Commentary and Opinion

Beyond the "descriptive vs. procedural" distinction

Wendell Piez
Mulberry Technologies, Inc.
17 West Jefferson St.
Suite 207
Rockville MD 20850
EMAIL wapiez@mulberrytech.com

There has come to be a consensus that the "procedural vs. declarative" distinction is useful, if only as a rough guide, in the design of markup languages. To understand how and why this is the case, we need to ask questions that are usually left unasked when this principle is proposed, such as "is it the model (the schema) that we consider to be descriptive, or the tagged document?" or, more deeply, "why do we validate our markup anyway?"

A number of our fundamental assumptions are not always true. Sometimes a schema might be more than a "go/no-go gauge", becoming a diagnostic and investigatory instrument. Sometimes marked-up documents look backward (as representations of something preexisting), not just forward to processing. Sometimes semantic opacity is a feature, not a bug. In order to understand the power of markup languages, it is helpful to keep in mind that they are both technologies and a species of rhetoric. New characterizations and categories of markup languages may help focus our design efforts.

Why are we still talking about the "descriptive/procedural" distinction? The conception continues to be a focus, because it continues to have explanatory power. Yet at the same time, it clearly demands quite a bit of refinement when we look at the increasingly broad spectrum of different markup languages, and markup language applications, that are now proliferating. Descriptive? Of what? Separation of format from content? What's content, and what's format? Apart from its format, to what do we refer to determine what "content" is; how do we specify it, and how do we go about designing tags for it? What kind of thing are we trying to model, anyway?

The traditional arguments around the "descriptive/procedural" distinction[1] have detailed a number of advantages to descriptive markup languages (also

1 Two "canonical" references I consulted ([Goldfarb 1990] and [Sperberg-McQueen 1994]) make the distinction between "descriptive" and "procedural" approaches. To label the different design strategies "declarative" and "procedural", while observing a distinction with a history in computer science, is evidently problematic in this context, mainly since those terms are so relative. In this paper, as I am deliberately reflecting on the distinction as traditionally rationalized, I'll use the terms "descriptive" and

Wendell Piez, "Beyond the "descriptive vs. procedural" distinction"
Markup Languages: Theory & Practice 3.2 (2001): 141–172

loosely identified as "generic" languages) over their procedural cousins: scalability, reusability of data, and so forth. While these advantages are demonstrably real, nonetheless the evolution of XML technologies, especially in such applications of XML as XSLFO, SVG, SMIL, or even XSLT,[2] shows that the opposite approach to designing a markup language also is playing an important role. Sometimes, it is clear, a procedural language is exactly what we want.

At the Extreme 2000 conference on markup technologies in Montreal, Allen Renear picked up the task of scrutinizing the opposition, proposing an alternative framework for which he introduced terms from linguistics and speech-act theory. I'll return to Renear's argument; but in order to get at these issues at a deeper level, I start by pointing out one begged question, and potential ambiguity, generally at issue when we assert these categories. When we describe a "markup language" (or a "tag set") as descriptive or prescriptive, are we talking about model or instance? That is, are we talking about the proposed, implied or asserted semantics of an abstract model for a document type (classically, as formalized by a DTD); or are we making generalizations about the tags in use, that is the (implied or effective) semantics of element and attribute types as instantiated in documents? It matters which one of these we are describing, not simply because they may be different (in the ideal case they should perhaps not be), but because the very fact that the two things (DTD and document) might possibly end up "meaning" something different in practice, raises questions about the relation between model and instance. In the real world (not to put too fine a point on it), sometimes users "mean" tags in ways not intended by designers, and this fact bears directly on the problem because it indicates how a model's "description" of a document *type* may not be exactly what a document's own tags "describe" (or may be purported to describe, depending on who you talk to).

Now any designer will seek, and will probably assume, that the semantics of model and instance should be the same, or at least not at cross-purposes. When they diverge, we call it "tag abuse", thus begging the question by simply handing authority for correctness to the designer's "intent", whether stated or implied. In fact, since we generally design models first and write instances later, it is a design goal, necessarily implicit and always assumed, that the model be complete and well-fitted enough to the problem domain, that its semantics[3] can be effectively

---

"procedural" when referring to the traditional dichotomy, and occasionally loose synonyms such as "generic" or "presentational" when it serves my purposes (one of which is, of course, to clarify what we might mean when we use these terms).

Another examination of this issue, tracing out several possible "axes" along which distinctions may be made (and thereby anticipating the arguments of Allen Renear [Renear 2000] and myself), is Mavis Cournane's. See [Cournane 1997].

2 Note that in this context, XSLT is procedural as a *markup* language (the tag set is closely bound to a set of processing requirements), while being declarative as a transformation or *processing* language: it is procedurally bound to a declarative application. This in itself is an indication of how relative these terms are.

3 Note here I specifically mean the *implied* semantics of the model, not any behavioral or operational

reflected in instances without strain. But it may also be that to engineer a system in which this ideal may be realized (or *nearly* realized), we had better come to an understanding of how the model and the instance relate to each other not just in theory as an objective, but also in practice, where things always seem to have at least the potential of falling short, and where nothing is so certain as the human capacity to introduce uncertainty through creative adaptation. I will suggest that model and instance need not always relate to each other in the same way; and in fact that the way requirements dictate they must relate to each other in any given application of a markup language, has a direct impact on the suitability of different strategies available to the designer. Since these strategies are commonly framed by distinguishing descriptive vs. prescriptive, declarative vs. procedural, or any of several other oppositions down to "separation of presentation [or format] from content", it is ultimately this distinction that we are illuminating.

To ask how model and instance relate to each other is to ask, in a very general way, about the process and role of what we usually call *validation*, that is the process by which model is applied to instance. (It is not the only such process; but the nature of validation — and usually, its purpose — is such that it can be taken to stand in for others.) So the first thing we need to consider is what validation is and why we do it.

Why it matters what "validation" is

What is "validation"? As soon as asked, it turns out that this is very much a live question. XML and XML-based technologies have lately been serving as an incubator for all kinds of new approaches to validation. Some seek merely to recast inherited notions of validation into new forms (presumably more tractable), some seek to enhance them with capabilities of alternative validation regimens, and some may go in entirely new directions. To say nothing of non-XML approaches (and I hope we see plenty of innovation on this side as well, insofar as there are certainly significant features of texts and interesting problems to which XML does not easily lend itself), in XML we have well-formedness checking, DTD validation, XML Schema, RELAX, TREX, XML-Data Reduced, Schematron, Examplotron, etc. etc.

It is not my concern here to consider these in any detail, or even to distinguish between them, except to point out the interesting (and significant) fact that they do not all take the same thing as their object of examination. Basically, when we validate, we take an instance (an "XML document") and a model (the

semantics that may be actuated in code. As Robin Cover points out, between DTD and the markup constructs as implied by the syntax, SGML/XML is a fairly weak format for specifying the latter [Cover 1998]. We are always free, however, through names, relationships, or explicit documentation, to assert informal "human" semantics: to say, that is, what we think we mean.

"schema" or "specification"), and compare these for purposes of saying whether the instance conforms to the model, or in what ways it fails to conform. But some of these approaches work on an XML document as a text entity (a sequence of alphanumeric characters, some of which constitute data, some of which constitute markup, as per the XML Recommendation [XML 2000]); while others operate on some kind of more complex structure, typically a document object or "infoset" held in memory. An important aspect of this is how the formalization in XML of well-formedness gives us a new platform on which to build and standardize validation techniques. A definition of "well-formed" (as distinct from "valid") brings with it the capability of doing what could be called a "plain parse", rendering a sequence of characters into an abstract information set without otherwise concerning ourselves with the higher-order semantics of elements and attributes. This is important because we may not know or care about such higher-order semantics every time we process. And when we do, testing the conformity of an XML document to any particular semantic profile (however represented) becomes, properly, just one more kind of processing, albeit of a distinctive kind (or to a specific end). Thus validation, considered in light of its purposes and often its methods, is actually closer to querying, say, or to transformation, than it is to parsing as such.

The proliferating approaches to validation also demonstrate that (among other things) any XML document — whether considered as a stream of characters or as an abstract information set — potentially exhibits a range of different features or characteristics which we might be interested in testing:

- Constraints on structure of elements and attributes by type ("a *head* is permitted inside a *chapter*, but a *chapter* is not permitted inside a *head*").
- Conformity of data elements (element content or attribute value) to specific lexical or other requirements: data type integrity; "authority control"
- Referential integrity of links and pointers
- etc. etc.

Any or all of these might be considered to be properly within the realm of validation; and more to the point, the list is as open-ended as we wish it to be.

Validation and workflow: strict validation

The intention or purpose of validation is to subject a document or data set to a test, to determine whether it conforms to a given set of external criteria. Validation may thus be distinguished from processing in general, which may not bother to conduct any such tests (or which may use other tests). It is precisely because the range of features and characteristics in which we are interested, and which we need to be able to constrain, is so open-ended, that testing becomes a useful thing to do in practice. (If it weren't, our tools could all be built so as not to need tests.) Our need to test is simply explained and understood (so much so that it

rarely needs to be explicated): if there exists a point in a process where it is less expensive to discover and correct problems than it is to save the work of testing and fix at later points, it is profitable to introduce a test. The ideal workflow, that is, is one in which we make any correction or adjustment to the materials being processed at the point where it is easiest and least expensive, making allowances for the expense of running tests. This assumes, of course, a workflow that is sufficiently defined to make this possible.

We validate, that is, because we want to know *in advance* of something whether our data set conforms to a set of specified requirements. Notice a key concept we have introduced here: such an operation only makes sense, and only becomes necessary, in an articulated workflow. Validation, that is, is a type of "quality assurance" applied at a particular stage in processing. We need downstream processing to be predictable, and wish to engineer away, to whatever extent we can, any possibility of having to decide how to resolve or render any given anomaly (however interesting it may be) at a later stage of processing. Rather, we want to invest energy now in assuring that our data already conforms to a set of clearly-understood criteria.

In fact, this is nothing more than the application of a simple rule of industrial engineering, here applied to information systems. In effect, we are designing a process (even if a simple one) — an assembly line. Validation provides us with what is called a "go/no-go" gauge.

