An SGML Implementation with 100% Tag Omission

(submitted to Extreme SGML 2006 but not accepted)

Robert Tischer
GrammarApps, Inc.
6905 Valley Brook Dr
Falls Church, VA, 22042
+1-703-340-4523

grammarapps@vacoxmail.com

ABSTRACT

The primary purpose of tags is to delimit content for computer processing purposes. Human readability of tags is inconsequential in comparison. The tag's predecessor, single character delimiter, originated in the mainframe era when computer memory was scarce, computing power was centralized, unique addressing across the network was limited and graphical user interface (GUI) technology was not yet invented. Are tags, then, necessary today? Certainly, the specifying of element boundaries in surface content is still cardinal, but with the technical capabilities of today, there is no reason their identification and tracking by computer programs has to be directed from the content data document itself. Eliminating them would allow DTD context as well as content to employ normal, unobtrusive textual and graphical structural cueing within documents. It would arguably allow computers to distribute context in a linguistically more defensible way than the current practice of sending tagged files from point A to point B.

General Terms

Design, Human Factors, Languages, Theory.

Keywords

Surface structure, presentation, deep structure, context, cues, graphical, markup, tags, context, parsing, SGML.

1. INTRODUCTION

This paper suggests that a real way to move forward with markup and to solve the problem of distributed context, is to reexamine the necessity of using explicit tags as the primary construct for associating deep grammar structure to surface presentation of content.

Markup tags in SGML has been the result of the text processing markup evolution that began with delimiters. Delimiters allowed computers to trigger functional code capability (e.g., formating of text, retrieval of data from database) at alterable points in text content. Single character delimiters, being devoid of natural language meaning, were later expanded in procedural markup to be multi-character mnemonics (e.g., .tb 4) which made their representation nominally meaningful to humans. By making the code triggers recognizable, they were more easily manipulated by the end user who possessed the data document. Then SGML broke with the limitations of procedural markup and tied content directly to a separate document type declaraction (DTD) context. The DTD used full identifiers (linguistics: signifiers) to assure that the underlying grammar structure for a particular data document could be tied to an explicit context structure. The DTD served as a proxy for the user's meaning structure and at the same time was machine processible (refer to [3] for a discussion of the nature of the proxy category).

But suggesting that tags be dropped from SGML, and even presenting arguments for doing so, would only be a thought experiment unless it could be put into practice. As evidence that the principles proposed in this paper are valid and viable, a **GramBox** application is presented and described in section 4. **GramBox** shows that the presentation layer of a worst case graphical application (i.e., does not contain text) can behave as if it were marked up text with traditional tags delimiting data element boundaries.

2. SGML Tag and Construct Problems

The marked up data document has a long history. A data document is any file of plain old data (POD) or text that conforms to some form of explanatory/prescriptive context. In the sections below, three aspects are explored: how the data element boundaries are identified, where the element boundary implementation resides, and what technology or constructs were and should be used to improve the first two points.

2.1 SGML Configurability

SGML was endowed with a number of mechanisms for altering its appearance and allowing it to adapt to current content in the field. There are constructs for defining alphabets, record boundary notation, and even a capability to change the SGML delimiters themselves (e.g., the angle brackets). The idea was that allowing the SGML implementor in the field to alter the syntax without affect the integrity of the resulting DTD was an advantage. Altering constructs changes the orthography of the data delimiting syntax.

But years of SGML practice have not proved a lot of these constructs to be useful. The question is, was the reason they weren't that useful because they were the wrong constructs? Or was it because legislating constructs even via an ISO organization does not help the issue of usability?

2.2 Pseudo SGML Problems

This example assumes that altering the SGML constructs will affect its viability and therefore implicitly overlooks alternative explanations.

It has been suggested that the problem with markup lies in the DTD's imprecise ability to express semantics [6]. That paper attempts to strengthen the meaning (i.e., semantics) of SGML DTDs by adding new constructs. But meaning is more complex than language expression alone and cannot be fixed by merely adding more expressive constructs. Meaning on the computer involves code behavior as well as naming (refer to section 3.1).

Since the DTD involves naming, then any resultant meaning must include the DTD's implementation.

Instead of adding more constructs to SGML, this paper argues instead that the DTD is richly sufficient to express any concept that language itself is capable of expressing, and that the problem lies in the fact that de facto SGML and DTD implementation is out of date.

2.3 Tags Originated in Mainframe Era

This 40+ year tradition of interspersing trigger points among text characters in a document is based on the need for computing programs to know precisely where to apply functionality. Start and stop tags have traditionally enclosed the text that is to be the target parameter of that function.

2.3.1 Electronic Data Interchange (EDI)

EDI, called ANSI X12 in America and EDIFACT in Europe, both use delimiters to markup data.

There was an economy to single character delimiters. Here is an example of an ANSI and an EDIFACT EDI document:

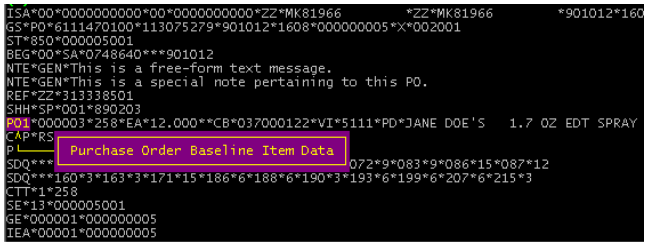

Figure 1: ANSI X12 EDI Purchase Order example

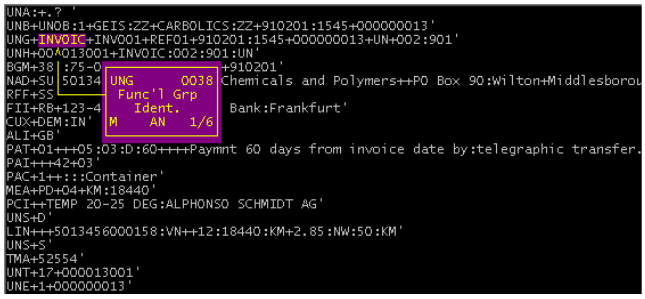

Figure 2: EDIFACT EDI Invoice example

The EDI documents in Figure 1 and Figure 2 use the same delimiter and segment mnemonic model. An ANSI document contains transaction sets like the 850 PO from which the document in Figure 1 was created. Mnemonic segment codes, like the PO1 highlighted in Figure 1, begin a data segment that is filled with delimited data elements. The 850 PO is equivalent to a DTD. By exercising various 850 PO options, many valid individual documents can be produced. Segments can loop. For example, a PO1 segment's limit is 100,000. Also, loops can repeat up to specific limits. Some data elements are optional and others are mandatory. Programs that read these documents need to understand the particular transaction set's grammar in order to make sense of the data. An EDI viewer is functionally the same as today's web browser.

