TECHNICAL REFERENCE

US PATENT NO. 7,124,362

1. GUIDELINES FOR THE REFERENCE

A. CONDITIONS AND LIMITATIONS FOR THIS REFERENCE

The following is a description of the reduction to practice of the US Patent No. 7,124,362 invention. All statements in this technical description are in full accordance with the issued US Patent No. 7,124,362. All technical representations and statements below are tied by a reference to the claims and specification of US Patent No. 7,124,362. D1 is the reference label that designates the original patent specification as submitted to the various patent offices around the world. D2 is the reference label that designates the US Patent No. 7,124,362 issued October 17th, 2006. And D3 is the reference label that designates the amended claims of US Patent No. 7,124,362 that were issued on October 16, 2006. D1 is an original document owned by Robert Tischer, the inventor, which has been submitted to various Patent Offices around the world for patent prosecution. Its references related to page and line numbers (Dn: page q, lines rr-ss). D2 may be retrieved from http://uspto.gov searching for US Patent No. 7,124,362 Full Text. D2 is the same text as D1 but is in web browser format and does not contain pages or lines as such. Its references are instead by section, paragraph and lines in that paragraph (D3: sect. q, para. rr-ss, lines tt-uu). D3 is the issued patent and may be retrieved from the same site searching under Patent Images. The format of D3 is columns and line numbers (D3: col. q, lines rr-ss).

B. FOR A PERSON SKILLED IN THE ART

The purpose of this Technical Reference (TR) is to provide a working model, or embodiment, of the invention. The goal of demonstrating this embodiment is to demonstrate that a person skilled in the art would be able to construct an embodiment of his own by using similar techniques as were used to construct this embodiment.

C. WITHOUT LOSS OF GENERALITY

This embodiment does not limit other embodiments as set forth in section **2.4 Generalized Document**[1] of the specification. The name of this embodiment is Hiveware® for Word (HfW) where the trade name Hiveware® stands for **H**yperstructured **I**nteractive **V**irtual **E**nvironment soft**W**are. In the remainder of this text and graphics, any particular executing instance of the HfW embodiment is referred to as a hive. Discussion of the HfW embodiment and its particular hive described below will be used

---

[1] Generalized Document: D1: page 9, lines 22-31

to refer to the invention without loss of generality. Because the invention imitates natural language[2] functions, the representation[3] will appear entirely different from embodiment to embodiment, and naturally from HfW hive to HfW hive.

The hive name for the description below is **SmallBusProposal**, which is the name of its root[4] node[5]. For clarity's sake only two authors are functioning in this hive. L represents a generic author to the reader's left and R represents another generic person to the viewer's right. During the course of the following series of table views, R first becomes a subscriber to L and then a content author for several nodes (5).

D. TOOLS AND CONNECTIVITY

1) HIVEWARE ENGINE

This TR uses the term hiveware engine, or Hiveware.exe, throughout to refer to communications programming common to all hiveware embodiments and their instances. The hiveware engine contains the following functionality: peer-to-peer[6], networking[7], document replication[8], and the ability to populate[9] new subscribers[10]. Additionally, the hiveware engine allows any fragment editor executable[11], if permitted by its ancestors, to generate these services and propagate them to any delegatee depending on his given privileges and capabilities. A node's (5) privileges and capabilities are referred to as his entity type[12].

The hiveware engine is an extension of the TCP/IP[13] stack overlaid with whatever evolving DTD[14] that the users are building. It has an industry standard COM (Component Object Model) interface with its own Interface Definition Language

---

[2] imitates natural language: D1: page 10, lines 5-18; D1: page 22, lines 1-11
[3] representation: D1: page 10, lines 15-18; D1: page 15, lines 17-19; D1: page 23, lines 25-26
[4] root node: D1: page 31, lines 23-24
[5] node: D1: page 19, lines 1-8; D3: col. 23, lines 47-49; D3: col. 24, lines 9-11; D3: col. 24, lines 49-51; D3: col. 25, lines 10-12; D3: col. 26, lines 6-8
[6] peer-to-peer: D3: col 22, lines 27-28
[7] networking: D1: page 8, lines 5-27
[8] replication: D1: page 8, line 28 to page 9, line 13; D1: page 19, lines 19-25; D3: col. 22, lines 37-57; D3: col. 23, lines 20-22; D3: col. 23, lines 47-49; D3: col. 24, lines 9-11
[9] populate: D1: page 9, lines 14-21
[10] subscriber: D1: page 9, lines 14-21
[11] fragment editor executable: D1: page 20, lines 25-28; D3: col. 22, line 25
[12] privileges, capabilities, Entity Type: D1: page 23, lines 3-21
[13] Transmission Control Protocol/ Internet Protocol
[14] evolving DTD: D1: page 13, lines 12-14; D3: col. 22, line 26

(IDL) that provides a standard binary interface between the generic hiveware engine and the embodiment. COM is currently limited to the Windows platform so if other platforms, like Unix, were to be incorporated in the engine, another interface would have to be developed. There is only one IDL for all embodiments.

2) INTERNET PROTOCOL USED

Interconnectivity among hives is based on the TCP/IP internet protocol, and could not have been UDP. TCP/IP is connection-oriented and UDP is connection-less. The invention synchronously preserves and manipulates context and content across machines and is not in theory dependent on the type of internet protocol used, but it is dependent on guaranteed message delivery to all other replicate (8) view nodes (5). With UDP delivery of a hiveware message is not guaranteed and hive replication cannot be guaranteed among entities. It is not a requirement that TCP/IP be used if suitable workarounds, for example application-level handshaking and time-outs, could be developed for the hiveware engine.

1) .DLL OBJECT FILES

Hiveware utilizes standard Dynamic-link Libraries (DLL) that are created by node (5) owners. Only editor types[15] are node (5) owners. These DLLs are the practical embodiment of the fragment editor executable (11). A DLL is created or inherited from parent nodes[16]. It is sent to a node's subscribers when the subscriber node is populated (9). It is used by that subscriber to receive updates from the sending author. In Hiveware DLLs are dynamically linked into a running hive. A DLL is object code whose internal addresses are relative. These relative addresses become absolute when they are linked into the application. This linking in process may occur statically or dynamically. The invention uses only dynamic linking so that all context authors may modify their node(s), according to their privileges and capabilities. Since operation is node-based, they may all operate contemporaneously[17]. There are different kinds of DLLs and the invention uses only C++ native code-produced DLLs. That is, the DLL code does not require any other code for the Hiveware communication's engine to function. Multiple authoring could not be cooperative if the contemporaneously developed fragment editor executables had run-time or other proprietary dependencies. This is true because non-hiveware intermediary code is owned and maintained by another entity outside the node owner's control.