This is not merely an analogy. If we look at the beginnings of mass production technologies, we find a significant transition occurs in the nineteenth century with the development of the"American System of Manufacture".[4] What distinguished this approach to mass production from previous efforts is that the ancient principle of division of labor was joined with a new one: making the component parts of the product to be interchangeable. Division of labor, of course, has been practiced for many centuries and in a range of societies worldwide. (Nor is it limited to human culture, being found also in the natural world.) But by itself, division of labor is not sufficient to win the economies of scale that result from modern manufacturing methods. As long as parts were not interchangeable, production of any manufactured item had to be done on a piece-by-piece basis, each piece being unique. Only when the *individual components* of a manufactured item were submitted to quality control mechanisms, such as jigs, gauges, and quality checkpoints, could higher-order economies be realized.[5]

A "go/no-go" gauge is a device used precisely to provide such a check. The utility, and ubiquity, of such a device is instantly recognizable to anyone working

4 As it was dubbed at the Crystal Palace Exhibition in London in 1851, where the arms manufacturer Colt demonstrated interchangeable parts. Needless to say, there is nothing inherently "American" about the principle (an idea that had been around, in Old and New Worlds, for many decades) or its application.

5 See [Hounshell 1984].

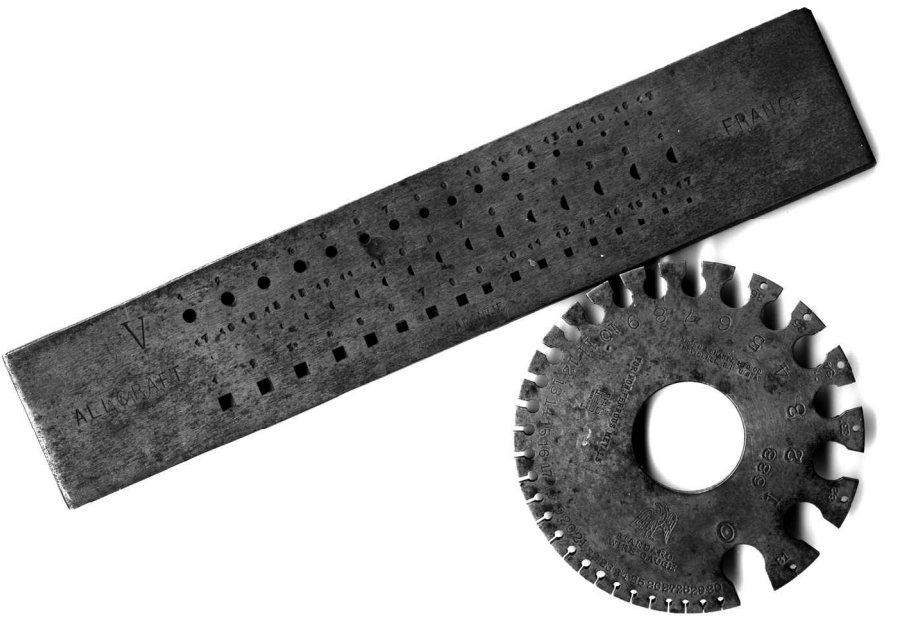

Figure 1 | A go/no-go gauge, with accompanying jig

The gear-shaped device is a wire gauge. Either a length of wire is a specified thickness, or it is not. Pictured with the wire gauge is a draw plate, used for drawing out wire of different gauges (thus serving as a jig). The draw plate must necessarily conform to the gauge in its measurements (and be checked from time to time to be sure it has not worn).

A DTD or formal schema functions as a gauge when we use it to perform strict validation, as a jig when we use it to configure, for example, a structured editing tool.

Thanks to B. Tommie Usdin for lending these examples of the tool-maker's art.

with a complex workflow — especially a process which already involves a complex division of labor or differentiation of roles.[6]

Although they are no longer physical objects, we test our information sets against abstract specifications for the same reason that in a factory, the machine tools are set up to mill parts to exact specifications, and are frequently tested (the tools themselves, that is) to reassure conformity. In fact, the markup industry's leaders have unerringly, if not always deliberately, been proponents of open standards for markup technologies for the exact reason (among others) that it is standards-based interchangeability, when applied to information objects, that provides us with the coveted advantages for our data of vendor- and application-independence, of modular architectures and layered systems, commodity tool

6 A significant detail about gauges used in machine tooling is that they may be crafted, and must be maintained, by hand. In effect, the craft shifts from the creator of each individual product, to the industrial engineer who develops a product *line*.

markets, and long-term data stability. (Not that any of these things become easy to achieve even on a standards basis: but at least with standard ways of judging correctness, there is some hope for them.)

Whenever a validation technology is applied this way, I think it appropriate to call it "strict": I want to convey that it proceeds by posing a binary choice: thumbs-up or thumbs-down. Note that this does not indicate anything about what, precisely, is being validated (structures, data types, referential integrity etc.), or even how extensively, but rather the manner of and rationale for validation. The expectation is that if a document instance fails to validate, there is something wrong with it, and it will be diverted away from the main workflow in order to be "fixed".

Strict validation is very usefully decoupled completely from specific applications. (The measurements of the parts of the gun may be tested apart from the gun itself.) The effect can be to loosen the bindings between stages of the process or layers of the system, allowing agents to work more independently. (Gun parts can be manufactured in one place and assembled in another.) "Can we process this in our system?" An electronic document, like any other manufactured component, must satisfy strict constraints in order to assure predictability downstream; but if we can validate *apart from* the eventual application, the producer of the document on one side, need not have any knowledge or interest in the operation to be run on the other. This decoupling creates opportunities for reuse: the familiar hub-and-spoke architecture of markup-based publishing systems — with a generic format in the center and different formats for production or interchange on the outside — becomes practical. In many cases, validation is therefore useful (as has not escaped notice) for specifying contracts, as the mechanism for a gateway (to an authenticated "safety zone"), or as a "seal of approval".[7]

While challenging to engineer and document, markup-based information systems that routinely subject their data sets to such rigorous specification and testing — and especially when built to standard specifications, enabling them to take advantage of commodity tools — have again and again proven to be both scalable, and more flexible over time, than single-layered systems handling media only in presentational (or application-specific) formats.[8] The principles underlying this

7 This was especially the case in systems like those for which markup applications were first developed: a formatter, for example, running replacement macros over a marked-up text, has to work with a narrow range of structured inputs; but by its very nature (it must use available resources to process what's there and not expend resources on exception handling), it is not coded to analyze abstractly whether a given data set's markup conforms to the expected pattern. The advantages of decoupling are here, that limited processing power can be applied strategically to one job at a time. Decoupling provides similar advantages when processing is distributed across organizations or along a workflow or supply chain.

8 Contrasting approaches would be dedicated word processors, which are generally only suited for end-to-end processing by a single person and not for complex editing systems with demanding layouts; or virtually any publishing system for print or the web, which (except experimentally) have only served to automate the very tail end of production.

are exactly those that allow a factory to become more efficient and productive than individual craft workers, once basic problems of workflow, parts specification, machining and conformance testing are dealt with.[9]

Validation regimens are useful (and sometimes necessary) because they stretch processing along a time frame, making it possible to encapsulate tasks, divide labor into roles, and systematize and routinize processing. In an automated system in which a document may take many forms in its passage — from authored drafts to editorial cuts to assembly to formatting and presentation for many media, through a range of various post-publication transformations including indexing and aggregation, only after many changes to end in the morgue or archive, or perhaps never ending at all but persisting as part of the cultural currency, like Shakespeare's plays or Lincoln's Gettysburg Address — dependable processing could simply not happen without validation. Albeit informally and manually, it happens in paper-only information systems all the time. Appropriate validation is exactly the practice that makes it possible not to be applying human intelligence repeatedly to mindless processing tasks, or to resolving (inefficient) decision-making tangles. Large complex systems learn this the hard way, even when they have lots of cycles to burn. Validation allows human intelligence to work better because it only has to concern itself with one set of standards at a time, not with all standards, for all conceivable uses or needs, at every point in the process.

Finally, by supporting interchangeability, external means of validation provide a foundation for an entire economy or even (at the grandiose extreme) for "information ecologies", because they introduce network effects among applications. So we see that XML applications, unlike older applications based on proprietary formats, work not to compete with each other, but rather to complement one another, since each can work in different ways to support a common data set. Accordingly, the usefulness and value of the entire information system (and thus of each application within it) goes up exponentially with the addition of each new application. This is exactly what happened when, for example, gauges of wire or threads of screws were standardized; and it is what is happening today with data encoding technologies.

This is the compelling and overarching benefit to standards-based validation, and it has been provided as a rationale for the deployment of DTDs (document models) in markup systems since their inception.[10] But it is not the only conceivable way of applying, or reason to apply, validation techniques to encoded data.

---

9 It may be that this can even serve as an indicator of those kinds of processes that are receptive to automation. For example, in the case of elementary education, can we define "workflow, parts specification, machining and conformance testing" sufficiently to automate it? Do we want to?

10 See, for example, [Goldfarb 1990]. SGML DTDs provide much more than a model against which an instance could be validated: by indicating tag omissibility, SGML DTDs (along with their associated SGML declarations) also give critical information about how lexical information in an instance (or the lack

Validation as discovery: loose validation

The usefulness of a validation regimen in framing a clearly-defined workflow makes an extremely compelling case for it. But a gauge that can be used to judge a piece of work as pass or fail, can often be used as easily as a measuring device. The same techniques (parsing or querying an instance, comparing the instance to a model) can be used in a more flexible kind of application. Preceding the operation of judging, is the operation of observing. What can we see about this data? Where does it fail to conform to a given pattern? Validation is essentially *analytic*: data may or may not satisfy given constraints; but our exception-handling may be permissive. In contrast to the use case described above, I call this kind of validation "loose". Note that it is the means of application that is loose or strict, not the routine in itself (whether it be referring a document to a DTD, an XML Schema or what have you[11]) — although typically, it may be expected that the type of processing in a loose routine may be less comprehensive, but possibly more narrowly focused, than a strict routine, and so, accordingly, that some tools will be better suited for the work than others. Such suitability stems not from any fundamental differences in technology or methods, but rather from the relative adaptability of different tool sets to the different requirements we seek to address with them.