There are three main differences in the SGML and the EDI model for producing grammars:

1) the number of grammars each can produce is different. For (ANSI) EDI there can only be less than 1000 and each transaction set has to be versioned every year or so.

2) their authoritativeness (refer to section 3.3.3) is different. For each transaction set there exists a body of professionals who have used years of their experience and expertise, plus Roberts Rules of Order, to vote on and agree upon the DTD transaction set's semantic grammar as well as the content definition of each EDI data element. Because of this, EDI transaction sets are authoritative. SGML DTDs are not in this sense.

3) their delimiter mechanisms are different. EDI delimiters are mostly nameless. The single character delimiters (e.g., + : * etc.) are anonymous, and only have meaning within their C6 (refer to 3.2) context. SGML tags are delimiters whose identifiers (linguistics: signifiers) relate them to their semantic grammar elements. EDI documents were never directly meant to be read by humans. Hence, the term *Electronic* Data Interchange.

SGML, by default, adopted the same A to B document transmission method as EDI. As a result, DTDs also have to be versioned and disseminated. Disseminating context by file transfer, however, is not equivalent to propagation of context (refer to section 3.3.4). SGML tag naming is many times more flexible than EDI deliminters offer. However, this flexibility has not been put to use. Compare, for example, the fact that there is only one HTML DTD that has spawned billions of web page instances. The semantic richness of the HTML DTD is mimimal and is basically the office memo with tags. Whereas, EDI has thousands of sets and messages. It was clearly not SGML's flexible and descriptive tag naming capability that made it a household word. It was also clearly not the viewer that made the different since an EDI viewer and the web browser function in almost the same way (i.e., they read and present data from a delimited data content file).

Eliminating tags entirely would mean that commonly accepted software applications such as word processors and spreadsheets could be grammarized and acceptance would be automatic. If this ultimate goal is a possiblity, then the problem of acceptance lies in the implementation. The We now turn to SGML's ancestry and examine how its implementation model has been inherited from the mainframe era.

2.3.2 Centralized Computing

The mainframe meant computing had to be centralized. Data creation and manipulation had to occur with the use of thin clients. It made sense to collect the data in a tersely delimited document and input it into the mainframe when it became your turn. But CPU ubiquity decentralized computing, so the data creator could create and maintain his data at their points of origen instead of collecting them into one content document. Since the meaning of C2 (refer to section 3.2) context is distributed as is the nature of language communities, then distribution of CPU computational power directly supports the linguistic concepts of distributed context leading to the eventual elimnation of the need for A to B DTD context and tagged content file transfer model (see section 3.3.4).

2.3.3 Limited Unique Addressing

But if tags are to be eliminated, then computing must keep tract of element boundaries across the network. To do this, there must be a near infinite capability of addressing the individual element

nodes. IPv6 will make this kind of addressing possible. Here is why this kind of addressing is necessary.

Tying element boundaries to executing code and to its grammar structure is implementation-wise quite simple, as long as the target content is text and as long as the DTD owner and content creator are the same person. In this limited case, a simple array of pointers is sufficient. But change the content to be a graphics application, which this paper explores, or distribute the context across more than one computing site, which is the implicit meaning of context and explicit purpose of the DTD, and more sophisticated programming techniques and functionality are needed to keep track of changing data element boundaries. This latter point is beyond the scope of this paper and will be addressed in future research. Suffice it to say, that unlimited unique addressing will be involved in order to uniquely keep track of virtual group members elements that they own and are recipients of instantiated elements at various sites.

2.3.4 The Sequential Parsing Dilemma

Part of the implementation inheritance problem is, a major part of SGML's unquestioned implementation inheritance is the sequential parser. The conventional parser is also a relic from the mainframe era. The sequential input character parser is incapable of addressing the superset of alphabetic characters, which is the realm of linguistic signs. For a description of the conventional compiler's input character problem refer to [10]. The purpose of parsing is to execute an action if the semantic case in a node tree warrants it. For example, a conventional parser emits intermediate code when a production rule (similar to a content model) is reduced (i.e., whose terms (SGML: elements)) are satisfied.

With the advent of GUI technology, it became possible to process more than sequential characters in a text or on a command line. By then, unfortunately, the construct of using tags and delimiters had already lodged itself in the emerging practices of document processing.

2.4 Tagged Configuration Files

The implementation of markup has been best exploited by the XML subset of SGML. But because XML also unwittingly adopted the conventional parser, it has fallen victim to the practice of sequentially parsing of tagged up documents in client server fashion. They are mainly used to configure applications or transmit data in a client server transaction from disparate applications with much redundancy.

2.5 The HTML Implementation Problem

The HTML DTD implementation (i.e., browsers) offers insite as to how a new SGML implementation has worked. Web pages provide meaning for client server users since for the first time it involves code that is not a part of the main implementation. The HTML DTD still uses the inherited data content file (i.e., .html or .htm file), but with a conceptually new addition.

HTML DTD implementations tie presentation to the tagged document, where an HTML DTD instance is one set of HTML web pages. Such an instance merely exercises different combinations of content model elements' connector and occurrence indicators of the HTML DTD. The resulting instance is called a DTD variant (computer science: parse tree derivation) and is merely a usage instance of the DTD. Some HTML presentation code has traditionally been tied to the elements themselves which the theory in this paper finds in no way objectionable. This is the case because deep and surface expression are linguistically inseparable and therefore presentation semantic action behavior is indispensible to the expression of context. For example, the HTML H1 tag generally has a different font size from H2 and so on, but this is true across all HTML documents. This means that H1, H2, etc. presentation code is not DTD instance specific, which is what one would expect. That code exists in the browser and it is the vendor who historically compiles and controls it.

