are in many ways similar to column types in a relation, their data values may include arbitrary substructures of elements, as allowed by that schema. The data values in XML documents are not simple values, but are composite objects whose subcomponents also have unique identities.

These differences are significant, but they are not the most fundamental ones. There is a larger difference between XML systems and database systems, which goes well beyond the differences in the kinds of data managed by these systems. This difference is in the processing models involved in processing these different kinds of data. Database systems are essentially oriented to the problems of updating, querying, and reporting (which is a special type of querying). These problems have been further broken down into the primitives of the relational algebra, and given a precise mathematical meaning.

The operational semantics of XML applications is not nearly so universal, as one of the fundamental ideas of content markup is to abstract data so that the widest variety of sensible sorts of document processing will be possible. Partly because of this emphasis, and the fact that special representations are often needed to control document processing for a specific task, like typesetting, eBook publishing, or web publishing, the most typical processing operation on XML documents is conversion to another format, for processing with another tool. Such a conversion is typically designed to create a customized presentation of all or part of the document's content. Most of these transformations are created with relatively powerful tools that allow any Turing-computable transformation to be created.

The notion of federated databases using the relational model is a concrete operational goal. This allows a meaningful evaluation of the consequences of semantic heterogeneity and benchmarking of the severity of various forms of heterogeneity. The problem of semantic incompatibility for federated databases is the problem of how to deal with differences between database schemas for the same application domain that affect realizability of this primary goal: that a federated database should process all queries against several databases as if there were a single, consistent database containing all the data in the component systems. When analyzing semantic heterogeneity, only those differences between schemas that require compensating action, or that prevent this goal, are important.

In order to evaluate incompatibilities between XML schemas, a similar operational scenario is needed. The widely prevalent use of document transformation in practical text-encoding work is an indicator that document transformation is at least one important application in this area. However, document conversion between schemas is a single problem that can model all of the ways that semantic incompatibility might cause problems for markup-processing applications. This is a direct consequence of the fact that a conversion function can be used to adapt one system of semantic encoding to another one. No matter what differences may exist between a pair of document processing applications, if the data suitable for one system can be converted into a form that is suitable for the other system, than that conversion function can serve as an "adapter," enabling the same data to function in both applications. We will consider the question of data schemas in terms of the existence and structure of computable functions transforming instances of one schema into another.

Changing a problem about the structure of data objects into one about arbitrary computable transformations does not constrain the problem very much (actually it seems to make it more difficult), but it does reflect the broadest possible construal of the problem. A similar thing is also done in the database domain; while database functionality is defined by what operations are permitted by the relational algebra, such a theory does not mean that the creators of database applications do not find it necessary to use general-purpose computer languages. Database theory simply considers a portion of the problem of database application design, while acknowledging the incompleteness of the theory with respect to other important problems. Because databases are now well understood, the boundaries between what a database system will do, and what requires a general-purpose computer language, are also well understood.

We examine the transformation problem in general terms first, and later break that process into parts, such that the need for generalized computation is isolated. This will mean that many transformation problems and ideas can be described independently of the need to consider all of the complex features of general computation (such as non-termination). In the following discussions we will use the term "function" to mean an effectively computable function in some transformation language. Conversion functions that are well-defined but not computable are not of interest.

Because there is not yet wide agreement about the minimal set of operations that constitute a useful transformation language for XML, we phrase our definitions in a form that is independent of the particular transformation language. In our examples and discussions we assume the use of a tree transformation language like XSLT. While XSLT is not perfect, it seems not only sufficient (it is Turing-complete, after all), but natural for expressing XML transformations. Most transformations, furthermore, do not require extensive use of the computational facilities of XSLT, but rather rely on its flexible pattern-matching and substitution facilities.

1.3 Semantic compatibility

In this section, we define semantic compatibility of XML schemas by increments, approaching the final definition by imposing a series of progressively stronger conditions which reflect theoretical and practical concerns. We start with the intuitive operational notion that a transformation is a program in a transformation language that reads one XML document and creates another XML document. We further limit the discussion to the level of the XML information set [XMLIS] (essentially the full parse tree produced by an XML parser). This simplification is warranted because the semantic issues are not generally affected by the practically important but incidental details of file format, encoding and organization. Further, we can ignore a good deal of information within the XML information set, without loss of generality.

The XML information set defines the information contained in an XML document that should be considered important to most processing applications [the reason for the word *most* is that some applications, like XML-aware editors, may need to treat an XML document as a string of bytes and not just as a structured object]. An XML information set is an ordered tree structure of nodes of specified types. We regard the XML information set (the abstract syntax of XML) as the denotational semantics of an XML document. The most important structures in this tree (and the ones that are controlled by XML schema languages) are elements, attributes, and textual values. While specialized nodes exist to represent

information like character encodings, namespace declarations, and the like, we will either ignore such nodes as unimportant (like character encodings), or treat them as properties of the nodes to which they are attached (like namespace declarations). For the rest of this section, the term "document" will be used as a synonym for "information set corresponding to a document."

Just as the particular character encoding used to represent Unicode characters is unimportant to the semantics of XML documents, we will generally find the distinction between attributes and elements semantically unimportant. We will treat attributes as elements that can contain only character data, and that are unordered with respect to other children of their parent element. This means that if two schemas represent the same information as an element in one schema, and an attribute in the other, there will be a semantic incompatibility only if the element-using schema marks significant information by that element's position relative to other elements, or if that element contains non-character data. While this is a real source of heterogeneity, this is a special case of the kinds of heterogeneity that already occur between elements with different content models. In other words, while the differences between elements and attributes can introduce incompatibilities, they are instances of the same kinds of heterogeneity as occur in a model with elements alone.

Denotationally, a transformation is simply a (computable) function from documents to documents. Furthermore, for any transformation we will restrict the domain of the function to those documents corresponding to a particular schema, and require that it be defined for every document conforming to that schema. We will use functional notation to represent document transformations, writing $f: A \rightarrow B$ to represent a transformation f taking input documents conforming to schema A and producing output documents conforming to schema B. Denotationally, a transformation f is a (computable, total) function whose domain is the set of documents conforming to schema A and whose range is B. We sometimes use the term *input schema* to refer to a schema that defines the domain of a function, and the term *output schema* to refer to a schema for the range of the function.

The first stage of defining semantic compatibility is to consider transformations that are *information-preserving* in the following sense:

Def: A transformation f meets the round-trip criterion if there exists an *inverse* transformation f^{-1} such that $f^{-1}(f(x)) = x$ for all documents x in the domain of f, and where = represents equivalence of XML information sets.

Def: A transformation f is an *information-preserving transformation* iff it meets the round-trip criterion.

Note that while an information preserving transformation is defined by a total function and its inverse, the existence of a lossless transformation $f: A \rightarrow B$ does not imply the existence of a lossless transformation $g': B \rightarrow A$. An information preserving transformation defines a 1-1 correspondence between every document conforming to its input schema and some documents conforming to its output schema. The domain of the inverse function f^{-1} may be a proper subset of B and its results may be undefined if applied to members of B that are not in the range of f or that they may not conform to the schema A. Even if f^{-1} is defined for all members of A, $f(f^{-1}(x)) = x$ may not hold. Intuitively speaking, the presence in B of exact equivalents for every element of A does not mean that B itself might not contain elements that cannot be properly represented in A.

Def: a schema *A* is richer than a schema *B* iff there is an information-preserving transformation from *A* to *B* but there is none from *B* to *A*.

We will consider semantic compatibility as a binary relation between schemas, and the existence of an information-preserving transformation from *A* to *B* to be a necessary condition for saying that *A* is semantically compatible with *B*. This relation of semantic compatibility is not be symmetrical between two schemas, because a schema may be semantically compatible with a richer schema, but the inverse will not be true.