In other words, we are not using a validation mechanism — a DTD, a Schema, a specialized processor — as a simple gauge. It may be more like a caliper or a scale, a measuring or reporting instrument.

It is possible to envision, in sophisticated systems, looser routines combined with stricter tests. Validation routines may even be connected in series or staged from looser to stricter. But in its purest form, we might expect "loose validation" to be most useful in an altogether different setting.

In a paper delivered to the 1998 *Markup Technologies* conference, David Birnbaum sketched out such a scenario [Birnbaum 1998]. Birnbaum describes an application in which an historical edition of a dictionary is being encoded in SGML, posing a dilemma for the encoder when the dictionary violates its own structural conventions. Does the encoder intervene editorially, changing the text

---

thereof) is to be interpreted. This, however, can be taken to be secondary, as such lexical optimizations are only possible given a deterministic element structure. By disallowing tag minimization, XML reduces the role of the DTD almost to its core, to provide a gauge or pattern for testing the element structure of an instance against a set of external constraints.

11 In XML terms, "validation" is necessarily strict, and with respect to a single given DTD (named in the DOCTYPE declaration). In XML terms, "loose validation" is a contradiction in terms, and it might be better if one were to speak of querying, structural pattern-matching, etc.

Likewise, just as in XML terms "valid" is itself a binary condition, it may be useful to consider "strict" an absolute in this respect, so that one would not say, for example, that one querying or type-checking regimen is "stricter" than another. Rather, *strict* would by definition mean "either acceptable or not"; whereas *loose* would be *any* routine in which a question may be raised whether the document should be rejected or "corrected", or some alternative course taken.

Figure 2 | Brown and Sharpe 599-100 0-1.2" Digital Micrometer

This model is available with an RS-232 port. High-resolution image provided, with permission, by Brown and Sharpe, Inc., of North Kingstown, RI (`http://www.brownandsharpe.com`).

to fit the normal model? This would be unacceptable for a project given to representing the record, not changing it. Does he relax the constraints of the DTD? Then he loses the capability of modeling properly the majority of the dictionary entries, which are structurally conventional. Does he model the exceptions in a parallel structure? This is possibly a workable compromise, but is less than ideal inasmuch as it is precisely that the anomalous entries are structurally exceptional, that the encoder wishes to trace. "We are not conditioned to think of syntactically invalid SGML as a natural or desirable state, or as a practical or appropriate way of representing syntactically contradictory source data", remarks Birnbaum. He concludes that markup-based systems could be far more amenable to the special requirements of scholars working on legacy texts, if they had some capability to handle structurally invalid markup, at least in some kind of transitional mode.[12]

How should we call this approach to markup? The primary goal of markup in such an application is apparently to describe a pre-existing object. In the extreme case, the objectives of future processing (or, more narrowly, of certain kinds of future processing) might be postponed; at any rate, the purpose of the

12 Birnbaum also explores the issue in an earlier paper, arriving eventually at a moderate position: "I do not advocate, of course, that we prepare and publish invalid SGML, or that SGML processing software be enhanced to react affirmatively not only to valid SGML events, but also to SGML errors. But I would suggest that when we perform document analysis on existing texts, we recognize that some oddities may at least logically (although perhaps not practically) be represented not as document structure, but as violations of document structure." See [Birnbaum 1997].

markup is to identify and trace those features of the text as object, that are interesting and important to the encoder. We might like to call this kind of markup "descriptive" — but since that term has already been appropriated for an entire species of markup applications that do not take such a radical position, I propose the terms *mimetic* and *exploratory* to distinguish it.[13]

Now, it should be admitted that in its pure form, exploratory descriptive tagging would be somewhat paradoxical, since the effort is clearly given to tracing textual features precisely so that patterns, as well as anomalies, can be recognized and exploited — in principle, recognized and exploited by automated processors (or we would be using a pencil to do the work). Even if the primary goal is to describe something pre-existing — if need be, to develop a language capable of such description — it remains the case that the goal of *this* activity may be to make automated processing over this description possible. Or is the latter goal the subordinate one; do we want to automate our processing only in the interests of a more exact and complete description? An answer to this question is very rarely stated explicitly. The potential stress between these objectives is something we will come back to.

Still, the idea of approaching a text and doing a direct, ground-up development of a set of markup conventions, without any great concern either for processing or for standards, has its appeal.[14] So, for example, it is not difficult to imagine how a scholar might go about creating a marked-up version of a literary anthology — only to change the markup and adapt it frequently, so frequently that it becomes impractical to track innovations in markup with a formal model. Different poems would have different features marked up. There might be stylesheets and processors that work on the material, but no explicit model that constrains the entire thing. The markup would be more in the way of a running commentary and apparatus, than it would be a single system bound to processing in a particular way.

---

13 "Mimetic" in that it aims to "imitate" its source, and "exploratory" in that its design is adaptable. The term "exploratory" was suggested to me by Geoffrey Rockwell, of McMaster University, who attributes it to John Bradley (of King's College, London): "One of the things that struck me about COCOA and XML is that in certain situations you don't know what the final hierarchy will be. In the early stages of markup of something for study ... you need something flexible and simple like the COCOA tags. At the end you should know enough to reencode descriptively. . . . I think John Bradley has called it exploratory coding. The problem with COCOA is that it doesn't let you make the transition from exploratory to descriptive easily. Ideally one wants something where you can, once you are sure something is fixed, replace it with a robust scheme" [Rockwell 2001]. The encoding syntax COCOA is a very flexible, non-hierarchical (stream-based), event- or milestone-driven markup scheme, interpretable by several early open-format text analysis packages.

14 Nor is there any reason why this couldn't be done with XML ... such a project would reverse the usual order (design and DTD development first, then mark up the texts), and concentrate on transcribing an analysis of text *in the process of analyzing it*, then working over the markup to recognize patterns and locate points of interest. Any models would only emerge later. Having introduced the notion of well-formedness, XML should be very well suited for this.

In contrast to more familiar kinds of markup, it is worth noting two particular aspects of exploratory, mimetic tagging. First, in this kind of work, the tagging comes first, the modeling later — if there is a model at all, it is subordinate to the tagging in the instance: it merely describes it, never dictates to it, and is not deployed as a way of introducing constraints, except provisionally. Second, in this type of tagging, there is no question as to what the markup describes (instance or model): it is always the instance. The model does not exist *a priori*, but rather only as a (second-order) description.

Interestingly, such a strategy seems to have been part of the original intention, at least among some of its developers, for TEI tagging:

> A balance must be struck between the convenience of following simple rules and the complexity of handling real texts. This is particularly the case when the rules being defined relate to texts which already exist: the designer may have only the haziest of notions as to an ancient text's original purpose or meaning and hence find it very difficult to specify consistent rules about its structure. On the other hand, where a new text is being prepared to an exact specification, for example for entry into a textual database of some kind, the more precisely stated the rules, the better they can be enforced. Even in the case where an existing text is being marked up, it may be beneficial to define a restrictive set of rules relating to one particular view or hypothesis about the text — if only as a means of testing the usefulness of that view or hypothesis. [Sperberg-McQueen 1994]

Note that in this view, validation is a means not only of testing a text, but also of testing the model that (provisionally) purports to describe that text.

After everything, exploratory markup will be difficult to justify for most applications, especially over the long term. Since it does not rely on or stress methods of strict validation, it does not share in the virtues of scalability. Likewise, it is difficult to envision how it could be conducted except by practitioners who are expert both in markup technologies, and in the specialized subject matter they are treating. As an instrument of analysis and representation of a literary text, however, this kind of technology would have great potential.[15] And it is not only the literary scholar who might be interested in this avenue of approach, using document markup in a new way. It could prove to be a useful methodology in psychology, sociology, economics — any study with a complex and manifold data set — and a source of hitherto-unthought-of ontologies and modeling techniques.

15 A fine example of a project of this kind is Willard McCarty's *Analytic Onomasticon* to Ovid's *Metamorphoses* [McCarty 1999]. The design of the (non-XML) markup is unique and especially suited to the indexing and tracing of interconnections that McCarty has developed for this poem. In the end, the markup will validate to its own kind of model (its own set of gauges). But this is a case where exploratory markup has grown directly into something more "procedural" (or at least application-bound).

Mapping the territory

Apparently there are two kinds of descriptive markup: the classical form (what I will identify as "generic" markup) which works descriptively but which is aimed at future processing, and what may be called an "exploratory" approach to markup. In practice, the difference between these is primarily that exploratory markup will not rely especially on strict validation, in particular when the requirements of a strict validation regimen may interfere with the markup designer's capabilities to introduce new terms to refine or extend an accounting, treatment or handling of the text. A more conventional generic language, however, validates strictly, thereby allowing more-or-less dependable bindings to downstream processing. As we turn back to the classic "descriptive vs. procedural" dichotomy, it may be helpful to keep this possibility in mind.

Descriptive markup and validation

Whatever the explanation, it is evident that "descriptive languages" work (meaning, this time, generic languages but not their exploratory cousins). It is possible, and at times highly practical, to have a formally-defined document type that provides considerable advantages for processing — because it admits of strict validation — and yet, that works by describing an abstract model rather than by committing a data set to one or another kind of processing format. In other words, although there is an inherent stress between, on the one hand, requirements for, or intentions or biases towards the kind of consistency enforced by strict validation (a consistency that lends a data set to future processing), and on the other, to the backwards-looking interests and tendencies of text description — although these purposes are sometimes at odds, nonetheless they are not so mutually incompatible that a workable compromise, taking advantage of the capabilities of either, is not possible between them.[16]

Generic markup languages occupy exactly this middle ground between being bound to a certain kind of processing (the "procedural" side), and very loose languages (maybe they are merely markup conventions or practices), that have great freedom to trace their subjects, but that may be hard to deploy or scale up in production — the truly "exploratory" descriptive languages.