Part of HTML's implementation is the embedded functional language called Java script. Java script is interesting in this paper's context because it ties content document delimiters (i..e., HTML tags) directly to the code that delimiters are supposed to trigger. The equivalent in EDI would be to put function code at the beginning of the data in Figure 1 and Figure 2.

The HTML tagged web page document uses java scripting to intersperse code and code calls directly within start and stop tags of a data document. The java script function behaves like a regular function call, but the call and the function have been placed within the tag layout of the document. This visually disrupts both the clear text flow of the tag reading and hinders ordinary function development and maintenance of the code. It is, however, a genuinely new form of implementation of the delimited data content file.

Java script is part of the semantic action of used HTML DTD elements. Semantic action is a computer science term that refers to the lowest level (i.e., indivisible) block of computer code that serves as the implementation of a non-terminals' production rule, where a non-terminal is a signifier like a DTD element, and a production rule is similar to SGML's content model. In short, a semantic action is the kernel construct of implementation of a hierarchical grammar. We combine java script, including its client server call capability, with the browser's implementation of the presentation layer.

There is no technical reason that it be interspersed among the HTML tags other than browser code is not compilable outside the vendor's perview. Both java script and browser code make up the semantic action of a particular HTML DTD instance.

Is the tag necessary? The sum HTML semantic action can be put anywhere in the process. This would imply that code could be employed to eliminate tags as well.

This discussion points out that tags are not technically necessary in even the most common of our DTD instance documents because all the presentation code could either be in the browser or interspersed among the content data. That is, the code's placement is arbitrary.

3. Theory

Without a unified theory of context, it is impossible to implement SGML and DTDs in such a manner that DTDs can serve as meaning proxies for humans.

3.1 Name + Behavior = Meaning

Regarding the phylogeny of meaning, linguistic meaning has a many million year development history which began when homo sapien started creating tools. Tool creation led to *tool thinking*. Tool thinking led to increased brain volume. Increased brain volume led to words and language capability which led to

language thinking and an even better brain. This evolution is testimony to the fact that meaning has a rich behavioral component and is inseparable from langauge expression [4][5]. The proxy representation of this associated behavior on the computer is code behavior. Since the de facto SGML code behavior model is out of date as this paper contends, then the DTD in practice appears to be incapable of expressing various aspects of language even though its meta-grammar constructs are more than adequate to nominally express linguistic phenomena.

3.2 Context Theory

Tags represent nominal context as described in the DTD. But context is not person specific. It is, rather, the expression of constant characteristics of groups of people. Human context is a labyrinth of tangible and intangible representations and conventions. Refer to [7] for research toward a unified theory of context. The highpoints of this research are the study of human language context, the cardinal aspects of which were modeled by addressing the human capacities and capabilities for context and linguistic expression from four different viewpoints:

1. the advent of human language capability and how it evolved in humans, how words, things and thing representations were historically created by individuals in a language group (**C**reate). This is the semantic phylogenic viewpoint and is the history of human tool making and word making behavior and its cumulative effect on us and our surroundings [5].
2. the advent of language competency in a child and how it adopts and absorbs its first natural language and acquires language competency in a short period of time. This is the semantic ontogenic viewpoint (**A**cquisition)
3. humans' competency for language understanding and expression (i.e., read, write, understand, talk) (**E**xpress)
4. human language's capability to reflect on language and how language seems to have no lower or upper semantic bound and therefore is always able to describe itself using itself (**R**eflexive)

These viewpoints constitute the targets of the academic disciplines of Language Psychology (or PsychoLinguistics) and Language Sociology (or SocioLinguistics). Hence, the **CREA** unified theory of context [7].

The following is a condensed taxonomy of connotations of context which in this paper will serve to make precise which kind of context is being referred to during the discussion. For a thorough discussion of context refer to [4]:

C1 Natural language grammatical context – The context that Chomsky built his phraseological observations on (noun phrase; verb phrase, etc.). It involves the standard parts of speech as every grade schooler has learned them.

C2 Natural language expression context – points to the fact that all linguistic expression (written, spoken, linguistic signs, etc.) occurs in its own a priori context individual and cultural. C2 is context which is encompassed by CREA.

C3 CFG contexts. C3 contexts are implicit in the strings of signifier tokens that belong to a language. Parsing these sentences of tokens reproduces the tree of tokens that represent this language. This type of context is not explicitly represented presentationally. EDI is an example.

C4 Programming language context – a programming language contains syntax and natural language-like key words (signifiers) and symbols (operators). These are used to detect inconsistencies and translate instances into machine instructions. C4 uses keywords (e.g., if then else) and operators (e.g., + = & *) to delimit its sentences.

C5 Software program declaration semantic context – is declared prior to software statements. The declared context of a particular program and its statements adher to a particular C4 so unambiguous instructions can be emitted. A C5 occurs within a C4 which when exercised yields a C3 instance.

C6 Semantic grammar – is an augmented CFG whose grammar and its elements are claimed by individuals to exist in the minds of several humans. A single human would mean the grammar is private which is a contradiction. C6 is also referred to as an ontology or a taxonomy of concepts. A C6 may have syntax such as occurrence indicators and connectors, in which case it is not an ontology or taxonomy. C6 is the proxy expression of C2 context.

Figure 3: Connotations of language context

SGML is used to create C6 semantic grammars called DTDs.

3.3 SGML Theory

The DTD is a proxy [1] object for the context that a text designer wants to be a presciptive form for future text input by himself or others. Transmitting a DTD from point A to point B does not share the context from one human to another unless an implementation exerts a proxy semantic/structural influence on the remote user and unless the user is motivated to learn and use it by the fact that it is authoritative.

If the purpose of SGML is to create a subtle connection to the real world of understanding [11] and still be computer processible, then by extension the DTD's purpose is to create a subtle connection to the real world of human context. The difficulty in current DTD writing practice arises when the DTD creator is not the same as the DTD user, which is the prevailing case. C6 semantic grammars are trying to emulate C2 language contexts.

The purpose of SGML is to coordinate more than one individual in the creation and production of linguistically expressive content, like, for example, documents.