We might be tempted to say that a schema *A* is semantically compatible with a schema *B* if and only if there exists an information-preserving transformation that can transform any instance of *A* into an instance of *B*. This does not work, because being information-preserving is not a sufficient condition for a transformation to usefully preserve semantics. Some information-preserving transformations do not make proper use of the document elements provided by the destination schema. A simple example would be an information preserving transformation that consistently misapplies markup in the destination schema, but in a reversible way. For instance, a transformation that interchanged author names and titles in bibliography entries would meet the round trip criterion, but would be misapplying the element types provided in the destination schema. More subtle variations of this kind of problem occur frequently enough in actual practice that the term "tag abuse" is in common use to denote the situation where a tag consistently used to represent a meaning different from that intended by the creator of the schema.

This is a more general problem inherent in any data representation, since the relation between the data and what it represents lies outside the formal system underlying the data itself. For instance, while the relational algebra constrains the operations of database systems, and provides a semantics that can be used in implementing software, the correct modeling of a real data management problem as a relational schema is a task that requires an analysis that includes factors outside the database itself.

This boundary between a model and the phenomenon that it models is inherent. One way to think about the semantic web is to consider that a successful semantic web would use ontology and deduction mechanisms to help push this boundary back for web and XML-based information systems, by providing more complete metadata about the contents of web documents. The work in Sperberg-McQueen, Huitfeld, and Renear [SHR00] attempts to do the same thing for structural characteristics of documents, and not their content.

We will use the term *external model*, to represent the non-formal equivalence relationships between external objects or concepts and the document element types defined by a schema. The external model for a schema is the natural semantics of the elements in that schema. We call a document model correct when the relationships correspond the ones intended by the creator of the schema. The correctness of a document model is a notion that cannot currently be formalized, although the issue of defining useful external models is addressed in the discipline of DTD analysis (as described, for instance, in Maler and El-Andaloussi [ME96]).

Def: An information-preserving transformation is *faithful* if the result of applying the transformation to a document with a correct external model for the input schema is any document that has a correct external model according to the output schema.

We are now able to define semantic compatibility between two schemas.

Def: A schema *A* is semantically compatible with a schema *B* if and only if there is a faithful information-preserving transformation from *A* to *B*.

Def: A *perfect* transformation is a faithful transformation from *A* to *B* for which there is an inverse faithful transformation from *B* to *A*.

If there is a perfect transformation from schema *A* to schema *B*, then we can say that they are semantically equivalent, since documents in either one can be translated to the other, without loss of information, or the introduction of defects into the external model.

1.4 Document transformation

While transformation languages are generally Turing-complete, the full power of arbitrary computation is rarely used. On that account, we will use a conceptual transformation framework that isolates the occurrence of arbitrary computation. Furthermore, transformation processes generally have a compositional structure that is based on the structure of the input, and is ultimately reflected in the output. This is a direct consequence of the hierarchical nature of XML data, and the fact that a perfect, faithful transformation creates a one-to-one correspondence between elements of the source schema and some subset of the elements in the destination schema. While transformation languages are generally Turing-complete, the full power of arbitrary computation is quite rarely used. On that account, we will use a conceptual transformation framework that limits the role of arbitrary computation to a minimum.

The process of transformation can be divided into two phases, input, and output. Each phase is characterized by a significant question. In the input phase, the question is "what have we got?" The goal of this phase is to precisely identify all significant data objects within a document instance, each correctly interpreted in context. In the output phase, the question is "where do we put it?" The goal is to take some or all of the information objects identified in the input phase, and place them in appropriate contexts within the output document so that the result will have a correct external model. Context is an important factor because document components all have identity and are related to each other. The relationships that define the context of an element are its position in the document tree: its containment relationships with other elements, and its sequence relationships. These relationships often imply additional relationships in the external model. A simple and common case is the use of "inherited attributes" to represent things such as the language of a text. While the XML Information set does not include such a concept, definers of schemas often create attributes like this. When this is done, the presence of an appropriate attribute on an element will affect all elements below it in the document hierarchy. Furthermore, even simple context-dependencies like this may have exceptions, if for example, there is a rule in the external model that says that all reader's notes are in English, and in consequence language attributes should not be inherited by notes.

The output phase follows the answering of the question "What have we got?" and is characterized by the fundamental question "Where shall we put it?" In this phase a new document (information set) is constructed. While transformations may introduce new data or elements that did not exist in the input, and may delete data or elements that did, the greatest part of the content in a transformed document is derived from the original.

This two-phase processing model corresponds to a simple implementation technique in which pattern matching selects data items and sets of items, and then separately constructs a new structure containing data drawn from the results of the pattern matching. This is, for example, the core model used by the XSLT transformation language.

We can extend this model by recognizing that arbitrary computation is sometimes needed. Without loss of generality, we will treat all arbitrary computation steps as filters that are executed between input pattern matching on some data item, and output pattern construction of the result. The functional nature of XSLT means that in fact most of general computational mechanisms, like conditional execution, looping, and the like, actually match this model directly. For many stylesheets, this abstraction is very to close to the concrete implementations. Of course, XSLT also provides features like file inclusion and the recursive invocation of processing modules. While transformations that use these features can still be modeled, the models are more abstract.

Modeling transformations in this way helps to expose some aspects of their structure; the constraint on where computation is performed is an illusory one — any arbitrary computation can be trivially represented as a pattern that matches the entire document, passes it as input to the program in question, and generates a single output (the result).

Fundamental to the model are the following two types of computation:

- on input, to parse or otherwise break apart the contents of an element into several data objects, by means of an arbitrary computation;
- on output, to collapse several data items into a single element, by means of an arbitrary computation.

When the only intermediate function required to define a transformation in these terms is the identity function, we have a limited form of heterogeneity that we can call "pure markup" heterogeneity. This is an interesting case, because the computation required in a transformation like this is limited to identifying data objects and copying them into the result structure to define a transformation. Interestingly, we can allow considerably more power into conversions (and incidentally come closer to the facilities of XSLT) without becoming Turing-complete, by allowing the additional functions and their compositions (the following examples are meant to indicate how powerful such a paradigm can be even for transformation programs that are guaranteed to terminate):

- conditionals that evaluate terminating conditions and select alternative outputs;
- bounded loops in the form of mapping functions that apply another function to each item of an element, and produce a new element containing the results;
- data pattern matching functions that turn, say regular expression occurrences in character data into element sequences;
- functions that sort elements.

All of these capabilities exist in XSLT, but in a form that is much closer to general programming languages. We have been struck at how rarely unbounded computation is needed in transformation.

2 Related work

2.1 Database heterogeneity

An extensive literature exists on issues of heterogeneity among databases. A problem is that not all authors use the same term to mean the same thing. For example, [Hul97] and [HAK01] both have "semantic heterogeneity" in their titles, but [Hul97] uses it inclusively of data ("heterogeneity in the logical representation of data, including different data models, different schemas, and different kinds of data") while [HAK01] uses it exclusively ("data heterogeneity and semantic heterogeneity"). Several papers propose taxonomies of the possible types of heterogeneity, though none has become definitive. [KS91a] describes a taxonomy based on the relational database table-and-attribute(s) model. The authors make a basic distinction between *schema conflicts*, where different databases specify their data structures differently, and *data conflicts*, where data differ despite identical specified structures. [GSC96] and [Duw97] expand the [KS91a] taxonomy to include conflict types associated with the class hierarchies of object-oriented data models. Although some work like [Duw97] specifically reuses and extends earlier work, other work continues to propose classifications from scratch [EWMM00].