Consider Figure 3. In this diagram the exact placement of one or another language might be disputed. At this point, the placement really matters only

16 Again, I do not think this is accidental. Ever since Gutenberg, the automatability of print has been regarded as one of its most important features. Print applications in particular — everything from newspapers to academic journals to catalogs of every kind — have always been at the forefront of automated production systems precisely because the codex has been a successful technology, answering to people's wishes for granular access to information. As technologies of production have evolved, so has the codex form itself — with its headers and subheads, footnotes, indexes etc. — into elaborations that require formal consistency to function. Thus, not just the technical, but the conceptual groundwork for markup-based systems was laid by the evolution of print media.

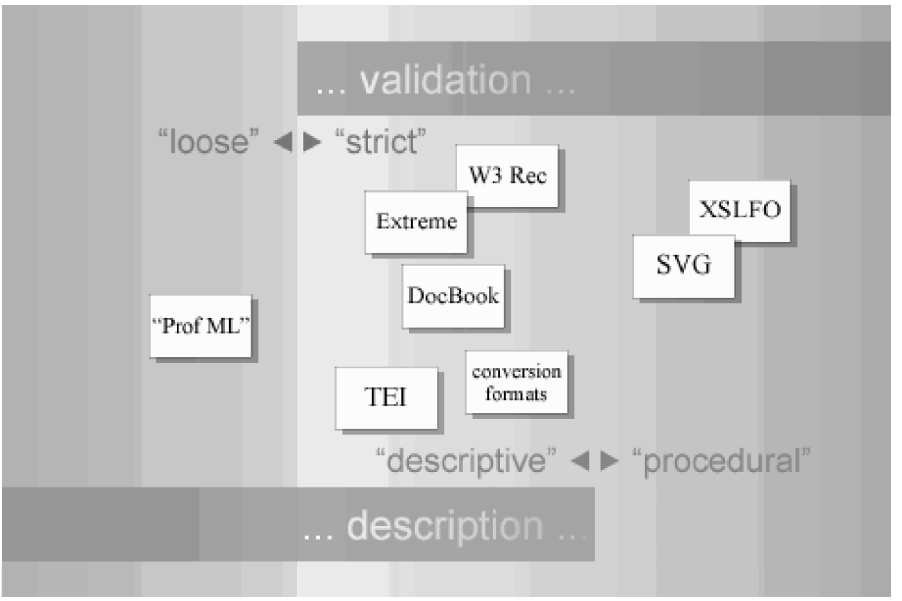

Figure 3 | Markup Languages mapped with respect to validation and description

Strict validation is only possible with a specified set of constraints, so it is at odds with any application of markup that must describe the data with "higher fidelity" than those constraints allow. Yet an in-between zone exists, where formal models provide for strict validation, but are "descriptive" (and so, application-neutral) enough to support a range of different kinds of processing.

along the horizontal axis. (Languages are also distributed vertically, both for legibility and in anticipation of my argument to come.) On the left is a fictional language, "Prof ML", which (we can stipulate) is a set of markup conventions that could be used in an exploratory way. Procedural languages such as XSLFO and SVG are far to the right, indicating not only that their binding to processing is strong (they are expected to be processed one way at least, if not others[17]), but also that if we wish to validate them apart from processing, DTD or even XML Schema validation may not, by themselves, be sufficient. (Both XSLFO and SVG imply, in effect, through their constraints on attribute values, notions of "data types" that are stipulated over and above the constraints on element structure. Whenever an attribute value is expected to resolve, for example, as CSS, XPath or SVG path syntax — all of these amounting to distinct syntaxes apart from the grammar of the document as XML instance — we will need more than a DTD to validate.)

17 A procedural language could in fact target more than one application. XSLFO, in fact, verges on this by targeting on-screen display, print, and audio output.

This diagram also dramatizes how, when strict validation regimens are introduced, there is also necessarily a shift in emphasis for design. On the left side, models are probably informal and implicit in the documents (since if we are not validating, any model must be provisional); whereas as we move to the right, models will become formal and explicit (in the form, say, of a DTD or XML Schema); so a generic descriptive language that validates, ends up describing not the text "as in itself it really is", but a theory about, a model of, the text.[18] To set out to describe "the text itself" runs the risk, at least, that in the long term validation will fail on us, as the model fails to "flex" to the ever-open possibilities for new description.

There will always be a tension, in some ways irreconcilable, between the impulse to fit and form a text, or a markup language, to the peculiar circumstances and opportunities of the moment, and the attraction, and profit, of submitting ourselves to a regimen good for all time. How to position our design between these poles, is what we are determining when we try to "tune", as it were, the level of abstraction of a markup language: we are determining to what degree and in what respects it will be flexible, in what respects specific.[19] But regardless of whether the underlying rationale is a fiction or not (the notion that there is one regimen of tagging that is good for all time — for some more narrowly scoped tasks, it may not be a fiction at all), there is a kind of genius in exactly that rough level of validation achieved by SGML DTDs (of which the XML DTD is, for these purposes, a more refined form). Enough structure is there to support workflow-based go/no-go tests; yet the models are semantically opaque enough[20] to work generically. This allows SGML- or XML-based systems to occupy a middle zone, validating up to a useful point, but also having enough flexibility to work, albeit fairly roughly (only one hierarchy, etc.), "descriptively" — at least when the tag set is well designed. That it is not truly exploratory is something that has occasionally been pointed out as one of SGML's weaknesses.[21] But any number of successful medium- and large-scale systems are demonstration enough that a middle ground is possible — and a rewarding place to build.

---

18 The Greek word at the root of "theory" has a sense of *seeing, beholding*, with an implication that there is some object there to be seen. Once we have a DTD, we actually have such an object. Of course, it can be argued whether a reader or interpreter *ever* encodes anything but a theory of a text; nevertheless, it should be evident how the necessity of modeling in a certain way, would influence the direction of what (and how) the text is theorized to be.

19 Here my argument has been anticipated by Liam Quin. See [Quin 1996].

20 Robin Cover ([Cover 1998] and [Cover 2001]) assesses SGML DTDs as lacking in semantic transparency, therefore inadequate for many modeling functions. But (as I will argue further below) the DTD's semantic opacity in this sense, is actually of benefit for certain kinds of systems.

21 See, for example, Ian Lanchashire's comments in [Lancashire 1995]. At that (relatively early) time, Lancashire's critiques addressed perceived shortcomings in both SGML and TEI, without always being clear which is which. But many or most of his arguments would have been neutralized if TEI tagging could be something closer to exploratory (which, given the role of the DTD in SGML systems, it could not have been).

Adding another dimension

When Allen Renear examined these questions [Renear 2000], he came up with an analysis of the problem with several points of contact with mine. The gist of Renear's argument can also be presented as a diagram.

Note that Renear was not concerned to examine the role of validation in these systems, so his horizontal axis maps only roughly to mine, distinguishing only between different "domains" which a markup language might address. But I think it is not unfair to relate a discrimination between logical and renditional domains, to a distinction between the kinds of constraints each domain may be expected to introduce, and the conditions of their introduction — even apart from the semantics those constraints imply. Whereas a renditional domain must, in the end, "validate" in its application (either the stuff formats properly, or it does not) — and whereas it is likely that in order to do so, some markup semantics may need to be observed that are outside the scope of DTD-based structural validation (so that a DTD-based validation regimen would need to be supplemented to be complete) — the "logical" domain, on the other hand (especially as it concerns what Renear describes as "content objects") might well be defined in such a way that a DTD is sufficient to describe it.[22]

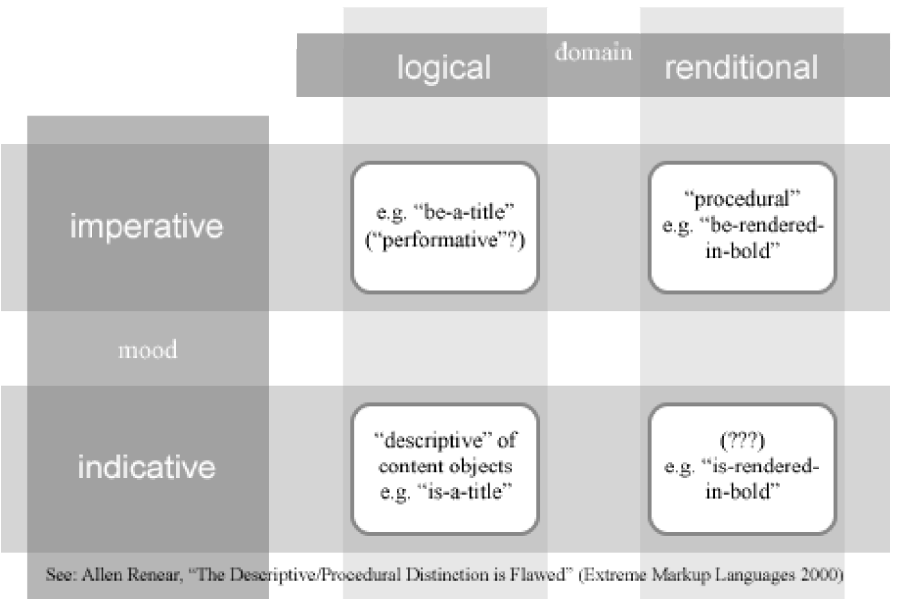

Figure 4 | Renear's Map

Allen Renear's speech-act linguistic analysis of markup languages. See [Renear 2000].

22 Recollecting Robin Cover's argument about the semantic capabilities (or rather, the lack thereof) of SGML/XML ([Cover 1998]; see also [Cover 2001], it may be that we have here a case of the tail wagging

What my analysis adds is the suggestion that to bind a tag set to a particular kind of processing (whether it be in the "renditional" domain or not) implies both strict validation, and a range of other considerations and constraints such as data typing or referential integrity between elements (which may require more fully-featured validation mechanisms than DTDs alone); whereas to work in the "logical" domain puts us in a relatively free in-between zone, where validation provides us the benefits of predictability, control, and a model-centered design, but where the semantics of the markup itself does not rise to the level of specifying behaviors (without the kinds of mapping or augmentation that are provided by stylesheets) — thereby leaving it to be "clean", "logical" and "generic".