---

[15] editor types: D1: page 18, lines 10-23
[16] inherited implementation: D1: page 26, lines 7-12; D1: page 26, lines 25-30; D1: page 32, lines 9-11; D1: page 35, lines 21-23
[17] contemporaneously: D1: page 6, lines 1-4; D3: col. 22, lines 31-36

2) USE OF INTERNATIONAL STANDARDS AND COMPILATION FOR PLATFORM INDEPENT MULTIPLE AUTHORING

HfW was developed using Microsoft's Visual Studio C++ Compiler which creates software for the Windows PC platform. The invention is, however, not limited to the Windows platform since internet connectivity protocols function independently of computer platforms. Microsoft Foundation Classes (MFC) was used to develop the Hiveware.exe engine. MFC wraps the Microsoft development platform Win32 call interface in such a way that software developers can use the International Standard Organization (ISO) C++ computer language to develop the fragment editor executables (11). SGML (Standard Generalized Markup Language) and C++ are the fundamental building blocks for Hive developers precisely because they are non-proprietary standards. Conversely, without international standards recruiting new hive members would be limited to the delegating computer's environment. As such, a hive may have members using other development and run-time platforms like Unix. In that case, when an author develops a DLL, he must also port it to that platform and propagate it as well. He knows what platform was used because during the signup process the author collected not only user information, but platform information as well. Using that information, he would be able to propagate the proper fragment executable to the subscriber capable of running it. For example, if an author designed his hive to run with either Unix or Windows subscribers and he were developing on a Windows platform, he would develop his DLL and perhaps cross-platform develop the SO (Unix object file) fragment executables with equal functionality, and propagate the DLL to PC users and the SO to Unix users. It would be similar for any other platform.

The DEMONSTRATING INVENTION CLAIMS section shows by narrative and pictures the principle elements of the invention's claims.

The FOR HIVE DEVELOPERS section demonstrates the software development techniques used to create and modify multiple fragment editor executables (11) and graphical user interface cues[18]. In specific this section will describe how newly created computer code can, using standard computer development techniques, be added to already executing applications, in this case, the **SmallBusProposal** hive.

2. DEMONSTRATING INVENTION CLAIMS

Six facets to claim 1 of the invention will be presented pictorially below:

1. Multiple fragment editor executables (11)

---

[18] graphical user interface cues: D1: page 11, lines 6-21; D1: page 20, lines 4-11; D1: page 26, lines 1-24; D3: col. 22, lines 29-31

2. Evolving DTD (14)
3. Navigation by graphical user interface cues (18)
4. No servers[19],
5. Replicates (8)
6. Contemporaneous editing (17).

The steps for creating a hive by modifying content and delegating authoring to another subscriber will be demonstrated.

E. SERIES OF SCREEN SHOTS SHOWING L R G AND I

The example is a 2-author standard Small Business Proposal. L refers to the Left author, and R refers to the Right author. Both authors are working contemporaneously (17) on a single virtual document[20] using their two computers connected directly by the internet. The second letter may be R which refers to representation (3) or G which refers to Grammar tree: the underlying DTD structure[21]. The second letter may also be I which refers to the instance tree. The numeral, n, lets the reader know at which point in the series the set of screen shots was taken. Hive snapshots, therefore, will be shown in this format:

LRn	**RRn**
LIn	**RIn**
LGn	**RGn**

.

For example, LR7 LI7 LG7 RR7 RI7 and RG7 occurred after LR6 LI6 LG6 RR6 RI6 and RG6. The intent is to impart knowledge to the reader of the critical parts of the process that takes place in order to build a hiveware virtual document for this particular hiveware application. In the summary, these critical hiveware parts will again be related to the invention in general.

G and I always refer to the structure of the hive and what nodes (5) have been created. G and I are normally not visible. As dictated by the invention, each representation element is technically tied to an instance node.

---

[19] no (central) server: D1: page 12, lines 1-6
[20] single virtual document: D1: page 21, lines 23-24; page 22, lines 12-23
[21] underlying DTD structure: D1: page 22, lines 12-17; page 22, lines 28-31

L1 TYPICAL BEGINNING OUTLINE FOR HFW

The overview topology of the HfW embodiment is shown in Figure 1.