[Col97] distinguishes *structural* from *fundamental* semantic heterogeneity. The former "occurs when the same information occurs in two different databases in structurally different but formally equivalent ways" and it "can be resolved essentially by a series of (possibly complex and subtle) view definitions which in principle respect the autonomy of the component information systems." Fundamental semantic heterogeneity, on the other hand, "cannot be resolved by view definitions, even in principle" and "[where] it occurs between information systems, tight coupling cannot be achieved without changing at least one of the systems, thereby compromising design autonomy". Database views are a very close equivalent to perfect document transformations, since they can change table definitions, but must do so in a way that allows updates to a view to be automatically translated back into equivalent updates on the base schema.

The resulting situation for document schema heterogeneity is in some ways similar, because analyzing the causes and types of fundamental heterogeneity is useful to the practitioner, but will not be completely resolvable by any automated means. Cases of structural heterogeneity can be resolved by creating an appropriate transformation, which, like a database view, can then be regarded as solving the problem. This analogy is not perfect, however, because faithful transformations often suffice even if they are not perfect. This is a consequence of a processing model that is less constrained than a database view, which must allow updates because of the purpose of a DBMS. Updates in documents are relatively infrequent and can often be controlled. In many document-processing situations, there is a single "master copy" encoding according to some XML schema which can be the site of all changes. All other applications and their schemas are consumers of data from that master copy, and thus the existence of faithful transformations from the master schema to the others is sufficient to resolve the heterogeneity.

Most of the literature discusses structural semantic heterogeneity and concerns itself with ways of reconciling discrepant views. [Mil98] argues that "[d]espite its prevalence, and despite the plethora of work enumerating and categorizing types of schematic heterogeneity, no systematic study of how [sic]this schematically heterogeneous structures can be used and queried in practice has been undertaken." The author uses the SchemaSQL language to "propose a solution in which all schematic heterogeneity is reconciled by a limited form of higher order views called *dynamic views*". In [SCI94] the

authors define semantic value as datum+context information (i.e. metadata about the datum). They express the semantic value in the LISP-like form "value(context_information);" for example, "4(LengthUnit = 'feet')." Conversion functions can then convert between equivalent semantic values. [DBK98] does not use the term "heterogeneity" but does address relevant problems, with a slightly different take from other papers: "[it] is clear that there may be many transformations, with differing semantics, corresponding to the same schema manipulation, and that it is necessary to be able to distinguish between them. In contrast to existing work, our focus in this paper is therefore on how transformations effect the underlying data itself. We will use the term 'database transformations', as opposed to the more common 'schema transformations', in order to emphasize this distinction." (56).

In the case of fundamental semantic heterogeneity, where "neither database contains sufficient information to resolve the [semantic] differences" ([Col97]), one approach is to sacrifice component database autonomy and change one or more of the component schemas [GSC96]. An alternative approach, as [Col97] describes, is to make databases which need to interoperate be members of a "standard language community", where each member must declare the semantic relations between its own schema and the language community's global schema.

[KKR01] mentions heterogeneity in the context of integrating XML and relational database systems. They describe the "X-Ray" system in which users first define the mappings they want between an XML encoding model (or a particular XML document) and a relational schema. This mapping knowledge is stored as a generic meta schema which "mediates between heterogeneous concepts and provides the basis to automatically compose XML documents out of the relational database when requested and decompose them when they have to be stored." The paper does not present details of the mediation itself.

The database literature provides a solid and very useful foundation for discussing the problem of semantic heterogeneity in content markup systems. A taxonomy like Kim and Seo's [KS91a] describes many of the types of heterogeneity germane to markup. Nevertheless it is insufficient for our purposes because XML encoding models have the significant added feature of hierarchical nesting. Even taxonomies based on hierarchical data models (e.g. [Duw97]) still assume database-like data, i.e. discrete data that lend themselves to re-arrangement, rather than data as components of an overall structured object.

2.2 Document structure transformations

Parse tree transformations are central to resolving structural heterogeneity, and an extensive literature discusses their formal properties and their implementation. [AQR97] gives a formal analysis of type conversion problems that such transformations involve, and notes the difference in user intentions between *static* (in essence, schema-to-schema) transformations and *dynamic* transformations (of the kind associated with interactive editing sessions — e.g. a cut-and-paste). Numerous syntax-directed translation schemas and tree-transformation-based languages have been described, e.g. SIMON [FW93], Scrimshaw [Arn93], ALCHEMIST [TL94], SYNDOC [KPV94], TranSID [JKL97], DSSSL [ISO91], XDuce [HP00], HARMONIA [Bos01]. The best known and most widely adopted document transformation language is XSLT (XSL Transformations) [W3C99]. [JKS00] assesses XSLT's ability to resolve different types of schema heterogeneity (primarily structural semantic and data-related). Specifically, where "B" is a source database and "M" is a target relation, the authors describe four categories of heterogeneity:

- *crossover schema mismatches*: **Type 1**, "when a concept is represented as data in 'M' but as relations in 'B'"; **Type 2**, "when a concept is represented as data in 'M' but as attributes in 'B'"
- *domain structural mismatches*: "when a domain 'M' corresponds to a domain with a different data type or several data domains in 'B'"
- *domain unit mismatches*: "when a domain 'M' assumes different units of measurements from the corresponding data domains in 'B'"
- *domain population mismatches*: "when a domain 'M' assumes different population from the corresponding data domains in 'B'" (e.g. if 'M' and 'B' are company databases, the population of the domain "Jobs" in one may be different from the population in the other)

The authors show that XSLT can resolve each of these heterogeneity types.

Some commercial content-manipulation and delivery products ([Omn] [Bal]) can perform structural transformations. These are industry tools built to solve the transformation problem but without forming any particular theory about it.

3 Examples of heterogeneity among five DTDs

3.1 Fundamental causes of heterogeneity

XML encoding schemes model data objects for document types, usually objects we might think of as *textual data structures*, such as headings, lists, paragraphs, tables of contents, etc. In display these features commonly have distinctive formatting; they are familiar to all readers and the general nature of each (its logical structure and communicative function) is well understood. However, there is no common understanding of their *particulars* and hence no authoritative specification of those particulars. Just as different printers format these textual data structures differently among different documents, so encoding scheme designers specify them differently.

Encoding schemes are fundamentally needs-driven. They do not attempt to express the universal truth of documents or create an ur-language that perfectly represents all possible document (and document component) types. Indeed, the few attempts to create universal document models have failed. Rather, each encoding scheme reflects the particular needs of a particular user constituency. Generalizing, users' needs fall somewhere along three axes: prescription/description; specialization/generalization; less choice/more choice. (*Pre*scriptive schemes predominantly guide the creation of new documents whereas *de*scriptive schemes are used to encode legacy documents.) Where users' needs fall in relation to these axes profoundly affects encoding scheme design and hence inter-scheme heterogeneity. While there is no *necessary* connection among the first terms of the three pairs, prescriptive schemes tend to be specialized to a field, an organization within that field, or even to one of the organization's products. As specialization increases and the potential for variation decreases, the scheme imposes stronger semantic and structural constraints. Element names become more specific, content models allow fewer possible encodings, and the likelihood increases that the scheme is designed with specific downstream processing in mind. This can equally be said of narrowly-specialized descriptive schemes, but as the number of document types that a descriptive scheme wants to adequately represent increases, so either the number

of elements increases or the generic identifiers become more general (or both), the content models become more flexible, and the encoding has less or zero relation to a specific application.

3.2 Sample encoding schemes

In sections 3.3 to 3.6 following we consider different types of textual function — bibliographic citation, annotation, linking, and high-level structural division — that have generally-agreed-upon textual data structures associated with them (citations, notes, chapters, etc.) and with reference to a selection of actual schemas we show the heterogeneity that can exist among different schema models of the same basic structures.