But Renear's strongest contribution is in adding a dimension we have not really attended to. By discriminating on a second axis (I have made it vertical) between "indicative" and "imperative" (or "performative"), Renear isolates a very useful axis that had gone pretty much unnoticed. (I believe his basic proposition, that the descriptive/procedural distinction has served to mask this dimension, to be essentially correct.) In my diagram we might notice, for example, that notwithstanding the apparent advantages of generic markup, it is still evident that there is a clear difference even between (say) the W3C Rec document type (the DTD by which W3C drafts and recommendations are marked up), and (say) TEI markup. In a sense both may be considered to be descriptive: but it still seems significant that one presumes to describe something that already exists (TEI documents usually purport to be faithful representations of texts already extant in print or manuscript), whereas another (W3C Rec) describes something that never exists apart from its tagging (or in products derivative of that tagging), to be created and then maintained in that form.

While Renear himself is not altogether satisfied with the categories he proposes,[23] it is evident that either or both "imperative" and "performative" can provide the necessary distinction from the opposite term, "indicative".

To reduce this to its essence, it appears markup can look "into the text", or "out to the application" (this would seem to be a very loose way of characterizing our old friend, the descriptive/procedural distinction); but it can also look forward in time, to eventual processing, or it can serve, irrespective of application,

---

the dog: if the semantic expressiveness of DTDs were richer, the "logical" domain could be accordingly more fully-featured. Models would be more directly tied to processing semantics — and we would not have had the same chance to learn the capabilities and occasional advantages of the looser coupling between model and application that the logical domain implies.

23 Although imperative and performative moods are supposed to be distinct in the scheme Renear proposes, in his treatment he is not quite able to clarify why the mood of a "renditional imperative" and a "logical performative" (a bit of markup that makes something a title, say, by so labeling it) should be considered to be different. I submit that the difference is one of agency. An imperative is spoken by one agent, to be performed by another, whereas a performative is something that is done in the speaking of it. But, when applied to markup languages, this in turn raises other questions: is such agency a property of the language itself, or is it determined by the design of the architecture in which it functions? In linguistic terms, the "pragmatics" of the situation are entirely different.

to represent some state that pre-exists, for example in a document already extant. While it might be tempting to call the latter kind of markup "descriptive", this requirement is in fact orthogonal to the requirement for application binding we have been examining so far. Renear's major contribution, by identifying a kind of markup in the logical domain, but the imperative or performative mood, is to show that descriptive markup (in the traditional sense of the term) can in fact look either back, or forward. In fact, many or most of the current initiatives in XML languages are of exactly this forward-looking type. The markup serves descriptively, but only to describe the text's content with respect to a logical model, designed to be amenable to some particular kind (or some range) of processing. This is quite a different thing from using markup to describe some extant artifact in the world. A confusion over the stresses between the two views is at the heart of many design problems and infelicities.

We can adopt this point of view in developing our map of markup languages: one way to name this new axis is between "prospective" and "retrospective" markup languages. A retrospective markup language is one that seeks to represent something already existing; whereas a prospective markup language is one that seeks to identify a document's constituent parts as a preliminary to further processing. Prospective markup, that is, may be "procedural" in the sense that SVG or XSLFO is. Alternatively, it may seek to claim all the advantages of generic markup (scalability, strict validation, content re-use etc. etc.) without having to be bound to describe anything apart from itself.

In my map (Figure 5), this could be distinguished by a vertical axis, "prospective" corresponding to Renear's imperative/performative mood, "retrospective" corresponding to Renear's indicative; but it is interesting to see that when we begin to place actual markup languages into this conceptual space, that there are blank spots. In particular, there are two positions left empty in a possible grid of six (we can conceive of Renear's arrangement with a new domain to the left, "exploratory/mimetic", next to logical and renditional to the right). For one, it seems unlikely that we would have an application of markup that is both prospective (Renear's imperative), but exploratory, having no use for validation or the kind of binding to (even implicit) semantics that validation implies: if we are creating a new format for a new application, what does validation lose us? It could be that markup instances that are purely *ad hoc* files for momentary processing, would fall into this category.[24]

Equally unlikely would be a conjunction between retrospective and procedural (or application-specific). This would correspond to Renear's category of "indicative renditional", which he also remarks would seem to make little sense.

24 I actually think there is an important role to be played by such little languages, exploring not artifacts or texts, so much as processing opportunities.

Evidently, procedural and retrospective markup serve requirements that are in conflict. We can either describe the world as we find it (with retrospective markup) or we can dictate in what way we need our data to be handled (with procedural markup). The fact that traditionally, generic markup systems (or at any rate, those that had retrospective designs) have sought to mediate this exact conflict, does not make it any easier to do so. The more we need our application to serve retrospectively, the less we can expect to find thorough, detailed and strict validation regimens of much help.[25]

That is, although we can distinguish a vertical axis that indicates a markup application's orientation in time (forward- or backward-looking), it is clear that this axis is not completely orthogonal to the spectrum of loose-to-strict validation that I began by tracing. It is likely that a prospective application will find strict validation both useful, and not particularly burdensome. To the extent that an application is retrospective (such as might be the case with a markup language written to support conversion of a legacy data set, or a scholarly project in textual editing), however, it may prefer any testing to be loose. In graphing it out, therefore, this axis appears on a diagonal.

Generic markup as a form of rhetoric

Prospective, procedural languages clearly have a place: it would be hard to argue against the utility of standard XML vocabularies such as XSLFO and SVG.[26] At the other extreme, retrospective, exploratory applications of markup would appear to be very fruitful as approaches to certain intellectual problems (although until it became practical to develop markup applications without DTDs, this kind of application of technology was severely hampered by a lack of a standard toolset), particularly problems that have directly to do with questions of how we represent non-digital phenomena by digital, processable means. But what is most interesting here is the broad grey zone between these extremes, a zone occupied by applications of markup that have a need for strict validation as an instrument in workflow and processing architectures, but that are not exclusively bound to any particular type of processing or application, as would be implied by a procedural language. This is the zone of "generic", loosely called "descriptive" languages such as TEI, W3C Rec ML, or for that matter, the language used to mark up this paper.

25 Nevertheless, applications like this are conceivable, and have even been executed in part. For example, if an attribute syntax were to be adopted on top of a generic markup like TEI, especially if the attributes worked to prescribe formatting (embedding, as it were, a style mapping into the generic instance), it might achieve something like this.

26 In fact, as for example in the "XSL Formatting Objects Considered Harmful" argument [Lie 1999], when these languages come in for criticism it is precisely because they have certain kinds of utility (though perhaps not others).

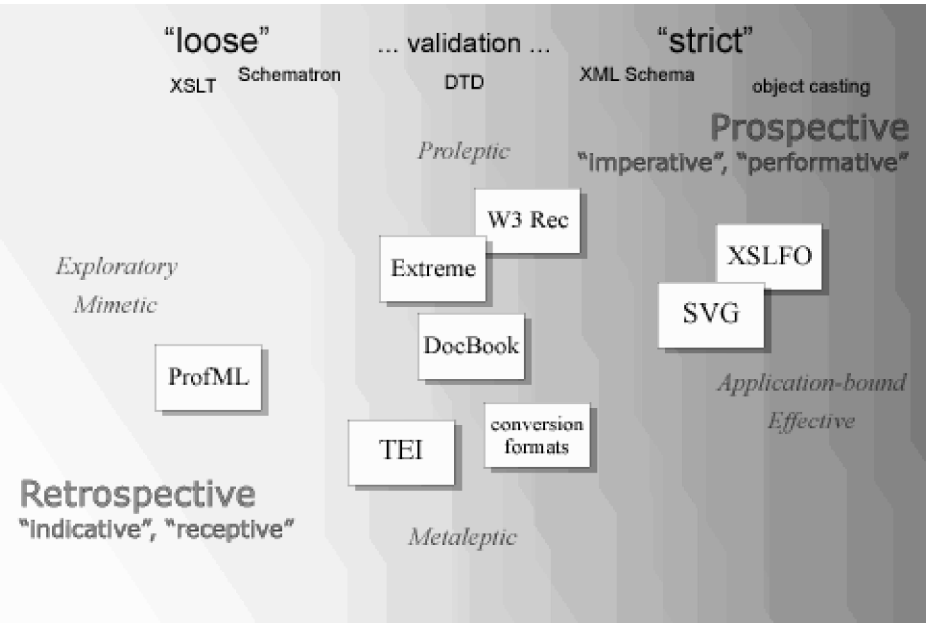

Figure 5 | Markup languages mapped on two axes

The horizontal axis represents the level of validation appropriate or called for, and thus the specificity of machine (behavioral) semantics. Requirements for a tag set to be prospective (provide for future use) or retrospective (describe a given artifact physically or "logically") align along the diagonal lower-left to upper-right.

Most discussions of "semantic" in the context of automated text (or "knowledge") processing end up having to distinguish between two meanings. There is the realm of human semantics, largely if not principally *representative*, our "meanings" when we express ourselves in language or by any other means. Then there are machine semantics, the sorts of behavior, events, products or controlled processes that can be expressed through a machine — and which are the normal objective of workflow-oriented systems. If you like, you can consider this a spectrum between *word* (on the "human" side) and *act* (the machine behavior). (Tim Berners-Lee, for example, in his discussion of the "semantic web", has openly affirmed that he is concerned only with the second kind.)

The world of "content" (text) that is encoded generically is a fascinating one, in which these two competing notions of the semantic discover themselves head-to-head. In this world, markup simultaneously links people and processes in different roles, and serves as a conduit or channel for "meanings" that have the interesting property of skipping or passing through stages in a process (a "supply chain") that can go directly from creator and producer, to audience or consumer. That is, markup provides a kind of framework or packaging by which words

(written texts or representative codes) can be passed without consideration of what they "say". As a kind of interchangeable part, as long as the package or framework is correct, the meaning or substance of the "text itself" (what we call the "content") can be more or less completely opaque to participants along the chain. The framing or wrapping provides the text with sufficient information (about its nature, about its internal structures and relations) that it can be passed and processed without constant rediscovery and reinvention. This wrapping or packaging takes the form of markup; and of course, relative to the processes, the markup is meaningful — yet its meaning is local and provisional. Accordingly, these can be staged systems in which interpretation happens in an articulated way. For example, authors decide some things, bibliographers some things, catalogers some things, layout designers others. In such a system, a degree of "semantic opacity" is a *feature* (cf. [Cover 1998]), allowing us to provide appropriate processing based on some kind of "intention" as a tag set presents that, but always leaving it up to us to decide finally what that means. The masking or withholding of any details or specifications for processing, that is, behind "element type names" or "generic identifiers", is part of what makes such a system work: by not insisting on a binding to any one thing, the mechanism of element typing is free to support an open-ended range of things. Note that exactly insofar as machine semantics is devalued (or rather, postponed or layered) in such a system, the expression of human semantics becomes very important: generic markup languages become worse than useless if their tag names are cryptic or if they are not well documented. But when a markup language is designed well, it can be used to frame and drive a process in which different participants can provide their added value, each without having to get involved in exchanges of no direct concern to him- or herself.