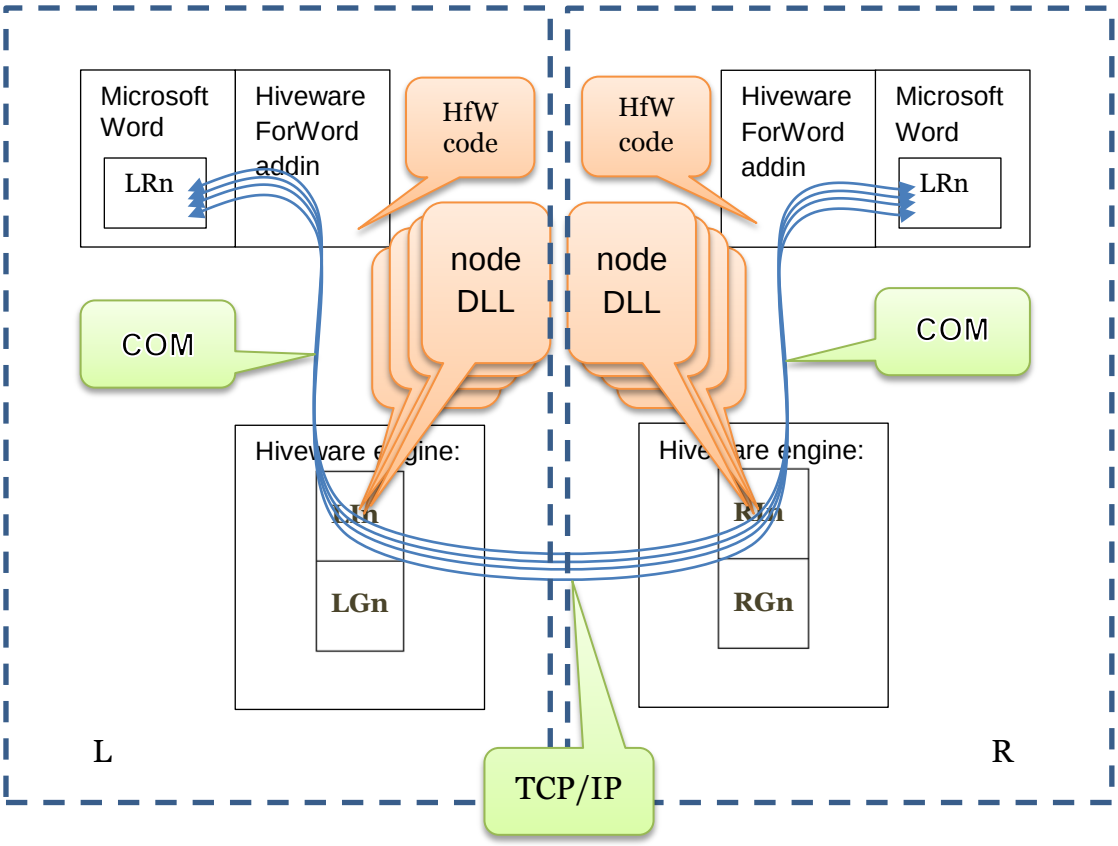


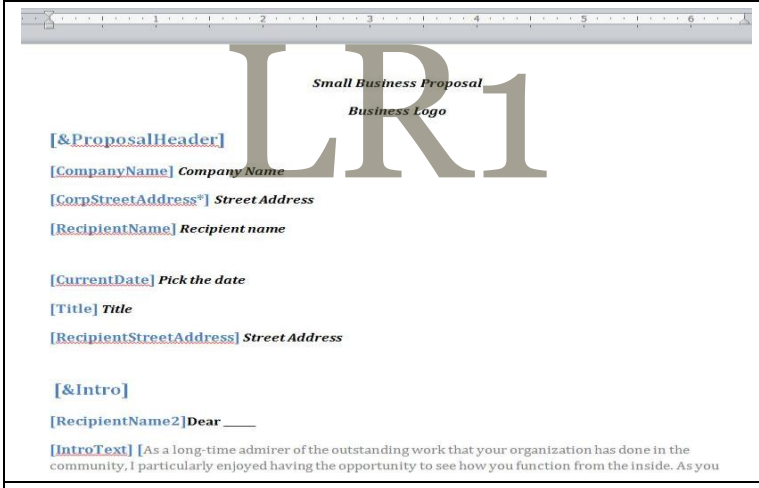
Figure 1: HfW structural diagram for L and R and their fragment editor executables, one for each node

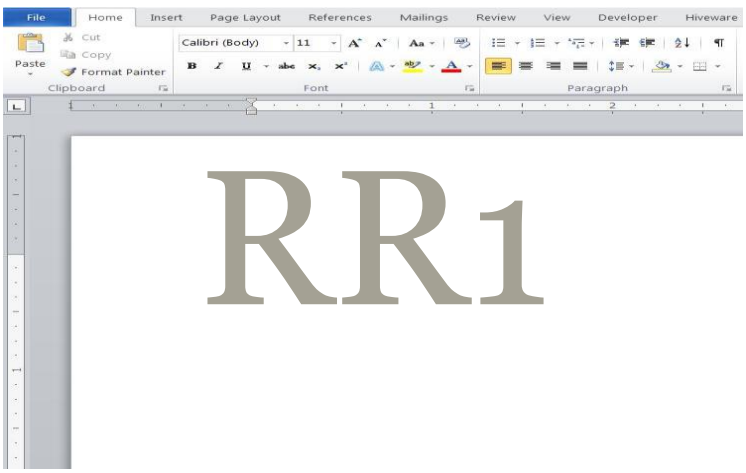
Table 1 shows a hive that begins with a **SmallBusProposal** outline. The HfW developer has chosen to use Microsoft Word as his graphical user interface cue (18) capability as his grammar and instance representations (3) of the Hiveware.exe communications engine. LR1 (Left Representation) shows a template which is a standard Microsoft Word construct. There is no hive activity yet on the other computer so RR1 (Right Representation) remains blank. Both L and R may be located anywhere in the world. Hiveware has already been installed on L so LI1 (Left Instance Tree) and LG1 (Left Grammar Tree) are already running in memory[22]. Normally, these two dialog boxes are invisible and are not necessary to interact with the engine since all interaction with context (another word for Grammar Tree) and structure occur through the representation's (3)

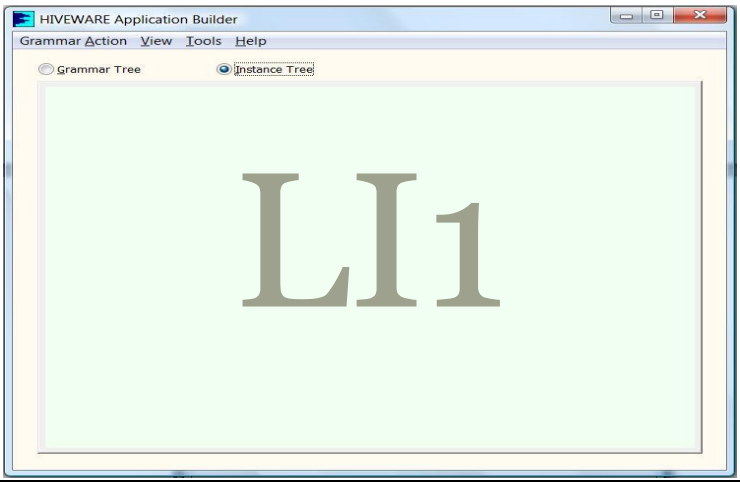
[22] running in memory: D1: page 11, lines 22-30

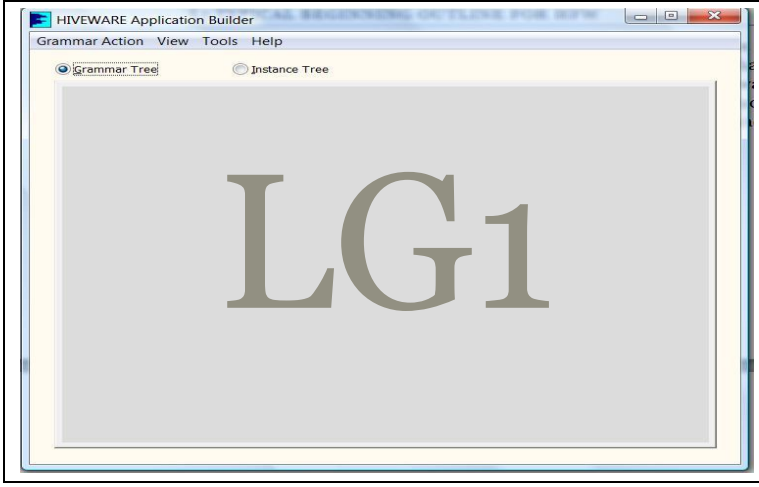
graphical user interface cues. In this case, that representation is Microsoft Word shown in LRn and RRn.

Table 1: Begin with hiveware-ized template









L2 R2

In LR2 the initial **document type declaration** (DTD)[23] is built by saving the LR1 outline. See Y2 menu ribbon options in Figure 2's Hiveware tab.

