General Architecture of HIVE/WARE

1. Layman's Description

One of the largest paradigm shifts in writing took place when we moved from the use of the typewriter to the use of the computer. Just as the typewriter "automated the pencil", the word processor "automated the page". These *pencil* and *page* metaphors still hold sway over the design of the word processor today which blocks out questions of what else the computer could do for the writing process. Like the pencil and page, the word processor allows a document to be worked on by only one author at a time. Secondly, the word processor only allows one author to change the structure of a document at a time. It does this by allowing any character in the document to be changed at any time, which inadvertently alters the structure of the document. Altering structure eliminates the possibility of simultaneous sharing a single document. In other words, word processors today are still designed to be used by the solo author. This is in spite of the fact that written materials require constant revision and are invariably the result of many creative minds. HIVEware challenges these two assumptions.

Instead of the character, the page, or even the document, HIVEware offers group control over organized sets of topics. A topic is represented by any kind of understandable identifier, for example **EasyHomeRepair** or **CardioVascular**. Sets of topics arranged into some order (also known as a taxonomy or more recently called an ontology), plus the rules for making new related topics (also known as a syntax) are referred to as *semantic grammars*. Every document, organization, scientific group, or production entity has an inherent semantic grammar that is shared in varying degrees by that group's participants. Even knitting instructions are based on a grammar. HIVEware permits each group's evolving semantic structure to be a part of the word processing activity. This allows authors, organized by virtue of their joint semantic infrastructure, to simultaneously work toward the production of a single document. This is different from the way today's word processors operator which are designed to handle only strings of characters (that is, any character can be changed at any time by anybody manning the keyboard). With HIVEware, a shared and evolving semantic grammar is the controlling fabric of the collaborative document, which allows many authors to work simultaneously.

Readers familiar with the workings of compilers will recognize the phrase "Syntax Directed Compiling". Today's software compiling begins with character tokens

which are sequentially parsed. Instead of starting with these, the **HIVEbuilder** begins with the root element of a semantic grammar. As new subelements are created by context authors, each is farmed out to different and appropriate contributors. Then, and only then, does traditional compiling take place. Figure 1 shows the semantic grammar of the GramBox demo **HIVEware** application:

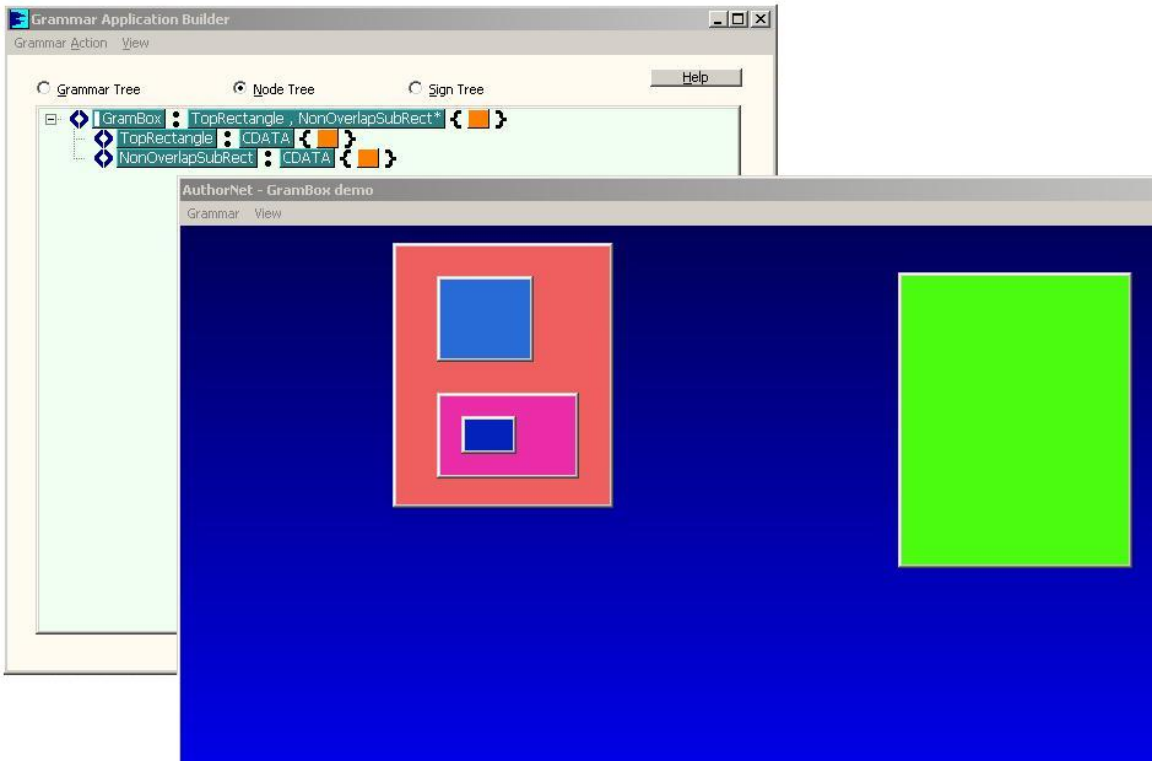


Figure 1: GramBox demo HIVE/WARE app

The art of compiling source code into executable code has not changed in 50+ years, but it should. Even though compiler theory and operation are well-researched topics, it is the implicit goal of HIVE/WARE that this process be modernized to include two obvious omission in the original mainframe era theory: the sign, or presentation layer, and meaning, also called semantics. Doing this would allow the compiling process to better address two major areas of concern in computer science today: distributed software development and execution, and native implementation of document markup technologies. The new type of compiler would be called a Semantic Directed Compiler of which the HIVEbuilder is an example. See the "Dynamic Syntax Compiling.doc" masters thesis for a description of this kind of compiling.

2. HIVEware Architecture

2.1 Context Authoring

HIVE/WARE is a socio-linguistic-technical approach that offers a comprehensive solution to the distributed CAD problem which addresses up front the distributed CAD system difficulties by providing scalable text and graphics exchange, alleviating the need for notification and merging, while enforcing ownership and accountability.

The objective of this proposal is to demonstrate an advanced technology that can utilize the collective knowledge of a project's distributed designers whether this knowledge is visual like a drawing or textual like a set of requirements. The innovative technology that seamlessly ties the textual and graphical dimensions together is **HIVE/WARE**'s synchronous distributed context, distributed content architecture (SDCDC™)..

HIVE/WARE applications are based on a constantly evolving knowledge structure, or grammar, that is altered by each and any of the authorized participants. Participants of a particular grammar-application are *context authors*, *content authors* or *subscribers*. The constellation of context and content authors reflects the organization chart of the people and devices in any working structure. The **HIVE/WARE** architecture is peer-to-peer replication and is fundamentally different from the current client/server architecture. Changes - context or content - are pushed to each other and the subscribers. If there are no information changes, the network traffic is zero although local processing proceeds unabated.

Figure 2 through Figure 5 indicate how an **HIVE/WARE** application works over time. Nodes are topics with an implementation to support it. There are four types of user entities: *root context*, *context*, *content authors* and *subscribers*, and only context authors can subdivide. Although content authors may be devices, it is only humans that can be context authors since it is only humans who can create language.

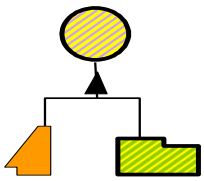


Figure 2: An HIVE/WARE topic node creates two sub-topic nodes

In Figure 2 context author,  , creates a sub-content author,  , and a sub-context author,  .

