Distributed Context as Long-running Parse Tree

Robert Tischer
Hiveware Inc
Falls Church, VA, USA

Elizabeth White
Department of Computer Science
George Mason University
4400 University Drive, Fairfax, VA, 22030 USA

ABSTRACT
In this paper, we describe a new implementation approach to replicating context that yields a jointly usable context by members of a distributed group. Without a jointly usable context, distributed software contributors are relegated to post facto asynchronous merging of conflicting content contributions. Since linguistically, all content occurs within some sort of context, describing this context, making it computer tractable, and sharing its likeness among a group of contributors becomes a viable approach to achieving a distributed context. To achieve this, established CSCW notions of using semantic grammars and concept-centric analysis as context representations are extended by showing how the conventional compiler's parse tree architecture can been drafted to represent semantic context. We show how a parse tree can be replicated to other sites using a recursive descent technique in order to establish joint context.

Categories and Subject Descriptors
H.5.3 [INFORMATION INTERFACES AND PRESENTATION]: Group and Organization Interfaces – *Computer-supported cooperative work*.

General Terms
Documentation, Human Factors, Theory.

Keywords
Collaboration architecture, Context representation, Long running parse tree, Replicated context, Semantic grammar, Concept-based, CFG, XML, SOA.

1. INTRODUCTION

Context representation and concurrency issues are closely related concerns. For example, the client-server methodology has typically been employed to force content contributors to take turns when creating input to the same content area. Inhibiting clients from making changes concurrently insures server data integrity. More typical for CSCW peer-to-peer (P2P) collaboration applications, however, is to allow concurrent changes to occur, but subsequently repair them asynchronously by merging similar data or mapping dissimilar data to each other [5]. Context in this case becomes a descriptive discovery process instead of a prescriptive guidance process (Refer to [7] for a formal discussion of linguistic context).

In order for two or more contributors in a collaborative system to function concurrently, and in order for them to make content contributions to the same target work without resorting to after-the-fact content reconciliation, they need to operate under the same context and they need to have agreed, prior to articulating the content, which part of the context belongs to each contributor. Collaborative systems that support this kind of cooperative behavior are referred to as *cooperative knowledge node systems* (CKNS).

Current CSCW context representations and assumptions are examined below and the problem of inadequate and even redundant context representation is delineated. The linguistic concept of context is the proper source from which to inherit a computer tractable methodology for representing context. As a first step in proving the usefulness of our approach, a demo is described where another site obtains an exact copy of a long-running parse tree.

2. Relevance to CSCW
Many collaboration technologies address issues without explicitly defining jointly usable context. For example, many collaboration technologies adopt context representation mechanisms such as files and folders for content sharing even in cases where this is inadequate (e.g., copying word processing files that have been altered among group members). Another example is the assumption that collaboration technologies which facilitate the juxtaposition of humans (e.g., teleconferencing) solve automatically the jointly useable context problem. While these technologies meld two or more humans' language contexts electronically by connecting them visually and verbally, juxtaposition merely relegates the context problem to ordinary group dynamic mechanisms and really does not employ the support of computing power for joint context resolution. A third example is the utilization and grouping of distributed computing potential (grid computing). In this case, context representation is possible because the attribute sought after is a trivial, uniform resource, like CPU cycles or local storage. In this example, the context representation question is merely implicitly minimalized (Refer to [2] for a survey of collaboration technologies).

Common for these examples of collaboration technologies is the insufficient application of context theory during design. From this perspective, if the CSCW community had a unified context representation theory and practice, cases where collaboration design defaulted to inadequate context representation might decrease. This paper attempts to describe and make practical an initial piece of that task.

Instead of the more familiar collaborative task of coordinating people with similar traits, CKNS coordinates human decision making, both context as well as content, in order to build a single work product. Coordination of human decision making toward a common goal is, in fact, quite common, as evidenced by any business or corporation. What this paper's implementation is attempting to do is to emulate such common divisions of labor while remaining computer tractable.

3. Cascading Parsing and Context Problem

A common method for representing context is to describe it using the XML document type declaration (DTD) resulting in the ubiquitous tagged XML file. Unfortunately, XML's de facto implementation method has been the use of the sequential parser. Consequently, each contributor site may have an XML file which contains content under its own DTD context. When an XML file is sent to a second site, which has its own XML content and DTD, the first site's XML data has to be parsed and reconciled with the second site's own data. This effect is cumulative and is repeated each time a data change is made.