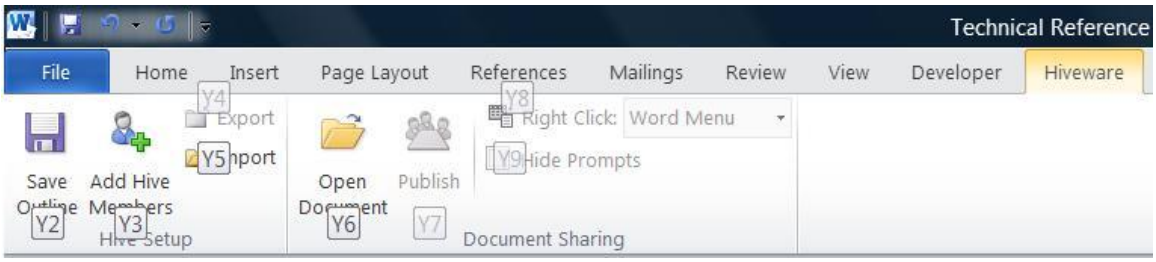
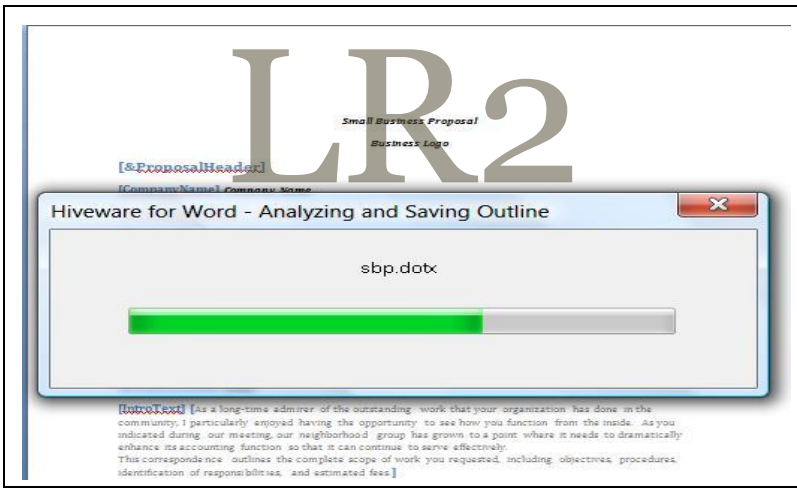
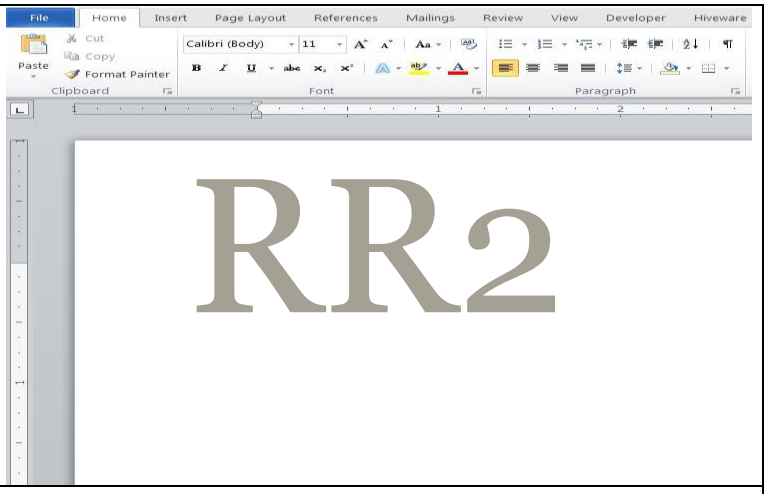
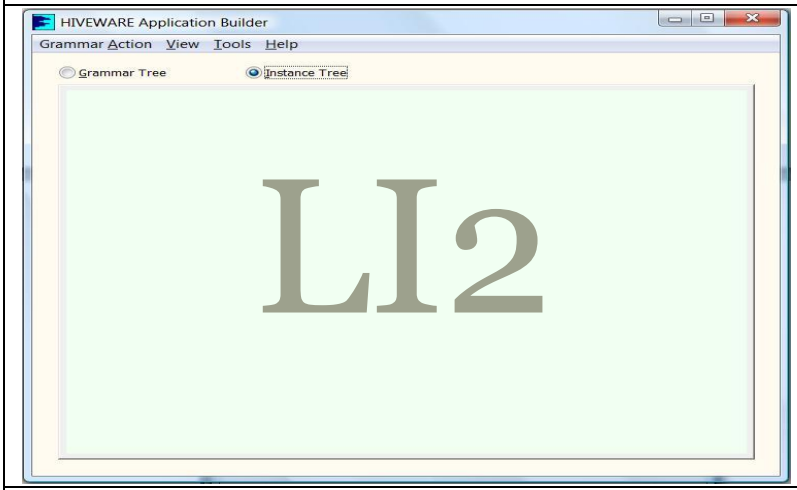
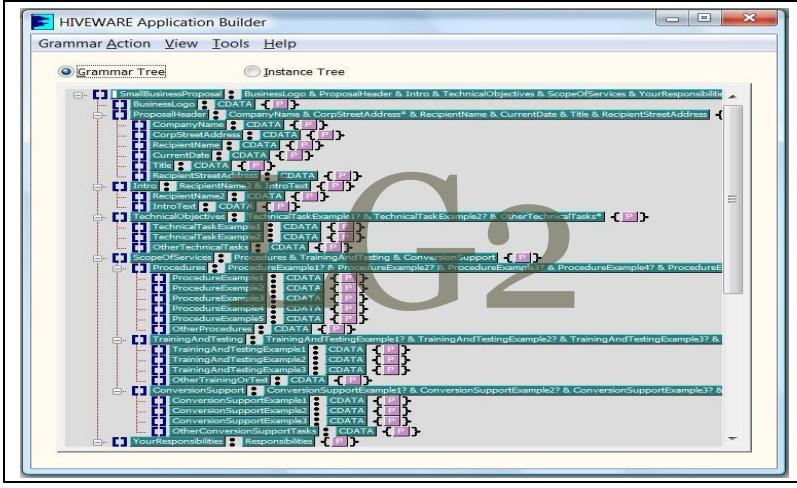


Figure 2: Hiveware for Word's ribbon interface

Executing a Save Outline from the Hiveware Word tab creates a new root context author[24] and appears as shown in LR2 of Table 2 where LG2 shows the resulting DTD structure in L's hiveware engine. Each line in LG2 represents an independent node (5) with a fragment editor executable (11). A default HfW fragment executable is provided for each grammar node as represented by the purple button P. It is these buttons that are used to develop additional node-specific functionality. See Figure 3 to Figure 13 for details on this process.