3.3.1 Tag Obtrusiveness

Although great pains were taken in the design of SGML to maintain the connection to the real world of understanding, SGML implementation practices do not yet allow meta-grammar context building to become invisible, which is the proper role of any real linguistic context.

SGML contains meta-grammar constructs, like tag omission, whose motivation was to mitigate tags' intrusive effects. But it went too far. of explicitizing a target work's semantic grammar, in the form of a DTD. It is this paper's premise that it still is possible and the tagless GramBox SGML implementation presented herein is a step toward this goal.

3.3.2 Linguistic Reflexivity

Linguistic reflexivity is the characteristic of human language that makes activities like DTD writing possible [4]. Anyone at any time can address any aspect of language. What he cannot do is not use language to do just that. Hence, writing a DTD is a descriptive

expression about potential expression. What the current practice of DTD writing does not take into consideration is the authoritativeness (refer to section 3.3.3) of the DTD writer. Consequently, the de facto practice has been to first describe a context by writing a DTD followed by the prescriptive projection of that DTD onto users. Compared to how smoothly real language's context and content work together, this practice is nothing short of draconian.

3.3.3 Authoritative

The semantic meaning of a computer grammar should be human verifiable for truth testing at some level which is not the case for analytic grammars. Analytic grammar are only internally consistent. It is this verifiability that makes the grammar *authoritative* or not. C3, C4 and C5 contexts are analytical since their constructs only reference other constructs that are internal to the language (e.g., chess). EDI produces authoritative C6 semantic grammars because they are verified by a group of authorized professionals, and SGML produces C6 semantic grammars authorized by the individual document writer or by a group of document writers who produce a single DTD. The activity of producing C6 semantic grammars is usually referred to as *knowledge mapping* or *topic mapping*, but not quite because mapping alone contains no generative capability. The syntactic flexibility SGML offers with its simple connectors and occurrence indicators provides a subtle connection to the human expressive capacity for cloning and copying existing entities. A semantic grammar is authoritative to the degree it is mutually exclusive with respect to other semantic grammars. If the ISO SGML Working Group 8 had followed through on its intent on setting up legally binding public identifiers and/or registering of DTD instances around the world, or perhaps had adopted EDI-like authoritative negotiation procedures, it too would have been able to produce authoritative semantic grammars. As it is, syntax-less ontology efforts are taking place in order to fill in the semantic void. An example of a very authoritative and simple semantic grammar is the domain name system. In contrast to authoritativeness, a private citizen from Cincinnati who has written a DTD on the History of Man, is not authoritative because the guy on the next block could have his own History of Man DTD. The reason why being authoritative is important has to do with human intentions and learning.

The only reason the HTML DTD is authoritative is due to its place in technical history and subsequent usage volume. The HTML DTD is, as a result, fully distributed, whereas the private citizen's DTD is not and therefore cannot serve as the machine context for anything but his own and perhaps his friend's purposes.

SGML DTDs are in one sence, authoritative. They are the product of the ISO standards organization. This form of authority does not, however, extend to the content of the produced DTD which is the problem.

3.3.4 Context Cannot Be Transmitted From A to B

SGML Document Interchange Format (SDIF) was the first attempt to transfer SGML context and content from one place to another. Although SDIF has limited use today, there are many successors including the lowly email attachment that use the A to B transfer model. A curious A to B method is the well-formed tagged document which implicitly carries the DTD from A to B. Although novel, the A to B transaction model is the same. The problem with the A to B model is, it ignores the linguistic realities of distributed C2 context.

Context is a complicated entity as section 3.2 points out. A curious thing happens when transferring a C6 DTD context from A to B: it loses its semantic (i.e., meaning) reference to C2. Why this is so is a much larger theoretical issue than this paper can bear, but is none the less at the heart of the transfer problem. Again, [4] can offer some insight as to why this is the case.

3.3.5 Recipe for an Invisible DTD Context

For a revamped SGML to be viable, it must make itself invisible like real language context. This is not impossible as the GramBox demo below seeks to prove by example. The DTD needs to behave like a fabric of behind-the-scenes cognitive language tools owned by a virtual group of participants. An element in a DTD needs to be present at all participants' sites. That element's ownership needs to be unambiguous, so content for that element never conflicts with the other members of the group. Any changes to that element's model or content needs to be propagated to the others immediately. Even the element and content model themselves need to be cloaked in the normal use of presentation layer. For example, if an element's name is CardioVascular, the GUI presentation of that element could be:

where the shading, section indexing and even capital letters are linguistic signs that indicate unobtrusively that this is a contextual element (see covert and overt tagging cues sections in [8] and modality taxonomy in [2]).

The above practical description of a revamped DTD usage proposed is an even better connection to the real world of understanding, all the while allowing the DTD to exert its influence in the harnessing of computers to support cooperative work. Obviously, from this perspective, efforts to eliminate the DTD are counter productive with respect to tecnically being able to move forward toward a linguistically defensible machine representation of context.

3.4 Parse Theory

Computer science has spent 50 years developing the conventional compiler which is based on the sequential parsing of input characters. (For a description of conventional sequential parsing refer to [10]). This ubiquitous technology is the root of the SGML tag problem because the basic forms of parsers developed (see Aho concerning LL(k), LR(k), LALR(k), SLR(k), bottom-up, top-down, predictive, etc. parser types [1]) were all developed based on the input character. If they were to be developed today, they would be based on the more powerful GUI capability. GUI technology has the capability of processing linguistic sign proxies whereas alphabetic characters are only a subset of the presentation layer's set of linguistic signs. For example, in the SGML implementation presented below, the right-click on the mouse is a computer processible linguistic sign that indicates the x and y coordinates of the screen surface, which, in turn, informs the program exactly which instance – TopRectangle, NonOverlapSubRect or child or grandchild etc. that the user wishes to address. The computer science sequential parsing technology that helps conventional parsers determine unambiguously where in a parse tree semantic actions are to be carried out are FIRST and FOLLOW sets [1]. These constructs are no longer necessary in a non-sequential parsing environment

like the one the GramBox demo application was built. Replacing these disambiguating constructs is the DISCRIMINATOR which is a sign that unambiguously focuses attention on an instance in a proxy context which will be described below.