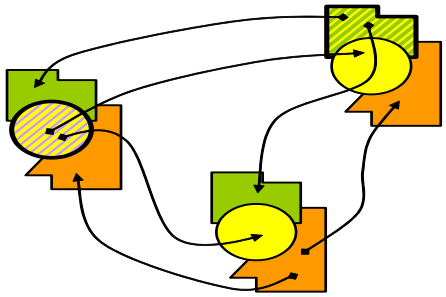


Figure 3: Nodes pushing one content change to their consumers

In Figure 3, the  context author sends new content to its two  nodes, which by definition are subscribers of  . Simultaneously, the same kind of event occurs for the  context author. Also at the same time, the  content author sends new content to its slave recipients. Each of the three replicates,  , remains synchronously up to date.

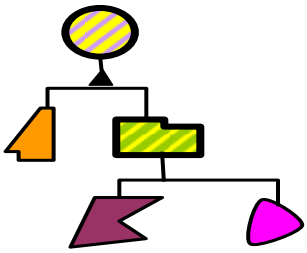


Figure 4: A sub-topic node creates new sub-sub-topic nodes

In Figure 4, the  context author creates the  and  content authors who cannot further subdivide. This occurs independently of all other activities.

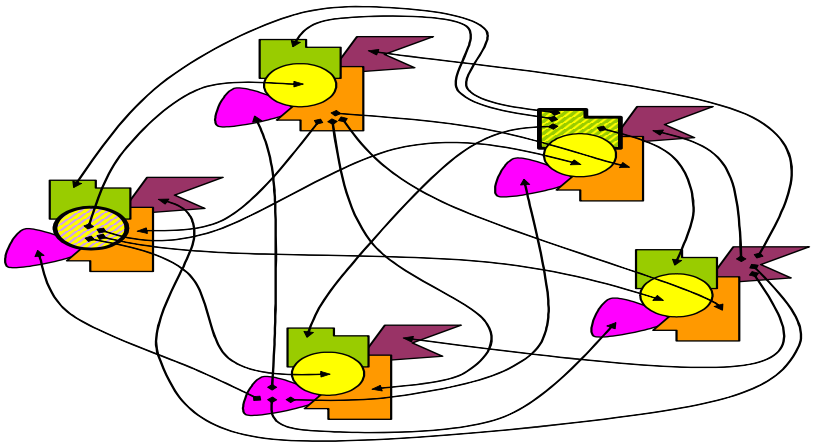


Figure 5: All topic nodes push one content change to their consumer nodes in the replicates

Figure 5 displays the same kind of operations as Figure 3 but with five replicates in play. Such is the functioning of any **HIVE/WARE** application.