[23] document type declaration (DTD): D1: page 15, lines 12-19
[24] root context author: D1: page 18, lines 10-13

Table 2: save Hiveware outline

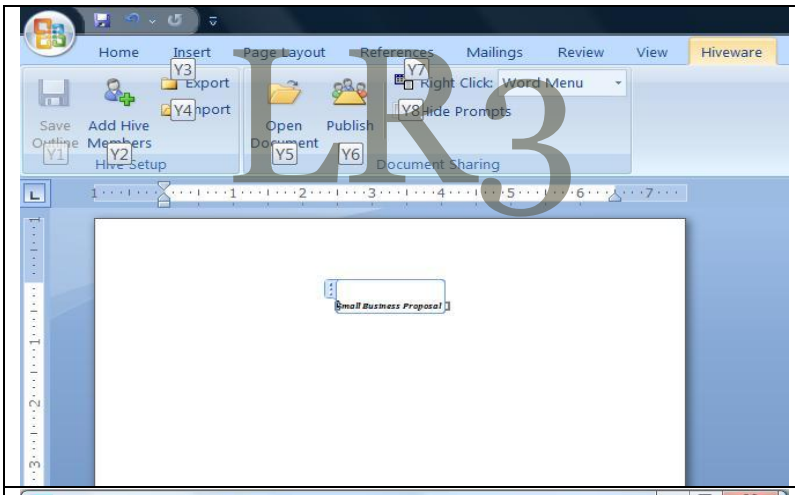
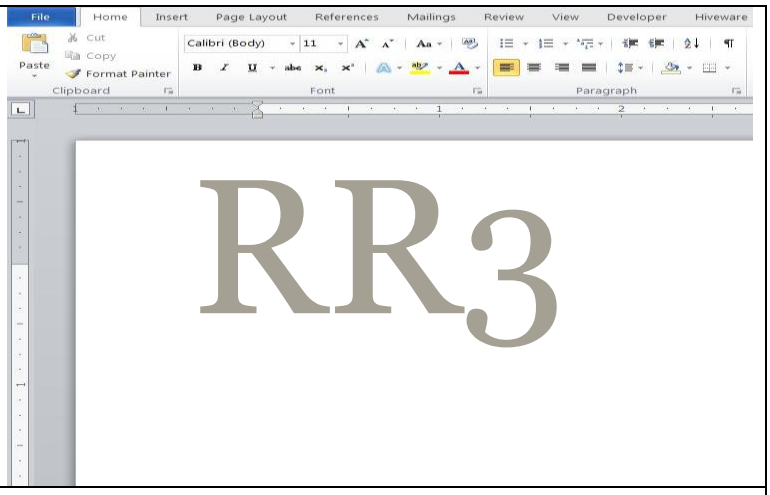
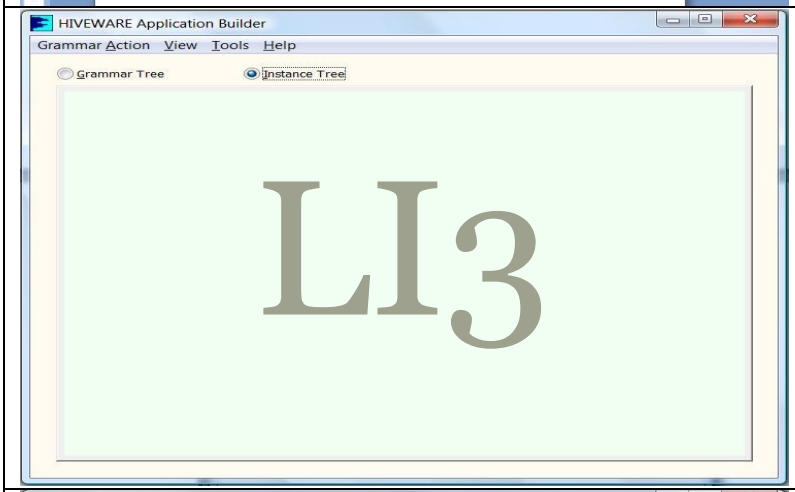
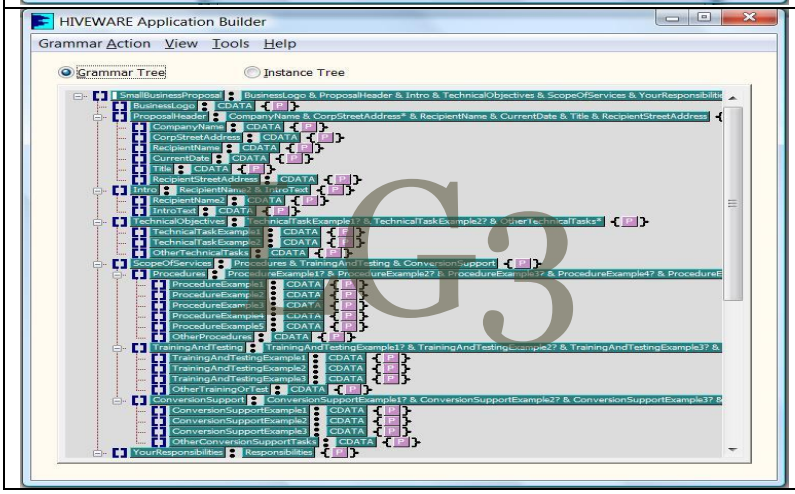
	
	
	

L3 R3

LR3 shows a single (root) prompt[25]. A prompt is an example of a common structural visual cue (25) utilized specifically in HfW. Since an HfW node (5) is always read-only unless you are the author, prompts are used as previewed choices that authors can make at any particular point in the outline (grammar structure) during the writing process. Inherent in Microsoft Word is a developer's application programming interface (API) that allows the detailed access and manipulation of Word content controls (CC) and their access privileges. Read-only is a significant graphical user interface cue (18) that is realized in Word with the use of this CC mechanism. Read-only and shading are common structural visual cues (25) that guide the authors and subscribers so that the entire virtual document (20) retains its integrity[26]. Here is an example of a Word read-only CC:  . Since the text is read-only, the subscriber is prevented from changing the contents. For each and any node the engine can specify CC as either read-write or read-only depending on its node privileges. For other examples of a read-only CCs, see LI12 in Table 12.

[25] common structural visual cues: D1: page 20, lines 4-11
[26] mutually exclusive and exhaustive DTD: D1: page 22, lines 24 27