Figure 1 describes the snowballing reconciliation effect, ***R***, of sequentially parsing XML files from one DTD site to the next for the g^{th} site. Each well-formed XML data file contains its content's entire context as stated by its DTD and therefore is a function of it. This is represented by the term in parentheses. Previous site's XML parsing and reconciliation effort is represented by $\boldsymbol{R_{n-1}}$ where its coefficient $\boldsymbol{k_n}$ determines how much of the effect is due to cascading XML files or is merely due to a linear summation of disparate reconciliation efforts.

$$R = \sum_{n=1}^{g} XML_{Data_n}\left(XML_{DTD_n}\right) + k_n R_{n-1}$$

Figure 1. Cascading sequential XML parsing effects

Even though XML is touted as being self-describing, by carrying around its own description in each XML file, it implicitly imposes an enormous redundancy burden without contributing to the resolution of a jointly usable context at multiple sites. Even in the simple case where there is no cascading reconciliation (i.e., a site only parses and reconciles a certain number of received XML files), ***R***'s parsing and reconciliation task is still linearly proportional to the number of XML files that contain changes (i.e., $\Theta(n) : k = 0$ in contrast to $\Theta(n^{1+\varepsilon}) : k > 0$). Because the file is parsed and the parse tree is thrown away, the next file has to be re-parsed from the beginning.

This snowball effect can be found in any middleware component architecture in much the same manner, one example of which is service oriented architecture (SOA). Each SOA provider represents a context. A particular user of multiple SOA functions has to reconcile each of these contexts in order to combine the result sets properly. The ***k*** term would be zero for an SOA unless the SOA is comprised of other SOA functions, in which case ***k*** would be some non-zero value for each member SOA.

4. Related Work

Relevant to CSCW are the notions of growing [4] or chunking [9] semantic grammars. The former paper presents an approach whereby difficult to obtain domain descriptions (semantic grammars) can, with their tool's help, be distributively harvested from non-experts. The latter paper offers a non-concept based method by which replicas can remain consistent. The implementation in this paper builds on these concept-based, semantic grammar notions.

In [5] the practical business context representation task of reconciling heterogeneous representations of product data catalogues is addressed. Their study approaches the topic area assuming that all data already exists and that the heterogeneous context problem is therefore a secondary data reconciliation problem. In contrast, this paper's implementation assumes that context representation creation occurs before content creation as is the case for linguistic context.

For an alternative method of populating new subscribers (latecomers) and thereby establishing context replication, see [3] which describes a method using the replaying of log messages. The replaying of log messages approach does not have the independent node (see section 6.2) characteristic.

5. Context Theory and Practice

There is much imprecision regarding the definition of context for use in computers and chronic confusion concerning context in the field of linguistics has not helped matters.

5.1 Linguistic Context

Because of the interdisciplinary nature of CSCW, it has been appropriate to include prior linguistic theory research concerning the acquiring and use of context in humans. In [7] a theoretical explanation was developed for how humans create, acquire and use context to communicate. This work provides a comprehensive framework from which practical context representation implementations like the one presented in this paper, can be spawned.

Linguistically, there is no such thing as a decontextualized communication process. This means that communication will always be contextualized [6]. But if this is true, what did Noam Chomsky mean by his famous example where he tried to ignore the existence of context in, "Colorless green ideas sleep furiously"? If this is a linguistic impossibility, then even his content expression must be couched in some sort of context. It is not difficult to find at least one plausible semantic context as Figure 2 shows.

NonsensicalYetGrammaticalSentence : Adjective* , Subject, Verb, Adverb?

Adjective : AdjacentWordNotMeaningfulWRTThisWord
Subject : AdjacentWordNotMeaningfulWRTThisWord
Verb : AdjacentWordNotMeaningfulWRTThisWord
Adverb : AdjacentWordNotMeaningfulWRTThisWord

Figure 2. A semantic grammar that binds Chomsky's "context free" sentence to meaning

The words in his sentence contribute uniformly to the **NonsensicalYetGrammaticalSentence** non-terminal, which is the meta-category of the meaning of his sentence. The **AdjacentWordNotMeaningfulWRTThisWord** non-terminal expresses that each content word in his sentence has a common meta-meaning with respect to its neighbors. Since we humans are trained from childhood to mentally process the sentence according to the natural grammar that is approximately expressed by the top production rule in Figure 2, we cannot immediately generate meaning because the sentence is so deliberately obtuse. But given the semantic grammar in Figure 2, it makes sense anyway. Refer to Harris' seminal work in [6] for a full explanation of why context cannot be discarded.

The field of computer science has made good use of the notion of Context Free Grammars (CFG), in that all computer programming languages today produce unambiguous sentences that can be compiled to emit intermediate code regardless of any subjective meaning the people involved may want to impose. The point of the above argumentation is, therefore, when dealing with harnessing computers manned by humans in a distributed environment, the computers' software must be dealing with some sort of emulation of subjective, yet integrated language context representation (i.e., meaning), or the resulting work content will be inconsistent.