4. A Tagless SGML Implementation

Instead of using a start and stop tag to obtrusively point to text or graphics presentation that belongs to a particular DTD element boundaries, the computer itself could be used to maintain the relatinship between a used element in a DTD and its presentation, and that relationship would be an invariant across all node elements and their DTD grammar counterparts. This model is the epitome of a computerized realization of semantic deep structure being directly tied to its surface presentation content.

4.1 Design of the GramBox Application

A word document is actually filled with cues that orient the writer as to the structure. Therefore, to test the theory of GUI controlled content, a worst case application was devised, called GramBox.

GramBox is a graphical windows application that was built using an interactive SGML distributed compiler (IDC). GramBox consists of three elements: GramBox, the root node, TopRectangle, whose visual component is a blue dialog box, and NonOverlapSubRect, which may be created if the user wishes, anywhere on the TopRectangle dialog surface. Each rectangle may be any color (the grammar doesn't specify), may be moved (again, the grammar doesn't specify), and may be resized. The only thing the grammar does specify is that the rectangles do not overlap each other and that they are subordinate to TopRectangle or as the recursive content model indicates, to a parent NonOverlapSubRect. If they were recursively created, then they should still remain within their parent's rectangle boundaries.

4.2 Revamping SGML with Axioms

Since no design guidelines existed for such an application, design constructs were developed and subsequently served to guide the development of SGML IDC. Without such axioms, only current implementation practices would have been adopted which typically use client server and remote procedure call (RPC) technology, and input token sequentially parsing compilers. The SGML IDC was used to develop GramBox. GramBox's functionality will serve as the defense of these axioms' correctness in this paper:

1. The Computer can only simulate temporal and reflexive linguistic behavior and phenomena. (refers to relationship between computer as anthropomorphic surrogate thinking (e.g., AI) vs. computer as supporter of cooperative grammar-based work) (GramBox is the latter)
2. Semantic Grammar Verifiability Axiom. (refers to authoritativeness of semantic grammars as discussed above) (Verification of GramBox's correctness comes from the element name, the SGML IDC's node tree, and the rectangles in the TopRectangle)
3. The Sign Axiom. (refers to linguistic sign being the arbitrary designates or of the whole relation between the signifier and the signified. It is the sign that is the GUI proxy on computers)
4. The Signifier Axiom. (refers to proxies (e.g., the camel-cased SGML signifiers) for things (e.g., PhillipsHeadScrewdriver) and linguistic conventions (e.g. RedStopLight) (GramBox's semantic grammar (i.e., GramBox, TopRectangle, NonOverlapSubRect))
5. The CREA Language Axiom. (refers to a complete theory context based on language Creation, Reflexivity, Expression and Acquisition) (GramBox and the SGML IDC do not violate basic linguistic context theory as depicted by CREA)
6. Revised Abstract Algebra Model for Computer Science (refers to computer science's parsing dependency on input characters) (refer to [12] for revised computer science compiler grammar model) (GramBox application has DISCRIMINATOR code (i.e., right click sign))
7. The Reflexive Grammar to Sign Mapping Axiom (refers to the relationship between DTD signifiers and the presentation layer. GramBox is pure sign representation, so there is no distinction between content and context representation)
8. The Production Rule to Sign Mapping (Syntax Directed Output) Axiom (refers to projection of production rule instances to GUI presentation) (The GramBox user can verify that the element names and code behavior are one to one related).
9. The Interactive Navigator-Sign Verifiability Axiom (Authoritative agreement that 8's pieces constitute the correct picture meaning regardless of whether the document is textual or graphical).
10. The Sign to Reflexive Grammar Kernel (Syntax Directed Input) Axiom. (refers to the DISCRIMINATOR construct described above)

4.3 GramBox Implementation

Since parsing ambiguity is always the argument used to legislate against alternatives to conventional character parsing, the **GramBox** grammar was made recursive. Recursiveness, which uses the same element at different points in the parse tree, provides the worst case test of whether a tagless implementation is correct or not.

Being recursive, **NonOverlapSubRect**s can appear anywhere in **TopRectangle** and they may appear as children of **NonOverlapSubRect**s. If the SGML application can without ambiguity keep track of the relation between all of the **NonOverlapSubRect** instances and the grammar, then it will have proved that a tagless SGML implementation is possible.

The next series of figures shows how the GramBox grammar is interactively built.

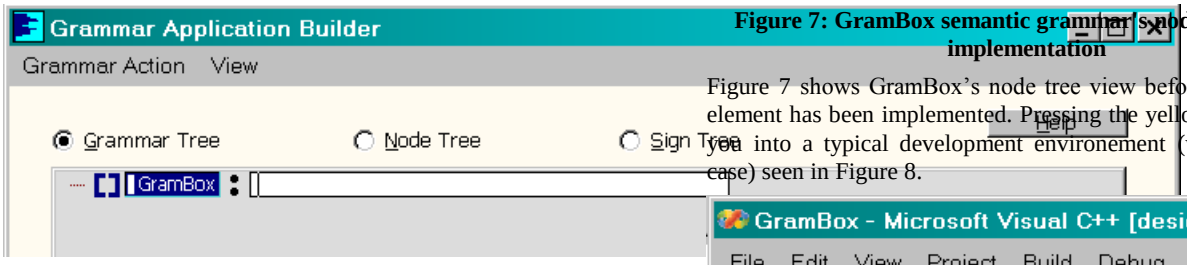

Figure 4: GramBox root node

Figure 4 shows the GramBox root node with a content model.

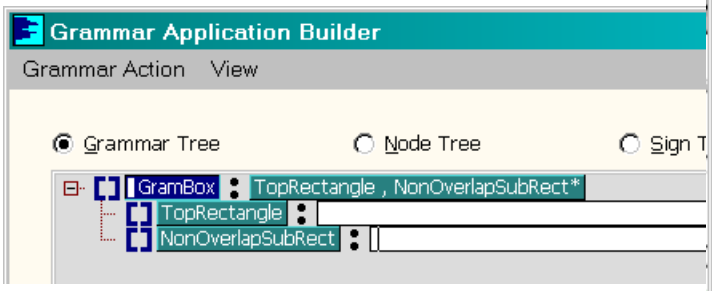

Figure 5: GramBox root node with content model

Figure 5 shows the GramBox grammar complete with content model containing the TopRectangle element sequentially followed by the NonOverlapSubRect element which may appear (i.e., be instantiated) zero or more times.