Table 3: Initial Prompt

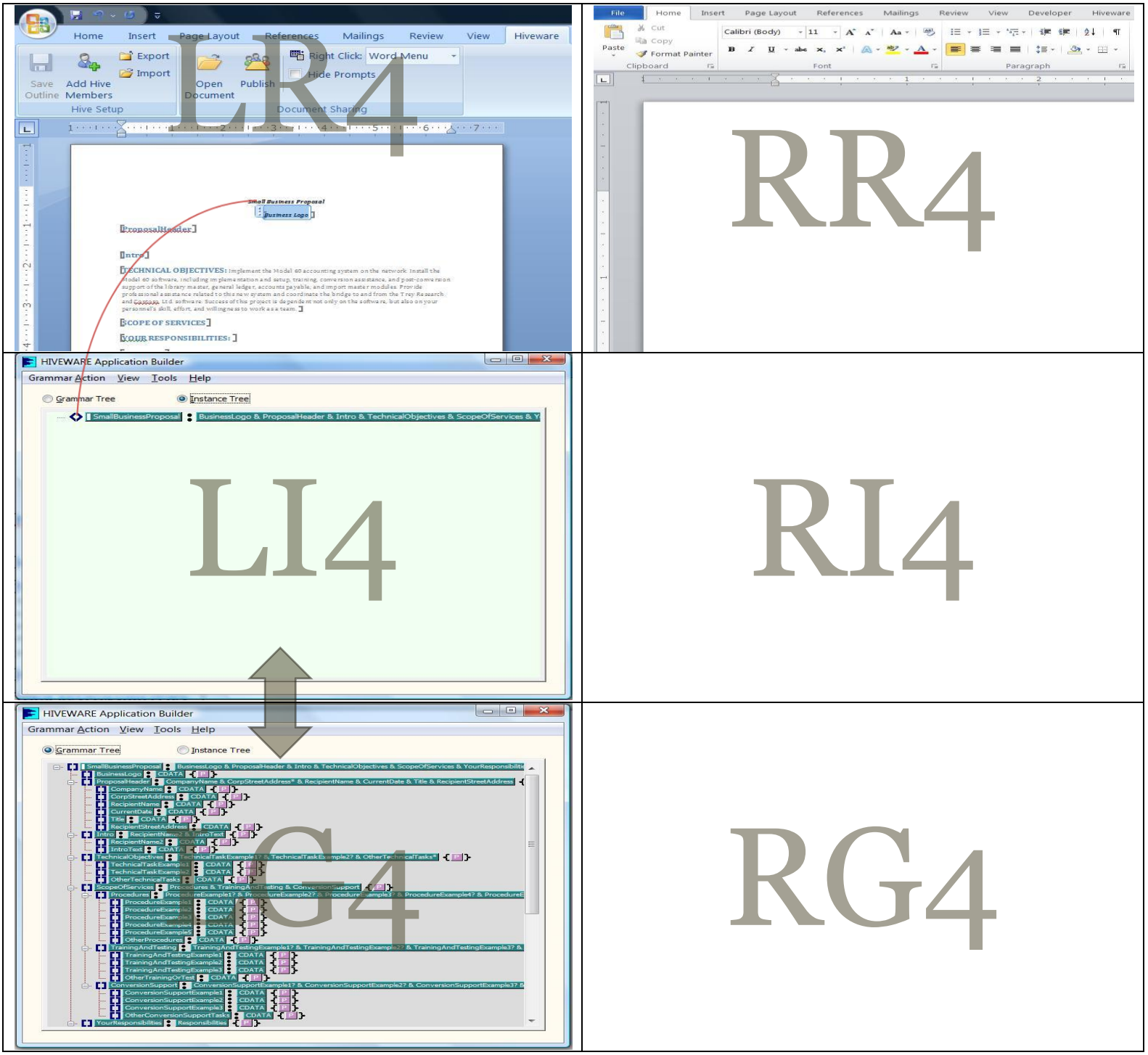
L4 R4

In L4 the initial root prompt (25) has been published. Notice that one level down in the outline, the next series of prompts is shown on the screen. The Word prompt system constitutes navigating by graphical user interface (18) in that it guides the author in the text growth possibilities without exposing the document to an editing free-for-all. Not even the author can violate his own outline grammar. Prompts are navigational cues (18) in the context of a particular outline grammar (which can itself change (14)). Programming the HivewareForWord Addin[27] portion of the fragment executable (11) utilizes Word's inherent developer's API. Specifically, each node in the engine makes calls to Word over COM via the API and vice versa. Refer to Figure 1 for the Addin's locations in the HfW 2-author embodiment.

The engine has now created a node (5) in the LI4 instance tree and the red line indicates the one-to-one COM connection from the HfW API to its corresponding node in the instance tree seen in LI4.

[27] Addin is a term Microsoft Word uses to refer to the product's ability to incorporate external pieces of code into their application.