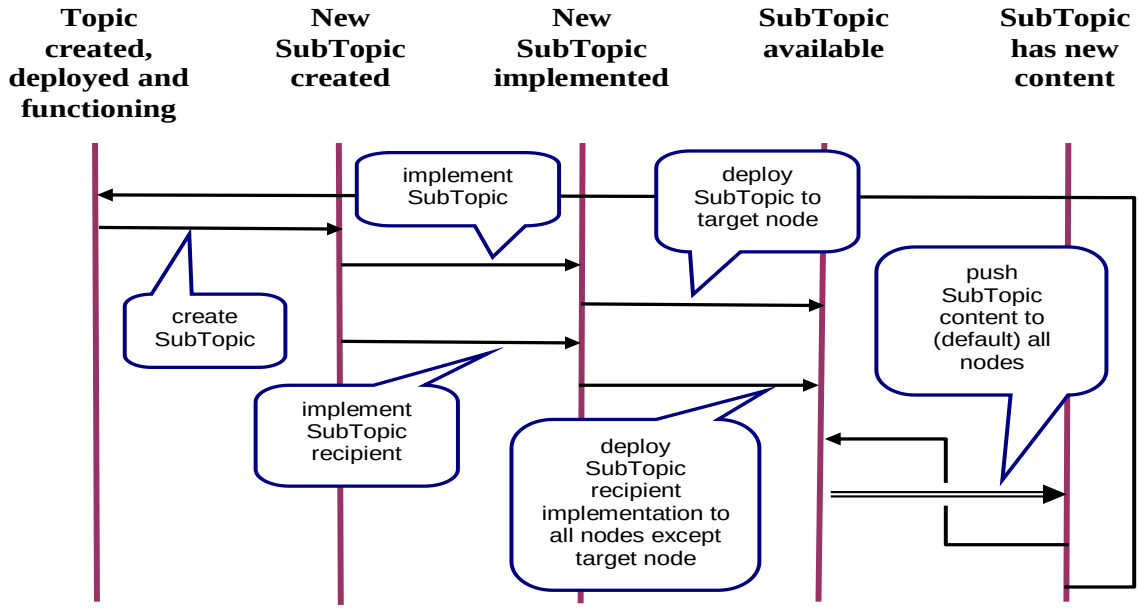


Figure 6: the HIVE/WARE engine

Figure 6 shows the **HIVE/WARE** state engine in UML sequence diagram form.

In case it is not apparent at this point, **HIVE/WARE** applications are peer-to-peer and event-driven. There are no periodic behaviors or servers in the technology which translates to maximum responsiveness.

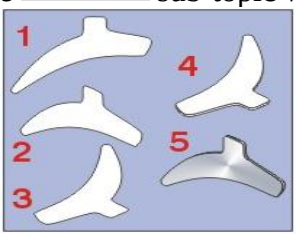
For this paragraph only, graph theory terminology will be used and are shown in italics. Each *node* in graph terminology is a member of three kinds of *trees*. The **HIVE/WARE**'s grammar structure is a single *rooted tree* of any *height* where each sub-topic is a *proper descendant*. Content authors are *external nodes* or *leaves* in the graph whereas context authors are *internal nodes*. Subscriber *vertices* that are neither context nor content author may connect to any *vertex* in the graph as a *free tree vertex*. There is no grammatical problem with a *descendant node* having more than one *ancestor*. Standard graph theory makes, however, no mention of multiple inheritance which is legal in **HIVE/WARE**. At the same time, each topic *node* is a *root vertex* in of a *rooted directed tree* with respect to its recipient *vertices* which is all of the other *vertices* in the graph. Each of the tree's *height* is by default one, but may be more than one if workflow management is desired.

As a result of this design, there are neither concurrency issues nor layers of indirection in **HIVE/WARE** applications.

2.2 Presentation layer and grammar context

The presentation layer has been an outcast in computer science and markup language communities since Noam Chomsky invented the basis for compiler parsing in the '50s. A close psycholinguistic study of language reveals that it can be viewed from four different standpoints: how humans create language, how humans acquire language as a child, how humans express language (i.e., read, write, understand, speak), and how natural language is reflexive (i.e., language is used to describe everything we know, including itself). It is this unified linguistic theory that is the foundation for the **HIVE/WARE** *replicate aggregate of complementary nodes* theory as exemplified in Figure 3 and Figure 5 with the  and  conglomerate objects. Take, for example, the **HIVE/WARE** production of a bicycle. This bicycle is a shared object (in the complementary sub-objects sense) whose presentation is of the graphical kind such as  . A **Specifications** topic is a shared object of the textual kind for which the **DerailleurSpecs** sub-topic might

1. Create an isometric front effect by applying 86.6% horizontal scale and then a −30.00 degree Vertical Skew.
2. Apply a −120.00-degree rotation.
3. Give the object some depth by applying a 0.08 deep extrusion with settings that produce the result shown.
4. Separate, fill as shown and group the assembly.
5. Apply a 120.00-degree rotation and group everything together.

have a textual presentation like . This is not to be confused with, for example, the Derailleur sub-topic which at one time might have a graphical presentation such as 

. It should be intuitively apparent that there is a strong relationship between these presentations and their underlying grammars (linguistics: *sign*, *signifier*, *signified*). It is this relationship which computer science has been ignoring these many years by only concentrating on parsing (computer science: deriving meaning from previously encountered tokens), and which is the underlying foundation of **HIVE/WARE**. The whole is larger, or at any rate, different than the sum of its parts, and when presented as such, the whole exerts a reverse influence (computer science: *disambiguates*) on its parts that obviates the need for token parsing.