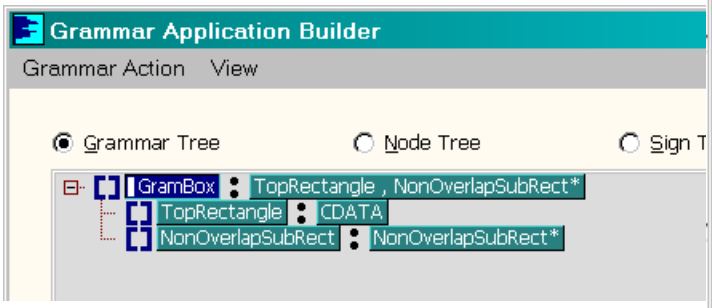

Figure 6: TopRectangle and NonOverlapSubRect elements

Figure 6 shows the TopRectangle and NonOverlapSubRect elements terminated where TopRectangle is terminated with the traditional CDATA and NonOverlapSubRect with itself, again with the occurrence indicator of zero or more.

Switching now to the Node tree, which shows that the semantic action (i.e., the code associated with the GramBox content model) is ready to be implemented. Refer to the yellow button in braces shown in Figure 7.

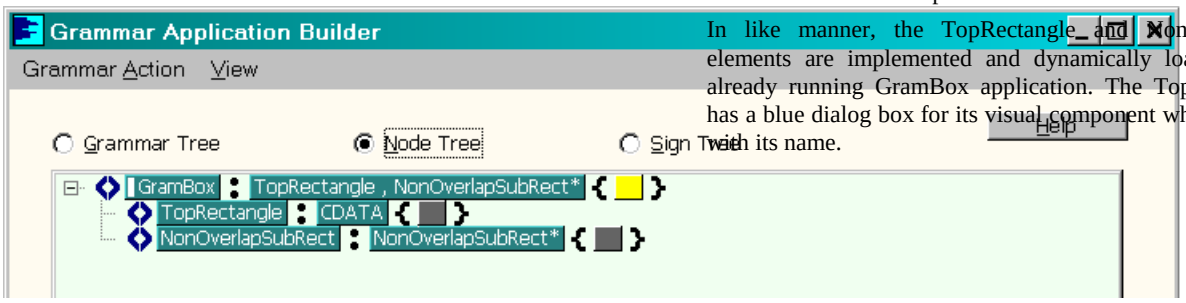

Figure 7: GramBox semantic grammar's node tree before implementation

Figure 7 shows GramBox's node tree view before the GramBox element has been implemented. Pressing the yellow button brings you into a typical development environement (windows in this case) seen in Figure 8.

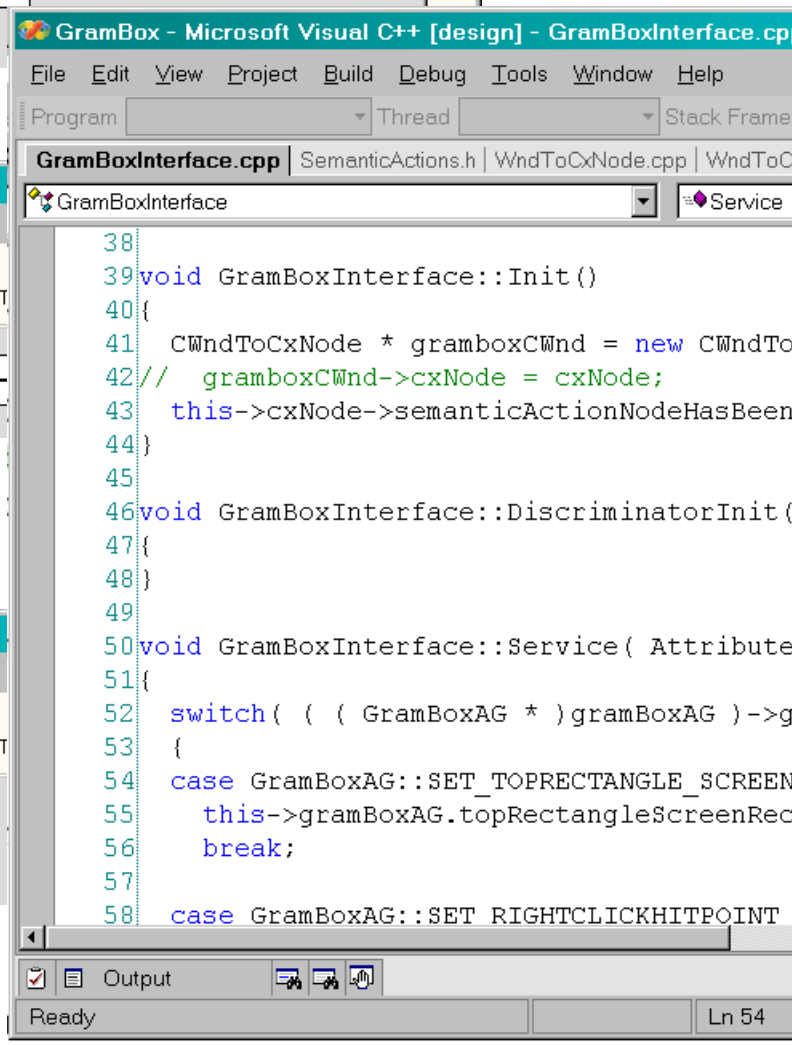

Figure 8: GramBox element development environment

Upon leaving the GramBox development environment, the resulting object code is dynamically loaded and begins running within the already running GramBox application. The GramBox node has no visual component.

In like manner, the TopRectangle and NonOverlapSubRect elements are implemented and dynamically loaded within the already running GramBox application. The TopRectangle node has a blue dialog box for its visual component which is consistent with its name.