In these respects, a generic language is able to be, or to pretend to be, exactly, *descriptive* or representational — meaning that, pragmatically, it has some kind of implied human semantics, without being bound to any, or any particular, processing (machine) semantics. That is, we take the tags to "mean" something — but what they actually mean, in the event a file is ever processed, may be different from (albeit in some way implied by) the meanings of the tags. In other words, there is a *slippage* between what a descriptive tag set purports to mean, and what it actually "means" (*does*) in the event.[27] This slippage is the

27 Robin Cover argues [Cover 1998] that this makes it important to provide XML with a means of strong semantic specification, which in and of itself it does not have (since XML syntax, nor DTD-based content modeling, are incapable of providing it). In Cover's terms, this is XML's lack of "semantic transparency". And for procedural applications of the syntax, this is certainly a critical issue. It can be addressed in several different ways, for example by providing some kind of formal ontology; by merely presenting a notation for some other data model; or by passing the problem into a syntax carried in attributes, such as CSS, XPath or SVG path syntax. Yet for descriptive or generic applications, XML's semantic opacity is actually be a feature of the technology. It's where things can get slippery between layers.

source of the power of descriptive languages, their famous "indirection": meaning nothing directly, they can be taken to mean a great range of things if we only bind their evident and ostensible meanings (that in practice do nothing but structure and disambiguate between types) to behaviors. To tag a data element as a `title`, say, may mean nothing more than "whatever you do with titles, do it with this thing".

So generic markup involves us in a strange paradox. It foregoes the capability of controlling behavioral "machine" semantics directly, but wins, in return, a greater pliability and adaptability in its applications for human expression. This kind of middle-ground markup would be systematic enough to be receptive to automation, but would not necessarily be automated "out of the box". Another way of describing this kind of markup application, as opposed to more strongly typed and validated kinds, is that this is the kind of system in which a stylesheet writer has something significant and important to do. Stylesheets are a natural way to get from an abstract model, into an application. But they might require, as stylesheet writers know, some addition of information, interpretation and restructuring, as well as mere mapping. Stylesheets are also where a great deal of creative work can come into play.

If this variety of markup language is not really a set of instructions, but a complex representation (on which a later process may be expected to act), the proper discipline for regarding it would seem therefore to be, not formal languages (that have the virtue of being readily bound to processing), but something closer to linguistics and rhetoric.[28] This is the realm where we experience slippages — whether inadvertent, or "intentional" — between actual and potential meanings.[29]

In effect, markup languages are far more than languages for automated processing: they are a complex type of rhetoric working in several directions at once, often in hidden ways. Inasmuch as markup systems then may begin to resemble other textual systems (such as literary canons or conventional genres), it is reasonable to turn to rhetorical or literary critical theory for explanations, or at least

28 Lately, Michael Sperberg-McQueen, Claus Huitfeld and Allen Renear have sought to formalize markup languages' (including generic markup languages') handling of meaning by saying markup "licenses certain inferences" about a text. (See [Sperberg-McQueen 2000].) In the notion of *inference* — and the evasion of the issue of how an inference can be constrained or defined (since isn't an inference precisely that kind of communication that can't be constrained or defined?) — they effectively elide this transition between formal information theory, and rhetoric (which is enamored of formalisms, but resists being comprehended by them). To "license an inference" is, in effect, to say something without saying it. Is this logic, or rhetoric?

29 To examine this in the context of Renear's categories: one difference between imperative and indicative, or between a "performative" and an indicative (the axis Renear describes as "mood: whether markup describes something, or requests processing" [Renear 2000]), is that an indicative refers back to the past (or disinterestedly to the present or future). It is the projection or implication of some reality apart from the markup (the separation of format from content!), whether this is a feature of some kind as documented, a perception, or an imaginative projection, which competes with processing objectives, that opens up the important area of slippage.

higher-level characterizations of them. I am not going to begin to plumb the depths of this subject here. Given both the complexities of real-world workflows, and the fact that many of the agents are human beings only as mediated through their machine proxies, it is difficult to say who is saying what to whom through (and in) a markup language or markup language application. Then too, the ways in which messages and meanings trace through an electronic text system, is going to be highly, sensitively dependent on the unique particulars of media, technology and culture at work in a particular case. One thing that does need to be observed here, however, is that in markup, we have not just a linguistic universe (or set of interlocking linguistic universes) but also a kind of "rhetoric about rhetoric". That is, markup languages don't simply describe "the world" — they describe other *texts* (that describe the world).

As it happens, critical theory has had occasion to consider such complex types of figuration, representation or meaning. I am going to draw on the work of scholars who have studied intertextual referentiality[30] (where this type of phenomenon is especially pronounced), to distinguish between the tropes *metalepsis* and *prolepsis*. These are distinguished from the usual run of rhetorical figures such as metaphor, metonymy and so forth, because unlike others (which are occasions of figurative representation), these are tropes about tropes. It is not "something in the world" that is represented in a metalepsis (or its less common complement, prolepsis), but rather some other act of figuration.[31]

Proleptic markup

Of *prolepsis* and *metalepsis*, the first is possibly simpler to grasp quickly: Prolepsis is a rhetorical trope or gesture[32] in which an expression or figure of speech takes its meaning from something that is to appear later. Dramatic irony (where a character in a play, for example, says something that carries an extra meaning to an audience that knows or guesses what is to happen in the drama),

30 In particular, on the work of the poet and literary scholar John Hollander [Hollander 1981] and his colleague, the critic Harold Bloom [Bloom 1982].

31 Hollander (and with him, Bloom) claims that this type of thinking is to engage not just in the usual kind of "synchronic", but a "diachronic" rhetoric. That is, ordinary treatments of rhetoric pay attention to the use of figurative language as if all the signifiers were related outside of time. (This would seem to be a Platonistic view of text, with all meanings always available *sub specie aeternitatis*.) But it is possible, not only to consider how language or signification interacts as a kind of "random access" system, but also to think of how meanings work over time and across it, how they shift and change in relation directly to one another, how they recapitulate or anticipate. This kind of thinking is extremely helpful as soon as we start looking at layered systems and complex, dynamic information interchange — but it involves us, in effect, in representing the flow of information, the stages of its passage.

32 The extremely useful word "trope" may call for some explanation. From the Greek for "turn", it is a traditional word to designate a figure of speech or signification (whether spoken, written, or by some other means), or any occasion when something is expressed by saying something somewhat different. Metaphor is a trope, though its cousin simile (a comparison using "like" or "as"), even when poetical, is only a trope in a loose sense. Other tropes include metonymy, synecdoche, irony, etc. etc.

or literary or dramatic foreshadowing, is prolepsis; but so is any "casting forward" or anticipation, such as an argument one might make in a conference paper in anticipation of counter-arguments. Consequently, the full meaning of a prolepsis is impossible to know without taking account of its relation to the future. Whether what is forecast does, in fact, come to pass in the way forecast, opens prolepsis up to capabilities for irony. On the other hand, sometimes saying something, makes it so: so prolepsis often has the capacity for a kind of poetic "fiat" or self-fulfilling prophecy.

Any prospective tagging might be called "proleptic" because the meaning of the tagging is intimately connected with our expectations for processing it. Even when such markup is generic, we call something a `head` or a `section` because we intend to treat it as a head or a section in processing. This is Renear's "performative" markup: the section becomes a section through the act of naming it so.

But it might also be that the term *proleptic* would be useful to distinguish exactly that type of prospective (performative) markup that works generically, such as DocBook, the W3C "XML Rec" markup, or even certain kinds of XHTML (probably "XHTML Strict"), as opposed to prospective markup that is merely, in effect, an application binding, such as SVG or certain other kinds of XHTML (such as a DHTML application, heavily laden with script and tuned to a particular browser). Admittedly, this too may be a spectrum rather than a simple either/or classification; also, it should be noticed how a markup language may actually "grow into" an application binding — or conversely, how an application binding or API may grow around a markup language.[33] Nevertheless, there will be occasions when, although we have expectations for processing our data, they may not be specific or limited expectations. In other words, we want a method (a generic language) that affords us that slippage between specification and processing. The word "proleptic" seems to allow for this: as a trope, the meaning of a prolepsis has to be seen as conditioned by the possibility, at least, that things don't quite turn out as expected. Especially when marking up new texts (or composing texts in a new language), this is a very powerful way to approach the design and practice of markup: an artful combination of specification and slippage is what enables most of the promises of generic markup to be realized.[34] When we design, we may want to know in detail (or at least in principle) the application requirements of a markup language; we may want to be prospective if not actually procedural. Nonetheless, we always want to keep our eyes also on the bigger picture, since a careful restraint devoted to modeling our information "logically" (that is, in some sense, descriptively, if only to be descriptive of an

33 So, for example, CSS has grown up as an API (in effect, albeit "declarative") around HTML, therefore pulling HTML/CSS further into the procedural than plain "generic" HTML on its own. As an API to a display engine, of course, CSS is useful to more than HTML.

34 Again, see [Quin 1996].

abstract model) rather than in the language actually of an application, pays off in the long run in data independence, reuse, longevity, and so on.