We include HTML 4.01 as the first example schema because it is a widely-used DTD, probably better-known than its XML-conformant counterpart XHTML 1.0, and, in the immediate future at least, likely to be either the source or target of numerous schema-to-schema transformations. The TEI Lite (XML) DTD comprises a subset of the much bigger and more complex full TEI DTD. TEI Lite is intended to be appropriate for encoding a wide variety of humanities document types and is popular among academic encoding projects. Our third sample schema is a DTD designed for papers submitted to the Extreme Markup Languages conference originally sponsored by the Graphical Communication Association (hence we refer to it here as the GCA DTD). DocBook is a very well-known and widely-used DTD for encoding printed books, articles, documentation, manuals, etc. Here we refer to version 3.1; documents conforming to this DTD usually require only minor changes to become valid XML documents. The final example schema is a DTD fragment used by publisher McGraw-Hill for articles in their *Encyclopedia of Science and Technology*.

The encoding examples try to be accurate but not comprehensive. That is, we do not give examples of *all* the possible ways each schema allows you to encode a particular textual feature.

3.3 Bibliographic markup

Documents commonly contain bibliographic information about other, external documents (printed or electronic) and each of the five example text encoding models provides for it to a degree consonant with their target document types.

- HTML 4.01 has a single phrase-level element, <cite>, with no specific provision for identifying components of the citation (author's name, document title, etc) through either child elements or attributes. <cite> can appear throughout an HTML document.

  Example:

  ```
  <cite>Renear, Allen, and Elli Mylonas, David Durand. 1996. "Refining Our
  Notion of What Text Really Is." In Nancy Ide and Susan Hockey (eds.),
  <em>Research in Humanities Computing</em> 4. Oxford: Clarendon
  Press.</cite>
  ```

- The TEI Lite XML DTD offers two options: either a loosely-structured citation using <bibl>, or a fully structured citation using <biblFull>. The first can be either just #PCDATA or an unstructured mix of #PCDATA and bibliography specific elements such as <author>, <editor>, <publisher>, etc. The

second option, <biblFull>, is a near-duplicate of <fileDesc>, a TEI metadata element, and allows for a detailed, structurally-constrained citation. Both <bibl>and <biblFull> can function as either block-level or phrase-level elements, and can appear in any of the major structural divisions of the document.

Example:

```
<bibl><author>Renear, Allen, and Elli Mylonas, David Durand</author>.
<date>1996</date>. "<title>Refining Our Notion of What Text Really
Is</title>." In <editor>Nancy Ide and Susan Hockey<editor> (eds.),
<title>Research in Humanities Computing</title> 4.
<pubPlace>Oxford</pubPlace>: <publisher>Clarendon
Press</publisher>.</bibl>
```

· Under the GCA conference paper DTD bibliographic citations appear as <bibitem>s in a <bibliog> which has to appear in <rear>. In the body of the text the citations are referenced by <bibref refloc=""/> elements where the value of refloc= is an IDREF that matches the ID value of the appropriate <bibitem>. Within <bibitem>, <bib> contains the expanded reference form (eg. "SJ90" goes to Smith and Jones 1990) and <pub> contains all the publication details. However, <pub> can only contain the emphasis elements <highlight>, <sub>, and <super>, and does not have granular tags for author, title, publisher, place of publication, etc.

Example:

```
<bibref refloc="RMD96"/>
...
<bibliog>
<bibitem id="RMD96">
<bib>Renear, Mylonas, and Durand 1996</bib> <pub>Renear, Allen, and
Elli Mylonas, David Durand. 1996. "Refining Our Notion of What Text
Really Is." In Nancy Ide and Susan Hockey (eds.), <highlight
style="ital">Research in Humanities Computing</highlight> 4. Oxford:
Clarendon Press.</pub>
</bibitem>
</bibliog>
```

· In the DocBook encoding model all full bibliographic citations are contained by a <bibliography> parent. Below that there is an optional subdividing element <bibliodiv> for the subdivision of citations into books, journal articles, online resources, etc. At the citation level there are two options. One, <biblioentry>, has only element children and so resembles the usage of TEI Lite's <bibl> with no #PCDATA in the content. The second option, <bibliomixed> has a mixed-content model to allow for formatting "as is."

Example:

```
<bibliography>
<biblioentry>
<abbrev>RMD96</abbrev>
<authorgroup>
```

```
<author><firstname>Allen</firstname><surname>Renear</surname></author>
<author><firstname>Elli</firstname><surname>Mylonas</surname></author>
<author><firstname>David</firstname><surname>Durand</surname></author>
</authorgroup>
<pubdate>1996</pubdate>
<title>Refining Our Notion of What Text Really Is</title>
<editor><firstname>Nancy</firstname><surname>Ide</surname></editor>
<editor><firstname>Susan</firstname><surname>Hockey</surname></editor>
<title>Research in Humanities Computing</title><volumenum>4</volumenum>
<publisher><publishername>Clarendon
Press.</publishername><<address>Oxford</address>/publisher>
</biblioentry>
</bibliography>
```

- In the McGraw-Hill DTD, each article may have a <bib> element as a container for individual bibliographic citations encoded as <bb> elements. The <bb> elements have a mixed content model of #PCDATA and child elements marking particular features of the citation, but not as comprehensive a set as DocBook.

  Example:

```
<bib>
<bb>
<author>Renear, Allen, and Elli Mylonas, David Durand</author>.
<year>1996</year>. <pubtitle>Refining Our Notion of What Text Really
Is</pubtitle> In Nancy Ide and Susan Hockey (eds.), <pubtitle>Research
in Humanities Computing</pubtitle> <vol>4</vol>. Oxford:
<publisher>Clarendon Press</publisher>.
</bb>
</bib>
```

Discussion: From the reference point of the element which in each of the encoding models of the five DTDs most immediately contains the whole citation several types of heterogeneity emerge. Structurally equivalent elements have different generic identifiers — <cite>, <bibl>, <bibitem>, <biblioentry>, and <bb> — and different attributes, eg. <bibl> has id | n | lang | rend | default while <biblioentry> has id | lang | remap | xreflabel | revisionflag | (etc.). They represent different degrees of content-type-specific structuring, e.g. none of the possible parents of <cite> relates specifically to bibliographic information whereas <biblioentry> is always a child of either <bibliodiv> or <bibliography>. Furthermore, different positional constraints apply; eg. <bibl> might appear in the front matter or body of a document whereas <bibitem> can only appear in the rear of a GCA paper. Even things that look the same can be subtly different; eg. <bibl> and <biblioentry> both have a "lang" attribute, but in TEI Lite "lang" has a data type of IDREF and the default value keyword #INHERITED while DocBook's "lang" has a data type of CDATA and the default value keyword #IMPLIED.

3.4 Annotation markup

· HTML 4.01 provides no elements specifically for annotation. The <a> tag allows inline anchoring of out-of-line annotations which might be encoded using (for example) , <p>, or tags. The "name" and "href" attributes of <a> both have CDATA type values.

Example:

```
<p>... in a recent paper Michael Sperberg-McQueen, one of the editors
of the XML 1.0 specification.[<a href="#note01">1</a>] argues that ...
</p>
...
<h2>Notes</h2>
<p><a name="note01">Note 1.</a> The two other editors are Tim Bray and
Jean Paoli.</p>
```

· TEI Lite provides <note>, which may appear as a child of <body> (though not of <front> or <back>) and of several block and phrase-level elements. Several attributes are available to indicate the structural nature of the note: "type", "place", and "anchored". The last says whether or not the <note> is anchored by some point or span in the text, but while TEI Lite does have an <anchor> tag that functions similarly to HTML 4.01's <a>, <note> elements can be anchored by any other element; the "target" attribute takes a value of type IDREF which must correspond to an ID value in the same document. (The full TEI DTD allows bi-directional pointing between <anchor> and <note>, but in TEI Lite the pointing is unidirectional only, i.e. <note> to the anchoring element.)