5.2 Semantic Grouping

Already introduced into the CSCW community are the context representational notions of growing [4] or chunking [9] semantic grammars. A semantic grammar is a grammar whose non-terminals correspond to semantic concepts (e.g., **CardioVascular** in Figure 3). Referring to Figure 3, the item to the left of the : is a non-terminal, or element if viewed as an XML document type declaration (DTD). Various types of connectors (e.g., | & ,) and occurrence indicators (e.g., ? * +) can be used in this semantic grammar which add declarative strength to the syntax.

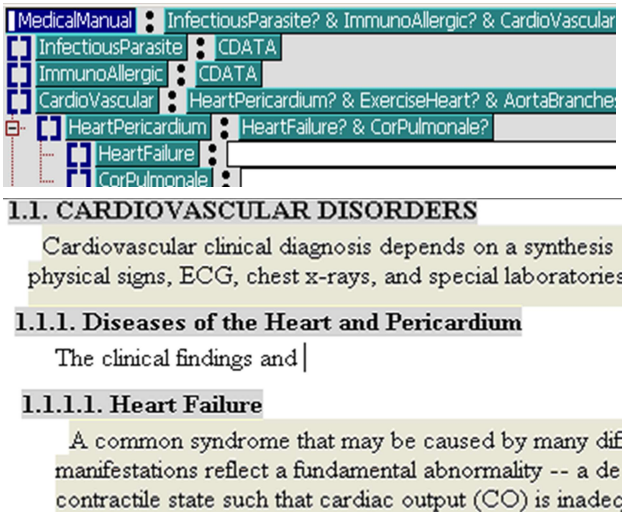

Figure 3. Medical manual grammar, rules and content

In words, a **MedicalManual** consists of an optional section that addresses the **InfectiousParasite**'s topic, an optional section that addresses the **ImmunoAllergic** topic, and so forth. The edit boxes to the right of the colon on some non-terminal nodes are nodes whose production rules have not yet been declared. CDATA terminates a production rule.

5.3 Extending Compiler Theory

Conventional compiler theory [1] can be modified and extended in order to enable non-sequential parsing which is necessary for parse trees to become long-running. The first goal of conventional compilers is to derive structure from sequentially arriving input symbols and to determine if this structure is consistent with the syntax and semantics of the programming language being used. The second goal is to execute semantic action code when a production rule is satisfied (i.e., is reduced). Semantic action code may or may not be the emitting of machine code [1][8]. Conventional compiling discards the parse tree after use because its task of emitting object code is complete. In like manner, parsed XML files build parse trees using the DOM (whole tree is built) or SAX (event callback but still sequentially processing of the tagged document) architecture. Our implementation expands the semantic action code to any executable code that is consistent with the everyday meaning of the node and its production rule. Because our implementation involves the execution of long-running, semantic action code, change may occur on any node in any order [8].

Instead of assuming that input tokens are characters in an alphabet, let them be any linguistic sign (i.e., a superset of alphabetic characters) that can be represented by graphical user interface (GUI) techniques. E.g., a mouse click could be an input sign.

A *grammar* is a 4-tuple, $\boldsymbol{G = (N, \Sigma, P, D)}$ (modified from [1]):

- $\boldsymbol{N}$ is a finite set of *non-terminal symbols*.
- $\boldsymbol{\Sigma}$ is a finite set of *terminal symbols*, disjoint from **N.**
- $\boldsymbol{P}$ is a finite subset of $\mathbf{(N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*}$ (i.e., the cross product of all qualified non-terminals and all non-terminals and terminals, the qualified non-terminals being prefixed and suffixed by any number and combination of non-terminals and terminals)
- $\boldsymbol{G}$ is related to GUI surface structure via GUI signs as well as strings or characters (traditionally, this has been limited to characters in strings of which strings symbols are comprised)
- $\boldsymbol{D}$ is a finite subset of GUI signs representing input content and signs, that unambiguously indicate which semantic rule from $\boldsymbol{P}$ is the desired rule whose semantic action is to be executed and whose annotation (decoration) updated.
- $\boldsymbol{N}$, the set of *non-terminal symbols*, are sometimes called the variables of a language or its syntactic categories.

6. Implementation Design

Guided by linguistic context theory and using the above modified compiler constructs, the new implementation approach to representing context in a CKNS is described.

6.1 Long-running Parse Tree

In our implementation, code behavior at each node in the parse tree does not emit object code, but rather determines how the contributor can express his content for a particular node. For example, if the semantic actions (i.e., code associated with a particular node) were like a word processor and its production rules were like those in the grammar part of Figure 3, the output might look like the lower part of Figure 3 where 1.1. CARDIOVASCULAR DISORDERS represents the **CardioVascular** non-terminal, 1.1.1. Diseases of the Heart and Pericardium represents the **HeartPericardium** non-terminal, the 1.1.1.1. Heart Failure represents the **HeartFailure** non-terminal, Cardiovascular clinical diagnosis depends on... text represents the **CardioVascular** node's content and A common syndrome that may be caused by many... text represents the **HeartPericardium** node's current content, and so forth.