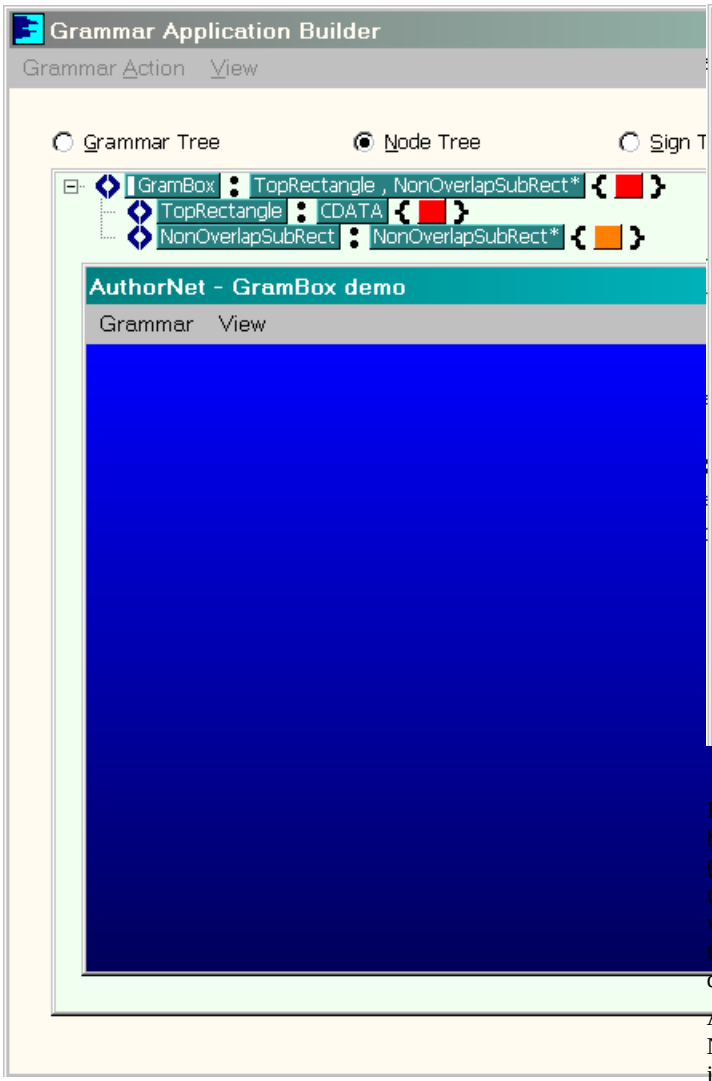

Figure 9: Fully implemented and running GramBox application

Figure 9 shows the fully implemented TopRectangle semantic action. The orange semantic action button for the NonOverlapSubRect node indicates that although the element exists grammatically and has been implemented, it has not yet been instantiated.

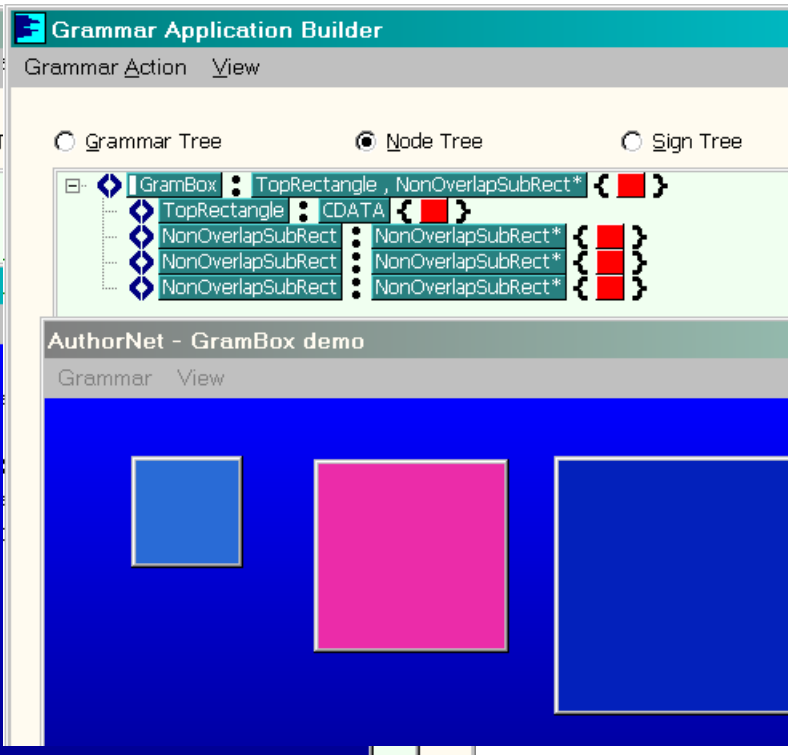

Figure 10: Three NonOverlapSubRect instances

Figure 10 shows that three instances of NonOverlapSubRect have been added the TopRectangle instance which was created (sequentially) ahead of any NonOverlapSubRect instances. Note that all three instances are represented in the node tree which verifies that the GUI presentation instances are rigidly attached to their underlying grammar nodes in the running parse tree (i.e., the deep structure).

A key piece of this technology is the method by which these NonOverlapSubRect instances were created. The TopRectangle instance in the form of a blue dialog box, is the DISCRIMINATOR node for any of its content model's NonOverlapSubRect instances. The right click was chosen to be the discriminating sign that indicates that a NonOverlapSubRect instance should be created just there at that point. Using computer science terms, the right click is a member of the FIRST set of that serves as a predictor of instantiation of a new NonOverlapSubRect exactly at that point in the GramBox node parse tree.

Figure 11 shows that no ambiguity occurs when instantiating four more child and grandchild instances of NonOverlapSubRect. One can see from the node tree's indentations exactly which NonOverlapSubRect is the parent and which is the recursive child.