Example:

```
<p>... in a recent paper Michael Sperberg-McQueen, one of the editors of
the XML 1.0 specification.<anchor id="anchor01" rend="*"/> argues that ...
</p>
...
<note id="note01" target="anchor01" anchored="yes" place="foot">* The two
other editors are Tim Bray and Jean Paoli.</note>
```

· The GCA DTD, unlike TEI Lite, distinguishes different types of notes at element level. Admonitory *nota bene* type notes are marked with <note>, a paragraph-level element that may appear within <body> either as a direct child or at section or subsection level. Regular footnote-type notes use <ftnote> together with the anchoring element <fnref> whose "refloc" attribute takes for its IDREF value the ID of the relevant <ftnote>. <ftnote> is a phrase-level element and there is no formally declared block-level container for it, which suggests that the structural placement of these notes is left to a stylesheet.

Example:

```
<para>... in a recent paper Michael Sperberg-McQueen, one of the
editors of the XML 1.0 specification.<fnref refloc="ftnote01"/><ftnote
id="ftnote01">The two other editors are Tim Bray and Jean
Paoli.</ftnote> argues that ... </para>
```

· DocBook's <note> resembles the GCA DTD's <note> in usage and structural status. Similarly, DocBook's <footnote> is like GCA's <ftnote>. <footnoteref>, however, is used not as an original anchoring element for a particular <footnote> but as a tag for additional references to that <footnote>. The DocBook DTD assumes that if an end user wants footnotes anchored by a symbol within the running text and the body of the footnote placed elsewhere the processing application will take care of it.

Example:

```
<para>... in a recent paper Michael Sperberg-McQueen, one of the
editors of the XML 1.0 specification.<footnote>The two other editors
are Tim Bray and Jean Paoli.</footnote> argues that ... </para>
```

· The McGraw-Hill DTD makes no provision for annotation of the article text, though it does add a <tfnl> ("table footnote list") element to table elements borrowed from the AAP Book DTD. The table footnote is an instance of another typical tagging situation: the need for element types that are specially managed variants of another element type, and that are specially processed and perhaps also restricted as to context. Table footnotes are conceptually just footnotes, but they can only occur within tables (restricted context) and they are formatted differently, since they are displayed within the table, and numbered uniquely only within the table.

Discussion: With annotation markup we see problems of homonymy (TEI Lite's <note> versus GCA's <note>), specificity (TEI Lite's <note> versus GCA's <note> and <ftnote>), and methods of referencing (HTML's "href" versus TEI Lite's ubiquitous "target" attribute versus GCA's "refloc").

3.5 Linking markup

· HTML 4.01 anchors links in a document with its <a> element. To link *from* an <a> tag the target is a URI given as the value of the "href" attribute. To allow linking *to* an <a> tag the "name" attribute is used. Both attributes can be used in the same <a> tag, allowing bi-directional linking.

Example:

```
<p>For more details, see [<a name="spe00-cit" href="#spe00-
ref">SHR00</a>]</p>

<h2>References</h2>
<a name="spe00-ref" href="#spe00-cit">[SHR00]</a> C. M. Sperberg-
McQueen, Claus Huitfeldt, Allen Renear. Meaning and Interpretation of
Markup. <em>Markup Languages: Theory & Practice</em>; 2(3); pages 215-
234; 2000.
```

· TEI Lite inherits some but not all of the full TEI DTD's suite of linking mechanisms. We describe in 3.2 how a <note> points to its anchoring element using the "target" attribute. Four elements - <ptr>, <xptr>, <ref>, <xref> — specifically indicate a link between points and/or pieces of content. As the generic identifiers imply, <ptr> has a simple pointing function that implies nothing about the relationship between source and target of pointing, whereas <ref> indicates that the source *refers to* the target. The targets of <ptr> and <ref> must be valid ID values within the same XML document.

The "x" versions of both elements allow pointing either to a target without an ID value but within the same document or to a target in a distinct XML document, by using in both cases "from" and "to" attributes with values in TEI's extended pointer notation, and in the second case also using the "doc" attribute to specify (via an entity reference) the document in which the target lies.

Example:

```
<p>For more details, see [<ref id="spe00-ref" target="spe00-
bibl">SHR00</ref>]</p>

<head>Bibliography</head>
<bibl id="spe00-ref">[<ref id="spe00-bibl" target="spe00-
ref">SHR00</ref>] C. M. Sperberg-McQueen, Claus Huitfeldt, Allen
Renear. Meaning and Interpretation of Markup. <title>Markup Languages:
Theory & Practice</title>; 2(3); pages 215-234; 2000.</bibl>
```

- In addition to its bibliographic citation and footnote linking mechanisms the GCA's DTD provides an <xref> element that serves the same purpose as TEI Lite's <ref>. The "refloc" attribute takes an IDREF value; the value of the "type" attribute can be either 'title' or 'number' and determines the formatting of the <xref> in output.

Example:

```
<p>This point is discussed further in <xref refloc="sct-01"
type="title"/>.</p>


<section id="sct-01">
<title>Section One</title>
...
</section>
```

formats as: `"This point is discussed further in Section One."`

- Several of DocBook's markup constructs allow linking behavior. In the case of two elements - <glossterm> and <firstterm> - the intent is to enable linking to a glossary entry for the word or term marked. Four elements enable general linking: <link>; <ulink>; <olink>; <xref>. The first, <link>, points to its target using an IDREF value on a "linkend" attribute. Like GCA's <xref> it has a "type" attribute whose value can determine application processing, though here the value-type is CDATA. The <ulink> element also has a "type" attribute, but identifies its target by a URL value on a "url" attribute. The <olink> element, like TEI Lite's <xref> and <xptr>, enables linking to another document, using an entity reference as value on a "targetdocent" attribute.

Example:

```
<para>This point is discussed further in <link linkend="sct-04"
type="internal">Section Four</link>.</para>
```

```
<section id="sct-04">
<title>Section Four</title>
...
</section>
```

- The McGraw-Hill DTD has several linking elements. The <xref> element is used to link to other articles by using an IDREF value on an "rid" attribute. The <urllink> element is equivalent to HTML's <a>, and takes a URL as value on the "url" attribute.

  Example:

```
<para>See also <xref rid="12345" type="internal">[article
name]</xref>.</para>


<article id="12345"">
...
```

  The McGraw-Hill DTD also offers another linking element, <link>, which links to a multimedia object identified with a CDATA value on a "filename" attribute, the kind of object being chosen from a fixed list of values on a "type" attribute.

  Example:

```
<para>The unusual design of this building <link type="graphic"
filename="fig1.png" id="ln01"/> is based upon ....</para>
```

  formats after processing as: `"The unusual design of this building [Figure One] is based upon ..."`

Discussion: The McGraw-Hill <link> element differs from the others that we have looked at in that it is EMPTY, and does not surround anchor text like many of the other linking elements. This is because it is used in cases where the link anchor text is to be generated by the document processor. From an authoring point of view this is the best way to manage citations to bibliographic items, tables, figures and the like, since it ensures uniformity of references, as well as allowing automatic updating of reference strings when the document is updated. This is a general problem in text markup, where markup can often lead to the generation of textual content that is not found directly in the source file. While reference strings are a common case, sometimes substantial quantities of boilerplate text like safety warnings will be represented by a single tag. Autogeneration of punctuation, while not always reliable, is another frequent case.

In descriptive tagging, such tags as present problems, since it may be important to preserve variation in the original anchor text. This is often true even if the anchor text was intended to be uniform, as numbering and labeling systems often become inconsistent over time, leading to reference numbers systems that no longer count, but include occasional irregular increments like 4a, 4b.