Table 4: Initial publish

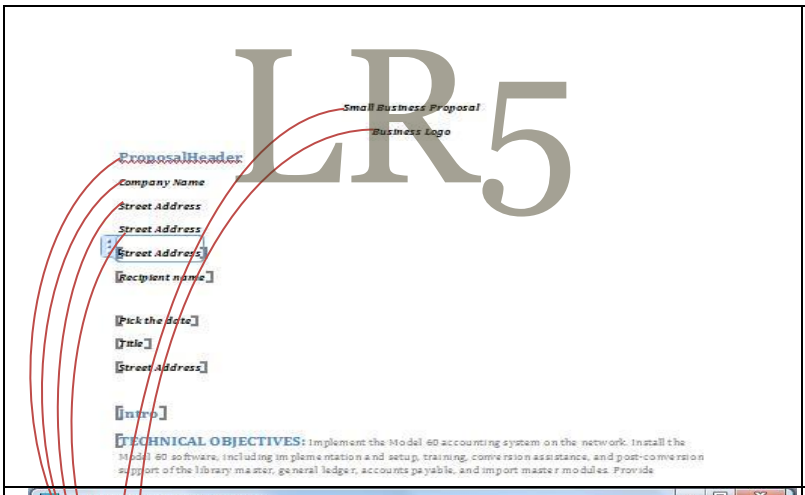


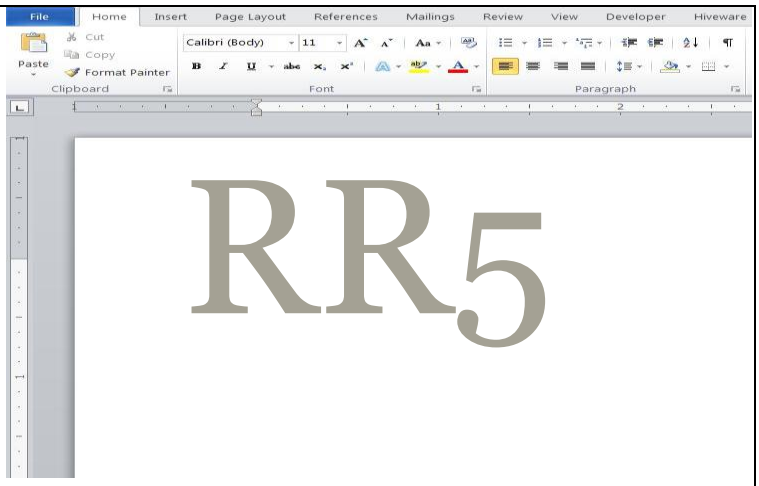
L5 R5

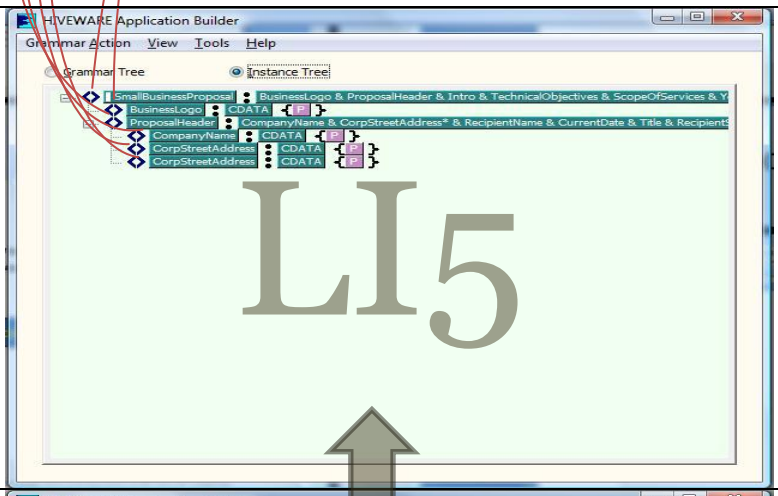
Four more prompts (25) have been published in LR5. All the entries above the blue box in LR5 are published and all the entries below the blue box are prompts. The red lines in LR5 to LI5 each start at a Word content control and are connected to an instance tree node (5). Each is a context author[28] at this point. In the instance tree, each connects to a reference counted DLL. In the HfW hiveware embodiment, the Word API, the common HivewareForWord C++ code (HfW Code), the COM link, plus the node-specific DLL (Node DLL) constitute that node's fragment executable (see Figure 1). Notice that there are now two Street Address instance nodes and yet another Street Address prompt. This demonstrates the generic implementation in the engine of the * (star: zero-or-more) SGML occurrence indicator[29].

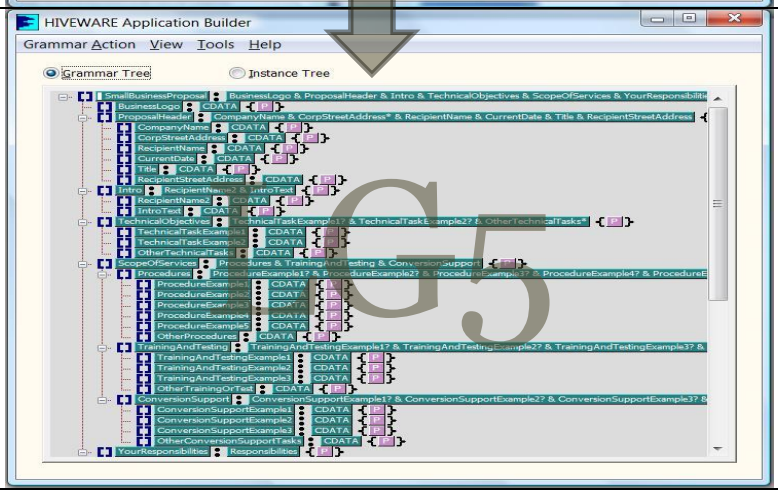
[28] context author: D1: page 18, lines 14-18
[29] SGML occurrence indicator implementation: D1: page 25, lines 19-24

Table 5: Publish other prompts





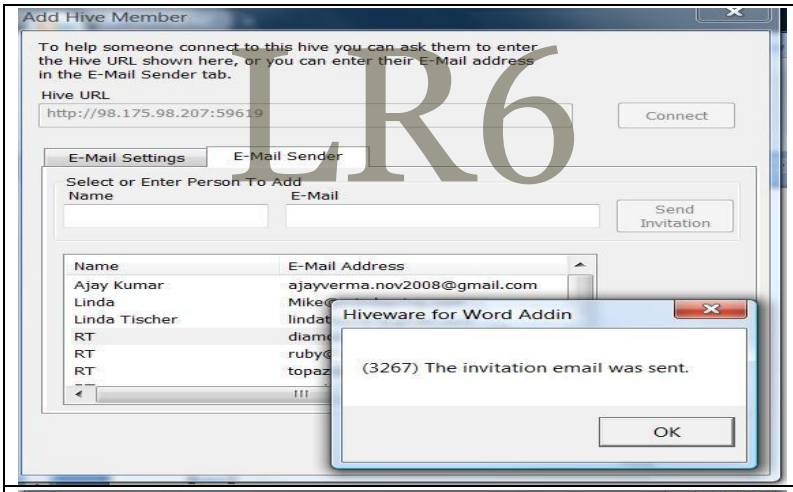
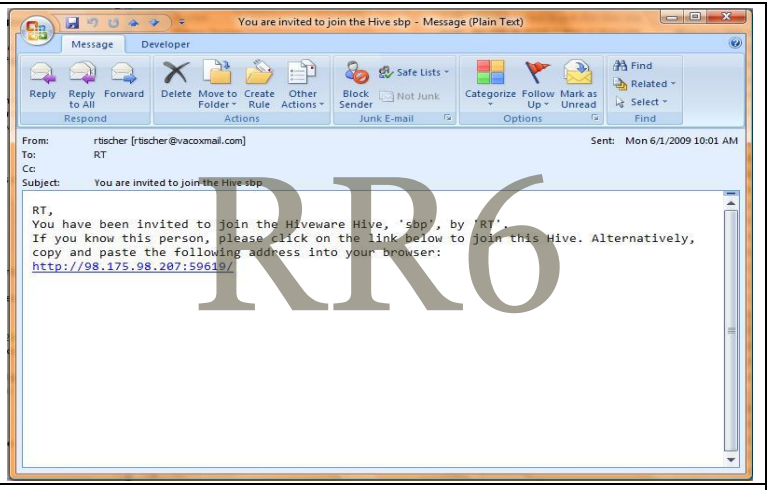
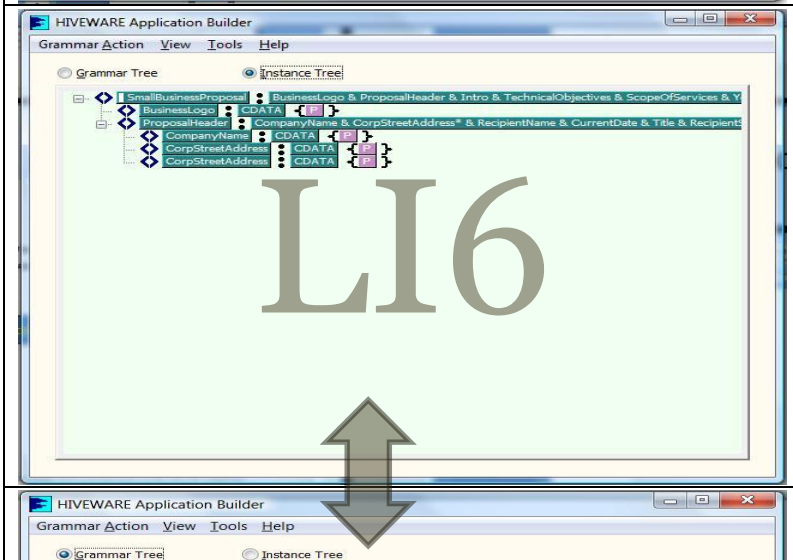
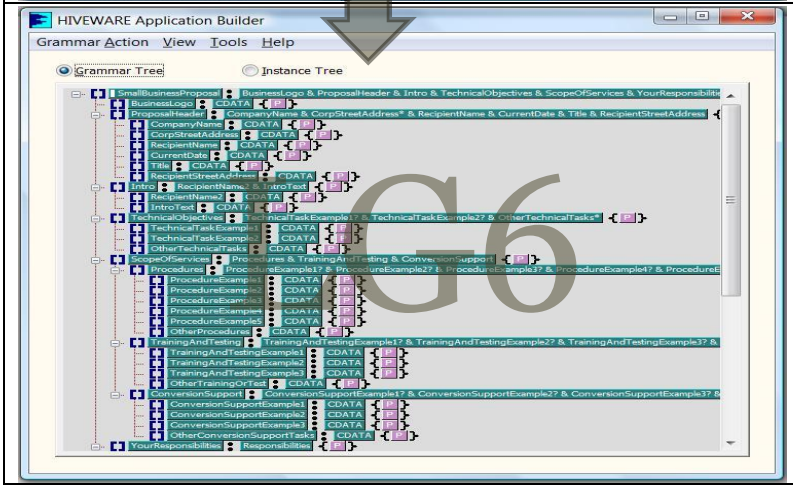