In this kind of endeavor, validation routines are going to be very useful. We will build our workflows around them. More interestingly, possibly, our means of specifying validation, such as DTDs, will be useful as specifications for tools, many of which can be automatically fitted to the task. This is an application of a gauge (the DTD), which is used to check conformity to an external measure, as a jig — a device or tool fitting, that allows us to make the component to measure the first time. In a sufficiently evolved production system, we may never even have documents that are "invalid" in the XML sense, and we may have needs and uses for all kinds of validation besides simple structural element type checking. But we may do all of this without any particular or specific expectations for processing.

Metaleptic markup

So proleptic markup is that type of generic markup that looks forward. What of generic markup that looks at what is past? That is, that tries seriously to register, in some disinterested and objective way, features and organizations of information already out there? In some ways it would seem unnecessary to have to submit a descriptive markup convention to strict validation, with all that implies (we remember Birnbaum's argument [Birnbaum 1998]). Nevertheless, whatever processing we expect to do over data sets, on however large a scale, will demand some kind of validation at some point, and there are many reasons, both intellectual and practical,[35] to try to design a generic language that also tries to capture some "truth" (or at least theory) about the world. Having formalized our theories in abstract models, we can then test them by running the very same validation routines that we apply to encodings that have been specifically designed for processing, not for representation. In the end, validation is not only a testing instrument in a workflow: it is an investigatory instrument in its own right. DTDs are representations of texts. So we look backward, in an interestingly formalistic way. But we also get the benefits of looking forward.

This type of markup tries to be retrospective (and in this presumes to *describe* the data set), but nevertheless relies on, and benefits from, strong or "strict" validation regimens. Such tag sets would include TEI,[36] or any tag set developed for data conversion or retrospective document conversion which seeks

35 Practical reasons: converting large amounts of data from a legacy format. A well-designed model that looks to how that data is formatted, can preserve information through conversion to an open format like XML, and ease the conversion process. Intellectual reasons: develop a theory about a (body of) text; formalize that theory; demonstrate its utility.

36 In most applications. TEI can also be used in a proleptic way, for example when it is used to drive a web site of original documentary materials (a task for which a TEI subset is actually fairly well suited). Notice that it is not a markup language (a tag set) that is *per se* proleptic or metaleptic, but an application of it. Some tag sets can be used in all kinds of ways: HTML certainly has been.

at once to be both descriptive and generic. Markup systems like this are evidently descriptive after a fashion; but it is also clear that their prospective applications, be those presentational, analytical or what have you, are a big part of their conception.

In contrast to prolepsis, *metalepsis* is the rhetorical trope in which the meaning of an expression is in direct reference to what has already happened in the past.[37] Of course, this is in some sense true of all rhetoric, since all rhetoric is situated, in some way, in a moment with a history (and inasmuch as this is the case, all rhetoric is metalepsis, successful or failed); but in a narrower sense, metalepsis is what occurs when reference is made to another*figure* that has already appeared (some event of meaning or figuration that has already taken place), but in such a way that the meaning of the earlier figure is itself changed by the appearance of the metalepsis.[38]

What then would be a metaleptic markup language? Keep in mind, to begin with, that document markup as rhetoric is necessarily complex; there are various levels of expression here. A tag set describes a data set, or it describes a theory about the data set; when it looks back, what does it see? Might it not sometimes have reference to one or more earlier systems of description (earlier theories?), including implicit traditions? A consideration of markup systems actually in use (I've mentioned academic projects including TEI, as well as transition or conversion formats being used in industry), suggests that such a reference is not, in fact, uncommon.[39] In general, metaleptic tagging will act retrospectively, and may even

37 John Hollander, considering metalepsis as a "diachronic figure", describes it as related to allusion but entailing a deliberate relation between before and after. "We deal with diachronic trope all the time, and yet we have no name for it as a class. An echo of the kind we have been considering may occur in a figure in a poem, and it may echo the language of a figure in a previous one. But the echoing itself makes a figure, and the interpretive or revisionary power which raises the echo even louder than the original voice is that of a trope of diachrony" [Hollander 1981]. As a variety of allusion with "interpretive and revisionary power", metalepsis is not any ordinary act of signification or representation: it is a representation with reference to another (previous) representation. Once he has alerted us to this possibility, Hollander is able to show that gestures of transumption (the Latin "transumption", with its morphological variant the verb "transume", has long been a variant of the technical Greek "metalepsis") are in fact not uncommon in literary language. "Save for dramatic irony, with its audience's — or reader's — proleptic sense of an outcome of which the dramatic speaker is unaware, and which engenders an interpretation more powerful than the raw intended meaning of the speaker himself, only transumption seems to involve a temporal sequence" [Hollander 1981]. His fascinating book *The Figure of Echo* contains a thorough examination of the dimensions and history of this category in critical theory. Nor is this conception, concludes Hollander, of application limited to poetic language. "Not only particularly preexistent metaphors, but formal structures — and M.H. Abrams and, more recently, Paul Fry, have shown us authoritatively the intricate turnings of the transumption of a previously public form in the history of the ode — are recreated metaleptically. So are genres" [Hollander 1981]. And so, I submit, are markup languages.

38 The Christian New Testament is metaleptic with respect to the Old Testament. Virgil is metaleptic with respect to Homer, Dante with respect to Virgil. Strong poetry is almost inevitably metaleptic, since poets, it seems, cannot help but echo and try again their predecessors, but in such a way that they commandeer the older works and set them to later purposes. Blake and Shelley succeeded at this so thoroughly with Milton, that we cannot even read Milton any more (if we do at all) without meeting up, in some way, with Shelley's Romantic heresy. This also happens in musical traditions: Brahms is metaleptic (or attempts to be) with respect to Beethoven, and so forth.

39 I suppose any extension of a standard or off-the-shelf markup language might be metaleptic in a simple

pretend, and attempt, to be thoroughly descriptive and retrospective in its relation to already captured information (figures already spoken); but it relies on strict validation. This betrays its true nature: its design and application is really done for purposes of future uses of the data (new meanings), not merely to "describe" in the more limited senses of that term. It is retrospective tagging for prospective purposes: thus, it works by saying something about the past (or about the presumed past), but in order to create new meaning out of it.

Typically, it does this by positing a model of the text and then asserting, implicitly or explicitly, that this model is sufficient for all practical (if not conceivable) descriptions or applications of the text. And in well-designed, mature systems (by which I mean ones which have clarified the way they actually work and are not confusing either their rationales or their design with those of other markup applications), metaleptic languages do in fact function very nicely as generally-accurate descriptions — though it should be added, that when they succeed in this way, it is typically because they determine *not* to try and describe *everything*.

Just like any other future-bound processing, this kind of markup will be able to take advantage of strict, go/no-go validation. Because this kind of tagging often originates as a description of a given artifact (a known text), it is easy to identify it with true descriptive markup. But as I've said, that is a very rare thing (unheard of in commercial or industrial applications): the goal of describing a pre-existing object must generally yield, at some point, to the more practical need to constrain and process the information set. Hence most markup languages that go by the name "descriptive" are only so up to a point — they are in fact metaleptic.[40] A metaleptic markup language (or rather its designers and advocates, perhaps its users) may be entirely innocent of any perceptions of stress between extant documents, and abstract models — fundamentally, the stress over which many struggles over validation will take place. Absent any consciousness of such a stress, a metaleptic design may take, or propose, its model or theory of the text as a kind of reality, thus claiming the title "descriptive". But we can know it for what it is when we see it being validated strictly, and when we also hear, in addition to its claim that it works by description, that it expects all the benefits downstream of validation, in the data set's readiness for further processing (be that publication of electronic or print editions, providing database access, or what have you).

---

way. But more common, and more complex, are cases where the references are merely implicit, if sometimes obvious.

40 In fact, the process of formal document analysis as it is practiced in the markup industry, can involve a complex interplay between an actual descriptive exercise, as a way of driving work and exploring the problem domain, while ultimately keeping focus on requirements for future processing.

So far so good; the dark side of metalepsis is, possibly, when it denies its own complex and layered nature. An act of transumption (a synonym for "metalepsis") changes, transfigures, that which it transumes (in this case, "describes").[41] To pretend otherwise, it would seem — to pretend, for example, that our representations are in all respects (or even all important respects) identical to what they represent — would only have the effect of setting ourselves up for disappointment. In the worst case, we may completely fail to determine our actual needs and fit our design to them: stuck on the horns of this dilemma, we may end up with neither an adequate representation of our source text (however we define that), nor data that is well suited for automated processing.

More commonly, rather than being purely descriptive/exploratory, or purely proleptic, applications adopt a metaleptic design strategy because they need to meet requirements on two sides, past and future. In time, if they are lucky, they grow into a consciousness of their ambiguous status; but the actual rationales, expectations, and design of these projects are often complex and intermixed. Sometimes project participants themselves have not exactly clarified what their main interest is; often they are working with several conflicting rationales or requirements.

Yet in general, the emergence of this type of markup is of great importance because it has led us more quickly and readily to understand the efficiencies, power and scalability of layered markup systems: just like proleptic markup (which is generic without being retrospective), metaleptic tagging is very much at home in such a system of at least two tiers, possibly because it itself has two faces, looking in and looking out.[42] And when they are well designed (which not coincidentally, often means *intentionally* designed) and appropriately deployed, such a markup language can be fascinating in its own way, quite differently from either of the other two forms of markup that are prevalent (leaving aside exploratory markup as more rare than it should be, we also see generic proleptic markup, and procedural applications). It has its own kind of art. It does not try merely to transcribe, as purely descriptive, exploratory tagging would (though as scholars know, "merely transcribe" is an impossibility and an oxymoron), nor merely to function in future systems, like prospective markup (spectacular though

41 In a metaleptic markup language, there is a missing term standing between the language itself, and the text, the presumed "content" that completes (and is completed by) the markup: that term is the theory of the text, the model, that the language formalizes. (Here I am concurring with Paul Caton, [Caton 2000].) It is the movement from one term to the next (here, from text to theory, theory to model, model to application) that makes for the rhetorical complexity of such a language, sometimes most complex when it aspires to be most "transparent" — and that may help make applications of these languages suitable for particularly interesting processing, as being particularly "slippery".