3.6 High-level structure markup

Documents have a predominant textual data structure according to the type of content. A prose document, for example, will primarily be structured as paragraphs. Documents often also display genre-specific structural organization at a higher level than the predominant textual data structure: paragraphs of a novel organized into chapters; menus grouped into sections by principal ingredient, etc.

- HTML 4.01 has a block-level element which may contain either block or inline child elements and which may therefore be used as a child of <body> to organize <p>, <list>, <table>, <blockquote>, etc. There is no specific "type" attribute, but the "title" attribute could be used to describe the division.

  Example:

```
<body>
<div title="chapter-one">
<h2>Chapter One</h2>
<p> ... </p>
<p> ... </p>
<p> ... </p>
<p> ... </p>
</div>
```

  However, HTML documents structured with the element are relatively rare. Most documents have structural divisions that are marked by the use of the HTML heading tags <H1>- <H6>. Because these are frequently used for their formatting effect, there is no guarantee that the heading levels will nest in a sensible way. This is a very common form of "tag abuse" which is seen in many documents that use a DTD that mark headings rather than divisions. Furthermore, the transformation from a document that uses such tags to a division tagging scheme can be very difficult. For instance in the example below it is not clear if the H3 tags should be enclosed in an extra level of elements, or if they should be treated as "really" <H2> elements, with a variant rendering. Furthermore, an analyst's judgement in such a case might change depending on what the content of the headings was. Ignoring the last heading, in the example below, it looks like <H3> is being used as a preferred way to format a second level heading. If one adds the last line into consideration, the question of how to construct a consistent and meaningful division structure is considerably more difficult. If there were a similar <H2> element at the start of the document, containing the text "Method", the problem would become even more acute.

  Example:

```
<body>
<H1>chapter 1</H1>
<p> ... </p>
<p> ... </p>
<H3>Section 1.2</H3>
<p> ... </p>
<H3>Section 1.3</H3>
<p> ... </p>
```

```
<p> ... </p>
<H2>Discussion</H2>
```

- TEI Lite XML offers two means of high-level structuring. The first is a generic element that groups block level elements. It can also contain itself, allowing unconstrained nesting of structural levels. Unlike HTML's it carries a "type=" attribute. The second means is a series of explicitly numbered elements from <div0> through <div7>. These elements make level-dependent encoding and processing easier. As with the generic , all the numbered division elements carry "type=" attributes. Generic s and numbered s may not mix in the same document.

  Example:

```
<body>
<div type="part" id="pt-01">
<head>Part One</head>
<div type="section" id="sct-01">
<head>Section One</head>
<p> ... </p>
<p> ... </p>
<p> ... </p>
<p> ... </p>
...
```

- The GCA DTD has a <section> element which, like TEI Lite's , can be a child of <body> and which groups block-level elements. Unlike , <section> elements cannot nest recursively; instead <section> can contain <subsec1>, <subsec1> can contain <subsec2>, and <subsec2> can contain <subsec3>. The only attribute carried by <section> and <subsec*N*> is "id=".

  Example:

```
<body>
<section id="sct-01">
<title>Section One</title>
<subsec1 id="sub1-01">
<p> ... </p>
<p> ... </p>
<subsec2 id="sub2-01">
<seqlist> ... </seqlist>
</subsec2>
</subsec1>
</section>
```

- In the DocBook DTD the "single document" element is <book>; below that can occur a high-level structuring element called <part>, which has a "label" attribute similar to HTML's "title". <chapter> elements can be children either of <book> or of <part>. in turn, <chapter> can have paragraph-level children (eg. <para>, <formalpara>, <simpara>) or can contain numbered or unnumbered sectioning elements (eg. <sect1>, <simplesect>, <section>). The numbered sectioning elements nest down to <sect5> level; <section> elements can nest to any depth; <simplesect> elements cannot have any

sectioning children.

Example:

```
<book>
<part label="I">
<title>Part One</title>
<partintro><para> ... </para></partintro>
<chapter>
<section>
<section>
<p> ... </p>
<p> ... </p>
</section>
<section>
<p> ... </p>
<p> ... </p>
</section>
</section>
...
</chapter>
```

· McGraw-Hill's *Encyclopaedia of Science and Technology* article DTD does not, of course, specify structuring elements above <article>. Below <article> the main element for the article content, <atext>, can contain regular block-level elements (for paragraphs, lists, etc.) or numbered sectioning elements. Unusually, where other DTDs constrain such elements to nest in order, <atext> can have either <sec1> or <sec2> children. However, <sec3> elements cannot be children either of <atext> or <sec1>, but must occur in <sec2> elements. All the sectioning elements have a <st> ("section title") child, but no attributes.

Example:

```
<atext>
<sec1>
<st> ... </st>
<p> ... </p>
<p> ... </p>
</sec1>
<sec1>
<st> ... </st>
<p> ... </p>
<p> ... </p>
</sec1>
...
</atext>
```

Discussion: As we explored in the discussion of HTML division tagging, the two major way of encoding structural divisions may be practically difficult or impossible to convert, because the obvious algorithms

based on heading level will not work for DTDs where headings are marked instead of divisions, and they are not nested strictly. In addition, DTDs provide methods of marking high-level structure that has the potential for heterogeneity even *within a single encoding model* if it allows numbered and unnumbered versions of an element. In a round-trip transformation the information lost in the numbered-unnumbered leg is hard to replace in the unnumbered-numbered leg.

4 Taxonomy of heterogeneity among encoding models

On the basis of the sample DTDs discussed here and of previous related work [KS91a; Duw97; SHR00] we distinguish several generic types of heterogeneity between encoding models: semantics-based; granularity-based; constraint-based; inheritance-based; data-based. Semantics-based refers to heterogeneity of *natural semantics*, i.e. to the everyday dictionary-type linguistic meaning of an element or attribute. Granularity-based includes the number and nesting depth of elements each model offers to encode a given piece of content and the amount of information added via attributes. Constraint-based refers to where and how many times an element may occur, to what values may or must be used with attributes, and to what data types the encoding model specifies for particular element contents and attribute values. Inheritance-based heterogeneity is a special case because it represents a *consequence* of problems caused by another type. Elements commonly possess properties by inheritance from an ancestor or by bestowal from a structurally unrelated element. Although invisible in the element's markup and in its generated XML infoset, these properties nevertheless form an important part of the element's *inference semantics*, i.e. the inferences the markup licenses [SHR00] (see Section 5.2). The failure to successfully translate an attribute in schema *A* into an equivalent in schema *B* means that any elements which inherited that attribute's distributed property in *A* will no longer have it in *B*, which could mean a significant information loss. Lastly, data-based heterogeneity involves differences in the format of existing content or differences introduced when a scheme-specific processing application generates content. These differences are particularly problematic because they may be *independent* of both the encoding scheme and the actions of any tree-transformation language involved in the scheme-to-scheme conversion.

Table 1 summarizes the different types of heterogeneity and gives examples.