6.2 Subscriber Population

The essence of this paper is to illustrate how a long-running parse tree works, how it can be copied, and indicate how this approach opens the door for a full-fledged CKNS implementation. Each long-running parse tree node is running code whose behavior (semantic action), is consistent with the node name it represents.

Replicating a long-running parse tree to a new site has been put into practice by pushing such a tree to a new subscriber recursively node by node. The new site is called a subscriber because after population has finished, the new site's copied nodes will eventually be receiving continual updates from the host site's individual nodes in order to retain its replicate status over time.

Copying the running node tree from site **A** to site **B** is a recursive descent process beginning with the root node and is a recapitulation of the node and production rule creation process that site **A** went through to build its node tree. Site **B** begins by listening on a particular port number and receives the root node name, its production rule and its semantic action. It parses the rule and dynamically loads the received object code. If site **B**'s platform is the same as site **A**'s, then it is sufficient to transfer object code to site **B**. Otherwise, conventional source code would be involved in the transfer.

After **B**'s parsing of the root production rule is complete and its code is running, each of the non-terminals in the production rule would now be ready to receive their respective production rules and semantic action code, and so forth until the leaves of the tree at **A** have been reached and pushed to **B**.

Accompanying this paper is a video that shows a site **A** containing a demo tree containing three non-terminals: **GramBox**, **TopRectangle**, and **NonOverlapSubRect**. Each non-terminal's semantic action is running code. **TopRectangle** displays a blue dialog box and **NonOverlapSubRect** instances would be represented by any number of differently sized rectangles that do not overlap each other. **NonOverlapSubRect** instances are not addressed in the video.

This demonstrates how any number of sites can be complete replicates of **A** while retaining their independent node behavior. Independent node behavior refers to a characteristic of our implementation whose usefulness will become evident when, in future work, node ownership is made transferable.

7. Conclusions

Context representation can be under designed as was suggested

above for some collaboration technologies, and it can be over designed as was suggested in the case of XML files, and its existence can even be denied as was the case for the CFG.

This paper presents an alternative distributed context implementation to after-the-fact data reconciliation techniques in use in some collaboration systems today. Motivated by linguistic context theory, this approach makes the assumption that any heterogeneous community's content and context can be built in conjunction with each other from zero, provided there exists a jointly usable context beforehand. It was demonstrated how an evolving semantic grammar could be implemented using a modified long-running parse tree, the first step of which was to node-wise populate a new subscriber.

8. Future Work

The next step is to implement the replicated trees such that a change in a node's name, production rule or semantic action is immediately pushed to all their cloned counterparts in the replicates. This allows the copied trees to continue to be replicates. Change to any node would automatically be able to occur in parallel.

By definition, a long-running subscriber parse tree as context does not contain content or context contributors. But that does not have to be the case because this paper's implementation has the characteristic of node independence within the grammar. Therefore, in order to fully realize the CKNS architecture, node ownership needs to be made deployable and thereby transferable.

9. REFERENCES

[1] Aho, A., Ullman, J., *The Theory of Parsing, Translation, and Compiling,* Prentice-Hall, Inc., 1972, p.85,91.

[2] Androutsellis-Theotokis, S., Spinellis, D. *A Survey of Peer-to-Peer Content Distribution Technologies*, ACM Computing Surveys, Vol. 35, No. 4, Dec. 2004, pp. 335-371.

[3] Chung, G., *Towards Dynamic Collaboration Architectures,* Proc. 2004 ACM conf. on CSCW, Chicago, IL, pp. 1-10.

[4] Gevalda, M., Waibel, A., *Growing Semantic Grammars*, Carnegie Mellon Univ., Proc. of the 36th annual meeting on Assoc. for Comp. Linguistics, Vol 1, 2005, pp. 451-456.

[5] Guo, J., Sun C., *Context Representation, Transformation and Comparison for Ad Hoc Product Data Exchange*. ACM SIGEcom Exchanges, Vol. 4, No. 1, May 2003, pp. 20-28.

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

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

[8] Tischer, R. G. *Dynamic Syntax Compiling*. Master's thesis in CS, George Mason University, 2001 (available at http://cs.gmu.edu/~white/xxx).

[9] Veiga, L., Ferreira, P. *Semantic-Chunks A Middleware for Ubiquitous Cooperative Work*, , Proc. 4th Workshop on Reflective and adaptive middleware systems, ARM 2005.