42 This is of course the famous separation of format from content. Two tiers would be the repository and presentation layers (think of a TEI text and its HTML rendition); this also maps over to the model/view/controller paradigm, with "descriptive" or "generic" instance as model, rendition version (say, HTML) as view, and stylesheet or script as controller.

that might be). It aspires to both, by seeking to balance between them. An effective markup language will work by establishing a self-contained, internally consistent and clear set of categories perfectly sufficient for handling the data set to which it will be applied, within the range of applications for which it is due. But this ideal is impossible for a truly descriptive language to achieve, since the world is not a closed, finite set of phenomena that is liable to such treatment.[43] Metaleptic markup gives us the next best thing: it invents its own imagined world, proposing earnestly or ironically that this serves both sides, both accounting for external reality as it is, and creating it as it needs to be.

TEI, incidentally, has occasionally been represented as a true retrospective tag set, yet is torn about the issue. It aspires to provide certain functionalities along with transcription, such as eased production costs for print or online editions, or eased repurposing across different applications, that can only be guaranteed through strict validation. Up until recently (when XML has made processing without a DTD more practical), validation has been a particularly all-or-nothing proposition. New (and newly accessible) tools and approaches supporting "loose" validation may now seem more of an option than they have hitherto (especially to strapped academic programs). Nonetheless, TEI cannot help but continue to be powerfully metaleptic: pretending to be simple, naive, retrospective (and accordingly, extensible!), *and* simultaneously stressing validation as a means of smoothing transitions of its texts to new media and new applications — a *prospective* gesture — it ends up "falling into" metalepsis despite itself.[44]

Finally, it may be worth observing how architectures supporting metaleptic languages or applications will differ from those for proleptic languages. For one thing, we will be able to tell the difference when we look at a project's regard for, and use for, validation mechanisms. For a metaleptic system, validation will need to be strict to the extent that future processing is anticipated. On the other hand, since there is at least a presumed interest in the prior or "original" nature of the textual content's own structures, features or organizations — however these are conceived — it may be at times that the best solution to a misfit between document and formal model is to change and adapt the model (and accordingly, the system) rather than forcing the document. As in pure exploratory applications, markup is designed first, formalized after, whereas in a proleptic system, the model or schema will come prior to the markup. This difference in emphasis may

43 This observation has often been made informally, in a variety of ways. "Selection is easier than synthesis, but the world is not finite", says Brian Reid [Reid 1998].

44 This tension can be seen to play out exactly in the role validation is expected to play in TEI projects. On the one hand, the tag set is provided with an apparatus to support extensibility. This is the promise of descriptive markup: that no text should have to be forced to fit. On the other, validation is considered indispensable, not only for usual quality-assurance reasons, but also because in it there is an assurance (for example) that the rigors of the `teiHeader` are observed, or that off-the-shelf (or nearly off-the-shelf) stylesheets be able to be used — or that interchangeability be achieved (a prospective requirement, perturbed by local extensions). It is, after all, the Text Encoding Initiative for Information *Interchange*.

make for different toolsets, to an extent. Also, we can expect of metaleptic systems, in particular, that the natural stresses between requirements for description (sometimes in the guise of backwards-compatibility) and for interchange, will be at their greatest: balance will only be achievable if we keep a realistic view of what we intend to achieve and how we intend to do it. But when metaleptic systems are well designed, the rewards, both in our mastery of complex bodies of information, and in our understanding of them, will be great as well.

Conclusions

- **"Descriptive" may not be the best word.** It means too many things. Even the procedural languages are descriptive: they describe a binding, API, or object model. The differences are in the closeness of the binding and the extent to which an abstract syntax allows us to validate without binding, hence letting us design a language at a higher level of abstraction (and get capabilities of reuse and refitting thereby). *Generic* is a somewhat more useful term: these are languages that can be strictly validated, but that are only loosely bound to processing. At the far end, markup that isn't validated at all, if it is retrospective, may be said to be descriptive, insofar as it describes some external object (and is therefore directly representational). But historically, no standards have existed to support markup systems of this kind. XML may help stimulate more of this work.
- **Watch out for clashing requirements.** Prospective ("performative") markup can be generic, and generic markup may seek to be either prospective (proleptic) or retrospective (metaleptic), or both together. But the more we try to "describe", the more difficult we will find it to validate (in the broadest senses of that term). We should be careful to distinguish the requirements presumably served by our design strategies. Academic projects with a commitment and interest in description of something external (say, a literary or manuscript text) may have a particularly difficult time with this — for example, when an exploratory design clashes with a requirement for interchange. The "descriptive vs. procedural" distinction can, if we are not careful, muddy the waters here even further.
- **New approaches to design: bottom-up.** Loose validation with Draconian error-handling at the syntactic level (e.g. XML well-formedness) — even if it involves no "validation" at all in the formal XML sense — should open up new possibilities for design strategies and methods, as well as for new applications of markup, including exploratory modes of markup such as I have described. Up to this point the design process for a document model has usually been driven by a top-down analysis, and centered on DTDs. As long as DTDs provide a useful means for testing for the kinds of interchange and

downstream processing that have been prominent requirements, this will continue to be appropriate. But if and as we design systems and markup languages with other aims — such as, for example, an exploratory application rather than a "performative" or direct application of markup to processing — other techniques and approaches may prove useful. What if designs were centered not on DTD validation, but on stylesheets and query sets that provided meta-information (including validation checks) along with or in place of their more usual kinds of transformations? What kind of markup applications would be well served by such an approach?

- **New complications include maintenance and oversight.** Already approaches to XML validation are proliferating. Which of the various approaches now being tried, both strict and loose, come to be prevalent (and which approaches in which environments and domains), is an issue I can't address. But nothing is either/or here: just because we use DTDs or XML Schema to validate one set of features to requirements, does not mean we can't use other means (stylesheets or query sets, for example) for others.

  If and as we do this, however, we should be careful to keep clear what we are doing where, and why. It could easily become a problem if the same set of constraints on a document set, or type, comes to be validated through more than one tool: this would introduce new problems of parallel maintenance. (It would be like having two rulers to measure things, but not being sure they measured the same "inch".) Yet different kinds of validation, and of tools to do it with, might well be very usefully done at different stages of a document lifecycle. (Such routines have been commonplace for years in any case.) When systems become complex and validation routines overlap, it might be helpful to have a "validation validation" regimen to appeal to. This is what, for example, testing suites for tools provide — just as standardization has been managed, again, even since the very first years of interchangeable parts.

Received 23 July 2001
Revised 5 October 2001
Accepted 8 October 2001

References

[Birnbaum 1997] Birnbaum, David J. 1997. "In Defense of Invalid SGML". At `http://clover.slavic.pitt.edu/~djb/achallc97.html`

[Birnbaum 1998] Birnbaum, David J. 1998. "The Problem of Anomalous Data". Markup Technologies '98.

[Bloom 1982] Bloom, Harold. 1982. *The Breaking of the Vessels*. The Wellek Library lectures at the University of California, Davis. Frank Lentricchia, Series Ed. Chicago: University of Chicago Press.

[Caton 2000] Caton, Paul. 2000. "Markup's Current Imbalance". Extreme Markup Languages 2000.

[Cournane 1997] Cournane, Mavis. December 23, 1997. *The Application of SGML/TEI to the Processing of Complex Multilingual Historical*

Texts. Doctoral Dissertation, University College, Cork. Cork, Ireland.

[Cover 1998] Cover, Robin. 1998. "XML and Semantic Transparency". At http://www.xml.coverpages.org/xmlAndSemantics.html.

[Cover 2001] Cover, Robin. 2001. "Conceptual Modeling and Markup Languages". At http://xml.coverpages.org/conceptualModeling.html.

[Sperberg-McQueen 1994] Sperberg-McQueen, C.M., and Lou Burnard, eds. 1994. "A Gentle Introduction to SGML". In *Guidelines for Electronic Text Encoding and Interchange*. Repr. 1997. Chicago, Oxford: Text Encoding Initiative. pp. 13-36. Available online at http://www.uic.edu/orgs/tei/sgml/teip3sg/

[Hollander 1981] Hollander, John. 1981. *The Figure of Echo*. Berkeley, CA: University of California Press.

[Hounshell 1984] Hounshell, David. 1984, 1985. *From the American System to Mass Production, 1800-1932*. Baltimore: The Johns Hopkins University Press.

[Lancashire 1995] Lanchashire, Ian. 1995. "Early Books, RET Encoding Guidelines, and the Trouble with SGML". At http://www.ucalgary.ca/~scriptor/papers/lanc.html

[Lie 1999] Lie, Håkon W. 1999. "Formatting Objects considered harmful". At http://www.myopera.com/people/howcome/1999/foch.html.

[McCarty 1999] McCarty, Willard. August 29, 1999. An "Analytical Onomasticon to the *Metamorphoses* of Ovid". On-line sampler. At http://ilex.cc.kcl.ac.uk/wlm/onomasticon-sampler/.

[Quin 1996] Quin, Liam. November 1996. "Suggestive Markup: Explicit Relationships in Descriptive and Prescriptive DTDs". SGML'96. Graphic Communications Association.

[Reid 1998] Reid, Brian. 1998. Keynote address to Markup Technologies '98.

[Renear 2000] Renear, Allen. 2000. *The Descriptive/Procedural Distinction is Flawed*. Extreme Markup Languages 2000. Reprinted in*Markup Languages: Theory and Practice*, 2, no. 4 (Fall, 2000).

[Rockwell 2001] Rockwell, Geoffrey. February 20, 2001. Private e-mail to the author.

[Goldfarb 1990] Goldfarb, Charles F. 1990. *The SGML Handbook*. Oxford: Clarendon Press. Annex A. Adapted from Charles F. Goldfarb, "A Generalized Approach to Document Markup", in *SIGPLAN Notices*, June 1981.

[Sperberg-McQueen 2000] Sperberg-McQueen, C.M., Claus Huitfeldt, and Allen Renear. "Meaning and Interpretation of Markup". Extreme Markup Languages 2000.

[XML 2000] Bray, Tim, Jean Paoli, C.M. Sperberg-McQueen, and Eve Maler, eds. 6 October 2000. "Extensible Markup Language (XML) 1.0 (Second Edition)". W3C Recommendation . At http://www.w3.org/TR/2000/REC-xml-20001006