Table 1: Types of XML DTD/Schema heterogeneity

Type	Description	Example
Semantics-based		
	Different names for semantically equivalent elements or attributes	[A] has <result> [B] has <sum>
	Same name for semantically different elements or attributes	[A] has <cat_num> meaning "the number this part has in the catalog" [B] has <cat_num> meaning "the number of the catalog this part is listed in"

		[A] and [B] both have <xsd:attribute name="weight_in_tons" type="positiveInteger"> but in [A] the units are long tons and in [B] they are short tons.
	Incompatible segmentation I	[A] uses normal paragraph tags: <p>...<p>, while [B] uses so-called milestone tags to indicate breaking points: <doc>....<p/>...</doc> In [B] an empty tag marks a paragraph break rather than the paragraph itself. This is a similar difference to the one between heading and division-style section breaking. Milestones are often (properly) used for hierarchy-breaking divisions like original publication page breaks.
	Incompatible segmentation II	[A] has markup, while [B] has heading-level based markup.
Granularity-based		
	Different number of elements mark equivalent content	[A] has <name>John Smith III</name> [B] has <name>John Smith</name> <designator>III</designator>
	Equivalent wrapper elements but different internal structures mark equivalent content	[A] has <location>Paris, France</location> [B] has <location><city>Paris</city> <country>France</country></location>
	Different number of attributes convey equivalent information	[A] has <aircraft manufacturer="Boeing" model="747"/> [B] has <aircraft type="Boeing 747"/> This is almost an attribute-based syntactic variant of "different number of elements mark equivalent content". It's also a variant of data incompatibility, because the contents of the attribute in [B] are generated from those in [A] by applying a function to them (concatenation).
	Attribute(s) not in common	[A] has <author regName=""> [B] has just <author>
Constraint-based		
	Different positional constraints on equivalent elements (Could be seen look-forward or look-back; here we view look-back, i.e. constraint imposed by	Example 1. [A] has <!ELEMENT foo -- (bar,blort) > [B] has <!ELEMENT foo -- (blort,bar) >; therefore [A]'s <blort> will be invalid under [B] because it should precede <bar> not follow it.

	an ancestor's content model)	Example 2. [A] allows <bannerAd> in any main structural division of the document [B] constrains <advertisement> to appear in <backMatter>
	Different number-of-occurrence constraints on equivalent elements (viewed look-back)	[A] has <!ELEMENT foo -- (bar,blort+) > [B] has <!ELEMENT foo -- (bar,blort?) >; therefore within each <foo> any <blort> after the first will be invalid under [B]
	Different requirement constraints	[A] has <!ATTLIST attribution certainty (high | medium | low) #REQUIRED> [B] has <!ATTLIST attribution certainty #IMPLIED>
	Different type constraints eg. strings versus integers	Schema [A] has <xsd:attribute name="shelf_num" type="string"> schema [B] has <xsd:attribute name="shelf_num" type="positiveInteger">
	Different default values	[A] has <!ATTLIST report access (public | restricted) public> [B] has <!ATTLIST report access (public | restricted) restricted>
Inheritance-based		
	Element inherits "invisible" properties either from an ancestor or from a special element elsewhere in the document (eg. TEI's <tagsDecl>)	[A] has <text language="French"><para> ... </para></text> [B] has <text><para> .. </para></text> Thus in [B] the <para> element no longer has the property of French-ness.
Data-based		
	Element content uses different data format	[A] has <date>1/1/1</date> [B] has <date>January, 1, 1901</date> Thus in either schema the <date> element can only be converted by a program that can parse one date format and generate another.
	Explicit vs. generated data inconsistency	[A] has <link dest="table1"/> [B] has <link dest="table1">(See Table 1.)</link> [A] and [B] are intended to be equivalent, but [B] has content data that [A] doesn't. Instances like [B] may vary (deliberately or accidentally) from the intended text

		generated by applications that use [A].

5 Solution spaces

5.1 Semantics-based heterogeneity

As we noted in Section 1, a structural transformation is *faithful* if the external model of the output document is correct whenever the external model of the input document is correct. This invokes a criterion of *semantic appropriateness*: the target schema tag should have a logicist semantics equivalent or similar to that of the source schema tag it maps to. Otherwise, content under the new schema could be tagged with an element structurally acceptable but semantically inappropriate, greatly devaluing the new document as an information resource. Current work on creating a "semantic web" aims at standardizing means for representing the logicist semantics of tags.

Every encoding model implies an ontology: a logicist version of the natural semantics of the concepts the tag set reflects. A semantic web involves realizing such ontologies in a way that makes them easily accessible, comparable, and integrable with existing web resources. Depending upon the degree of semantic complexity required, users can create ontologies just using two existing W3C Recommendations for web metadata - the *Resource Description Framework (RDF) Model and Syntax Specification* (RDFMS) and the *Resource Description Framework (RDF) Schema Specification 1.0* (RDFS) - or can supplement those with the DAML+OIL language. The RDFMS specifies a subject-predicate-object model of metadata where the subject is a resource (usually, but not necessarily, addressable by a URI) about which things are predicated (properties) that involve objects (literal values of the properties). Realized as markup this model has the form:

```
<rdf:RDF>
<rdf:Description>
<someSchema:foo>John Smith</someSchema:foo>
<someSchema:bar>Jane Doe</someSchema:bar>
</rdf:Description>
</rdf:RDF>
```

RDFMS describes only the fact of the class of property elements and does not give any particulars. In the example above, the <someSchema:foo> and <someSchema:bar> property elements would come from a schema or schemas constructed according to RDFS. This specification describes its purpose as follows:

> This document does not specify a vocabulary of descriptive elements such as "author". Instead, it specifies the mechanisms needed to define such elements, to define the classes of resources they may be used with, to restrict possible combinations of classes and relationships, and to detect violations of those restrictions.

RDFS allows users, for example, to construct a schema in which they declare a class of objects called documents, with sub-classes of handwritten, printed, and electronic documents. They can declare that documents have properties such as one or more authors, titles, editions, publishers, etc. RDF Schema also allows users to specify constraints on the classes and properties they declare. For example, users might

specify that while all documents can have authors only printed documents can have printers and only handwritten documents can have scribes.

The DAML+OIL language (which is realized as RDF) extends RDFS to enable more detailed ontological specification. For example, the enumeration feature allows users to declare a class and exactly those instances that comprise it. As another example, DAML+OIL enables cardinality constraints, eg. users could specify that a *person* has only one *biologicalFather*.

Under semantic web conditions, transforming a document from schema [A] to schema [B] involves [A] and [B] either pointing to or including one or more formal ontologies constructed using RDFMS syntax, RDFS constructs, and perhaps DAML+OIL extensions. A software agent programmed with the necessary criteria discovers and compares the declarations for an element in [A] and an element in [B] to assess the degree of equivalence between them. The application making the transformation chooses its steps according to the agent's findings.

The utility of the semantic web will depend upon both the availability of ontologies *and* their comparability. Having a standardized syntax to express semantics in does not guarantee the discovery of equivalence. Each ontology involved in a comparison could be a totally self-referential structure with no *overt* connection between them even though in fact the two structures of terms delimit the same concept. It will be important therefore to have either standard centralized ontologies that all schema creators in a particular industry, government department, field of knowledge, etc. agree to use; or at least standard reference sources for synonyms (eg. a standard thesaurus), word forms and usages (eg. a standard dictionary), and ontological structures. The last could be served by a general ontology such as the OpenCyc Knowledge Base in DAML format (www.cyc.com/cyc-2-1/cyc.daml).

5.2 Constraint-based and granularity-based heterogeneity

As [JKS00] shows, XSLT can solve many of the constraint- and granularity-based types of heterogeneity we identify. Within limits, it allows users to rearrange element structure, create new attributes, and change attribute values to element content and vice versa. As we noted in section 1.2, a principal limitation is that the transformation involves discrete units of information which in the result document must still make sense in their hierarchical context.

5.3 Inheritance-based heterogeneity

Markup may "mean" in several ways. If an element's generic identifier is a word or phrase from a known language, then by common-sense implication we can assume it is intended to convey one of the word's known meanings in that language. This is the element's natural semantics, which may be formalized into a logicist semantics by the methods described in section 5.1. Here the meaning resides *outside* the markup system. Another approach to markup semantics, described by Sperberg-McQueen, Huitfeldt, and Renear [SHR00], considers the inferences the markup licenses without recourse to its natural (or logicist-natural) semantics. That is, it concerns itself with what can be truly predicated of any node in the markup structure of a document, so that *the set of predicates* constitutes the node's "meaning". These predicates may be expressed in any suitable language -- in [SHR00] the authors use Prolog. The predicate set includes those inherited properties that are invisible in the actual markup or the XML information set it

generates.