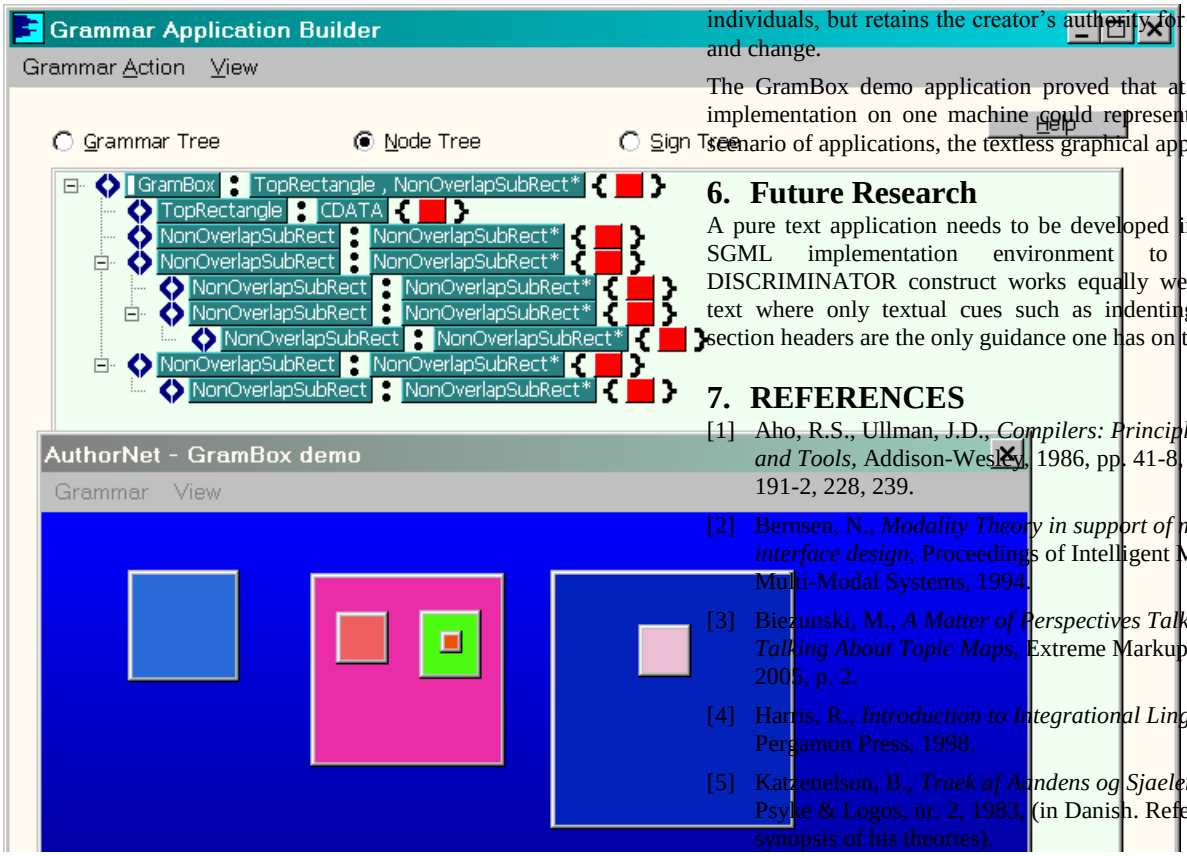

Figure 11: Four recursive NonOverlapSubRect instances

5. Conclusions

It was shown how SGML inherited by default, the same implementation from the mainframe era and how the deliminter of EDI documents was equivalent to tag. Allowing start and stop endpoints of DTD instance elements to be managed and code executed decentralized was not possible in the mainframe era, but is today. This allows for real language context to be implemented which would obviate the need for today's pseudo A to B file context transfer model.

In order to better understand the DTD, a unified theory of context was outlined to be used as a template for understanding a better role for the DTD as context proxy.

A set of computer science/linguistic axioms was presented that guided the development of the SGML IDC which, in turn, served as the framework for the GramBox application development.

Tags' human readability that connects them to their semantic grammar are no longer necessary. SGML IDCs can maintain the one to one deep to surface relationship without them as was seen by the GramBox demo application.

Despite the fact that SGML has many constructs that allow it to fit in with data in the field, without examining and coordinating its implementation, the DTD will not be able to transcend current limited sequential parsing limitations.

SGML implementation needs to develop an implementation practice that not only transfers elements to other sites and individuals, but retains the creator's authority for content creation and change.

The GramBox demo application proved that at least one DTD implementation on one machine could represent the worst case scenario of applications, the textless graphical application.

6. Future Research

A pure text application needs to be developed in the interactive SGML implementation environment to see if the DISCRIMINATOR construct works equally well with ordinary text where only textual cues such as indenting, indexing and section headers are the only guidance one has on the printed page.

7. REFERENCES

[1] Aho, R.S., Ullman, J.D., *Compilers: Principles, Techniques and Tools*, Addison-Wesley, 1986, pp. 41-8, 160-2, 188-90, 191-2, 228, 239.

[2] Bernsen, N., *Modality Theory in support of multimodal interface design*, Proceedings of Intelligent Multi-Media Multi-Modal Systems, 1994.

[3] Biezunski, M., *A Matter of Perspectives Talking About Talking About Topic Maps*, Extreme Markup Languages 2005, p. 2.

[4] Harris, R., *Introduction to Integrational Linguistics*, Pergamon Press, 1998.

[5] Katzenelson, B., *Traek af Aandens og Sjaelens Historie*, Psyke & Logos, nr. 2, 1983. (in Danish. Refer to [7] for synopsis of his theories).

[6] Renear, A., Dubin, D., Sperberg-McQueen, C. M., Huitfeldt, C., *Towards a Semantics for XML Markup*, Proc. on Document Engineering, 2002, pp. 119-126.

[7] Tischer, R. G., *The Anatomy of Context*. Master's Thesis, University of Copenhagen, Denmark, 1985 (in Danish with English synopsis available at http://70.169.161.243).

[8] Tischer, R. G., *Characteristics of SGML Text Entry Systems*, Contribution to Work Item 97.18.15.5 of ISO SGML W8, 1987, (available at http://70.169.161.243), pp. 2, 11-12.

[9] Tischer, R. G., White E., *Distributed Context as Long-running Parse Tree*. In press. (available at http://70.169.161.243)

[10] Tischer, R. G. *Dynamic Syntax Compiling*. Master's thesis in Computer Science, George Mason University, 2001 (available at http://70.169.161.243), pp. 4-12, 66-72.

[11] Tischer, R. G., *Theoretical Basis for the SGML Content Model*, in *The SGML Handbook* by Charles Goldfarb, Annex H, Oxford Univ. Press, 1990, p. 556, 558.

[12] Tischer, R. G., White, E., *Distributed Context as Long-running Parse Tree*, In Press (available at http://70.169.161.243).