6 Conclusion

We have explored the problem of semantic heterogeneity in XML from several perspectives. Theoretically, we considered the question of what is really meant when XML markup is described as being semantic, and we considered some closely related forms of semantics. The ordinary-language sense of semantics for XML data representations turns out to be a relationship between types of markup, and a conceptual model of data constructed by an analyst. We also saw that XML is unusual in not having a clearly fixed operational semantics, because XML documents are supposed to be largely independent of particular applications to which they may be put. Indeed, the more independent a schema is of any particular application, the more completely "semantic" that schema is held to be.

Computer realizations of semantics are always just more data. Such metadata expresses some form of predication about other, primary data. This analysis has focused on an important, but relatively unexplored kind of metadata: metadata describing the structure and purpose of the marked-up elements that make up a document, rather than metadata describing the content of the document. This particular level of metadata is still poorly understood, and parallel to the more well-known level of metadata about document content. Schema languages provide ways to constrain documents to have particular forms, but do not express very much about what those forms really mean, except in the application of names to elements and attributes. The analysis of semantics given here, (like the work by Sperberg-McQueen, Huitfeldt, and Renear) makes clearer what questions such possible extensions to schema languages will have to address.

We also found that even in the absence of a single application to serve as a benchmark for judging the operational consequences of semantic incompatibilities, document conversion, generally thought of as a purely applied problem in text processing, can be used as an operational model for judging the extent of the semantic incompatibilities between schemas. Simply put, even without a canonical application, we can evaluate the operational consequences of incompatibilities between two schema's definitions by examining how hard it would be to convert from one to the other. Not all obstacles to conversion are of equal severity for all applications, but the notion of conversion makes it possible to localize problems.

By analyzing a variety of DTDs used in publishing we were able to construct a preliminary taxonomy of types of schema incompatibility. Some problems, like the problems in incompatible global document structures and generated text are unique to marked-up text. This taxonomy is a starting point, since it is certainly incomplete, but we believe that this represents a breakdown of the problems of schema conversion that will contribute to moving the process of DTD analysis and manipulation from art to science.

There are many avenues of future work leading from the start presented here. One important thread will be to validate and expand the taxonomy of incompatibilities, and to continue to re-organize it. Many of the distinct problems that we have identified may be analyzable as compounds or compositions of other problems. In addition to refining the analysis of different types of semantic incompatibility, we would like to return to the area of document transformation languages and see if it is possible to use this taxonomy of phenomena to improve the ease of creating document conversions, and perhaps the quality of the results as well.

References

[AQR97] Extase Akpotsui, Vincent Quint, and Cécile Roisin. Type modelling for document transformation in structured editing systems. *Mathematical and Computer Modelling*; 25(4); pages 1-19; 1997.

[Arn93] Dennis S. Arnon. Scrimshaw: a language for document queries and transformations. *Electronic Publishing*; 6(4); pages 385-396; 1993.

[Bal] http://balise.xoasis.com/doc/doc.htm

[Bos01] Marat Boshernitsan. HARMONIA: a flexible framework for constructing interactive language-based programming tools. Technical Report CSD-01-1149, University of California, Berkeley, June 2001. http://www.cs.berkeley.edu/~harmonia/publications/harmonia-pubs.html

[Col97] Robert M. Colomb. Impact of semantic heterogeneity on federating databases. *The Computer Journal*; 40(5); pages 235-244; 1997.

[DBK98] Susan Davidson, Peter Buneman, and Anthony Kosky. Semantics of database transformations. *Semantics in Databases*; Leonid Libkin and Bernhard Thalheim, eds.; pages 55-91. Springer, 1998.

[Duw97] R. M. Duwairi. *Views for Interoperability in a Heterogeneous Object-Oriented Multidatabase System*. PhD thesis, Department of Computer Science, University of Wales College of Cardiff; 1997.

[EWMM00] H.T. El-Khatib, M.H. Williams, L.M. MacKinnon, and D.H. Marwick. A framework and test-suite for assessing approaches to resolving heterogeneity in distributed databases. *Information and Software Technology*; 42; pages 505-515; 2000.

[FW93] An Feng and Toshiro Wakayama. SIMON: a grammar-based transformation system for structured documents. *Electronic Publishing*; 6(4); pages 361-372; 1993.

[GSC96] M. Garcia-Solaco, F. Saltor, and M. Castellanos. Semantic heterogeneity in multidatabase systems. *Object Oriented Multidatabase Systems: a Solution for Advanced Applications*; O.A. Bukhres and A.K. Elmagarmid, eds; pages 129-202; 1996.

[Hul97] R. Hull. Managing semantic heterogeneity in databases: a theoretical perspective. *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of database systems*; pages 51-61; 1997.

[HP00] Haruo Hosoya and Benjamin C. Pierce. XDuce: a typed XML processing language (preliminary report). Proceedings of *International Workshop on the Web and Databases (WebDB'2000)*; Dallas, Texas. 2000.

[ISO91] ISO/IEC, *Information Technology - Text and office systems - Documents Style Semantics and Specification Language (DSSSL)*. ISO/IEC DIS 10179. 1991.

[JKL97] Jani Jaakkola, Peka Kilpeläinen, and Greger Lindén. TranSID: an SGML tree transformation language. *Proceedings of the Fifth Symposium on Programming Languages and Software Tools*; pages 72-83, 1997.

[JKS00] Senthil V. Jayavelu, Kailavani, and Saravanan. *XML Schema Interoperability Using XSLT*. Unpublished online report. http://www.cs.tamu.edu/course-info/cpsc608/hohin/fall00/projects/NASA-team3/projreport.htm

[KKR01] Gerti Kappel, Elisabeth Kapsammer, and Werner Retschitzegger. Architectural issues for integrating XML and relational database systems - the X-Ray approach. *Proceedings of XML Technologies and Software Engineering*; 2001.

[KS91a] W. Kim and J. Seo. Classifying schematic and data heterogeneity in multidatabase systems. *IEEE Computer*; 24(12); pages 12-18; 1991.

[KPV94] E. Kuikka, M. Penttonen, and M.-K. Vaisanen. Theory and implementation of SYNDOC document processing system. *Proceedings of the Second International Conference on Practical Application of Prolog, PAP-94*; pages 311-327; 1994.

[ME96] E. Maler and J. El Andaloussi. *Developing SGML DTDs from Text to Model to Markup*. Prentice Hall, 1996.

[Mil98] Renee J. Miller. Using schematically heterogeneous structures. *Proc. ACM SIGMOD*; 27(2); pages 189-200; 1998.

[Omn] http://www.omnimark.com

[SHR00] C.M. Sperberg-McQueen, Claus Huitfeldt, Allen Renear. Meaning and interpretation of markup. *Markup Languages: Theory & Practice*; 2(3); pages 215-234; 2000.

[TL94] Henry Tirri and Greger Lindén. ALCHEMIST: an object-oriented tool to build transformations between heterogeneous data representations. *Proceedings of the Twenty-Seventh Annual Hawaii International Conference on System Sciences*; vol. II; pages 226-235; 1994.

[XMLIS] W3C. *XML Information Set*; John Cowan and Richard Tobin, eds. http://www.w3.org/TR/xml-infoset/

[XSLT] W3C. *XSL Transformations (XSLT) Version 1.0*; James Clark, ed. http://www.w3.org/TR/xslt