L6 R6

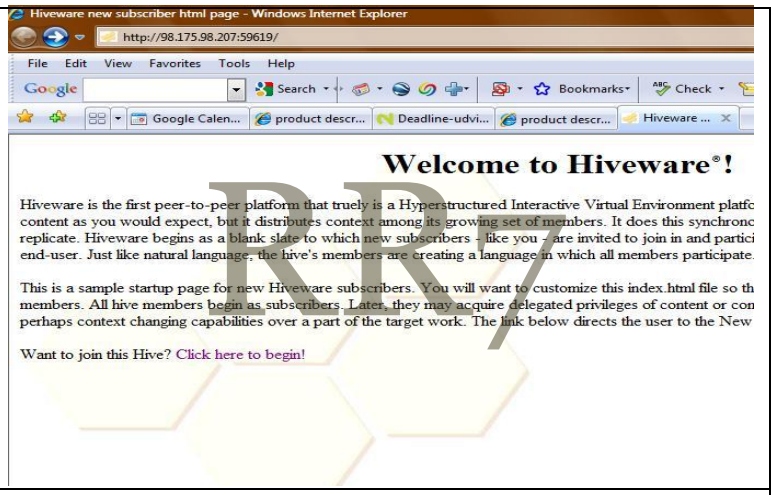
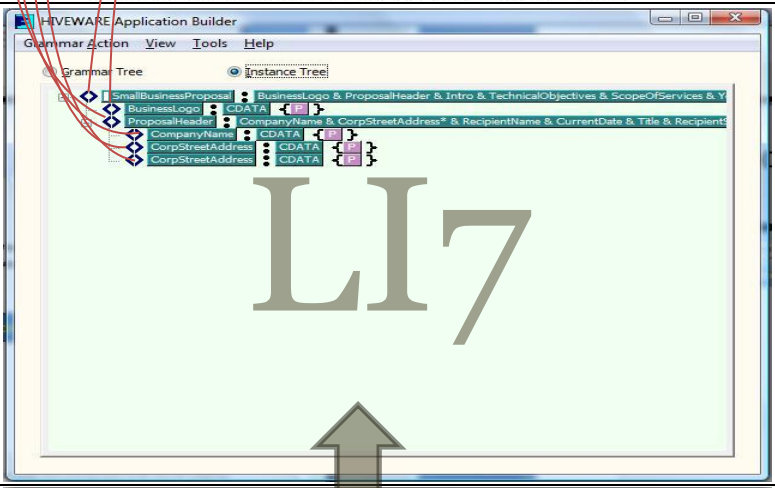
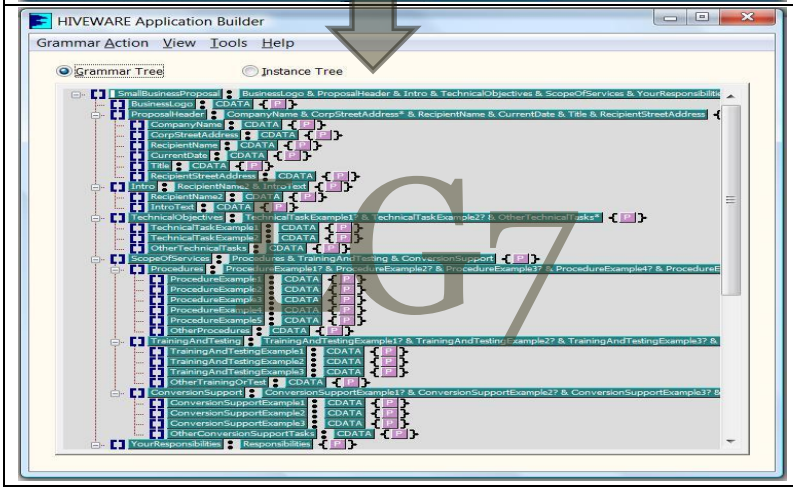
The hiveware engine contains ordinary SMTP email functionality so that L can send R an invitation to join his newly minted hive. This is seen in LR6 where he just finished sending the invitation. RR6 shows the opened email. RR6 clicks on the email link which brings up RR7. Each hiveware node (5) contains code that can respond and issue HTML GETs and POSTs so there is no need for a central web page server (19).

Table 6: New subscriber email

LR6 	RR6 
LI6 	RI6
LG6 	RG6

There are no other participants in this new subscriber process other than the internet service providers at each end that carry the network packets back and forth. This allows new users to use today's well-known email and browser technologies to give and gain access to a hive in peer-to-peer fashion (6). This new subscriber process signup access is an integral part of any hiveware node which means that any hive node can potentially signup any other person who has access to the internet and email.

Table 7: Hiveware Welcome browser page

LR7 	RR7 
LI7 	RI7
G7 	RG7

RR8 shows the six published parts of the document on L now replicated (8) on R. Notice the blue background shading on the nodes (5) in RR8. This and the fact that on R the blue-shaded text is read-only, constitutes graphical user interface navigational cues (18) that influence the subscriber's comprehension and direct his actions so that they are contemporaneous (17) with the author. Each context author (28) node in LR8 may push[30] content or context changes to his observer, RR8, at any time.

The light-blue dashed lines from LI8 to RI8 indicate that the L DLL is TCP/IP communicating to their RI8 DLL counterparts. Each RI8 node DLL contains a listener whose Internet Protocol (IP) end point is known to the RI8 DLL. Hence, RI8 DLL nodes have an observer list of end points to which it may establish a connection at any time. The significance of the dashed line is that the connection is not kept alive, but rather is established only at time of use. If no author is making a change anywhere, there is zero network traffic, but yet all may be working locally on their document contemporaneously (17).

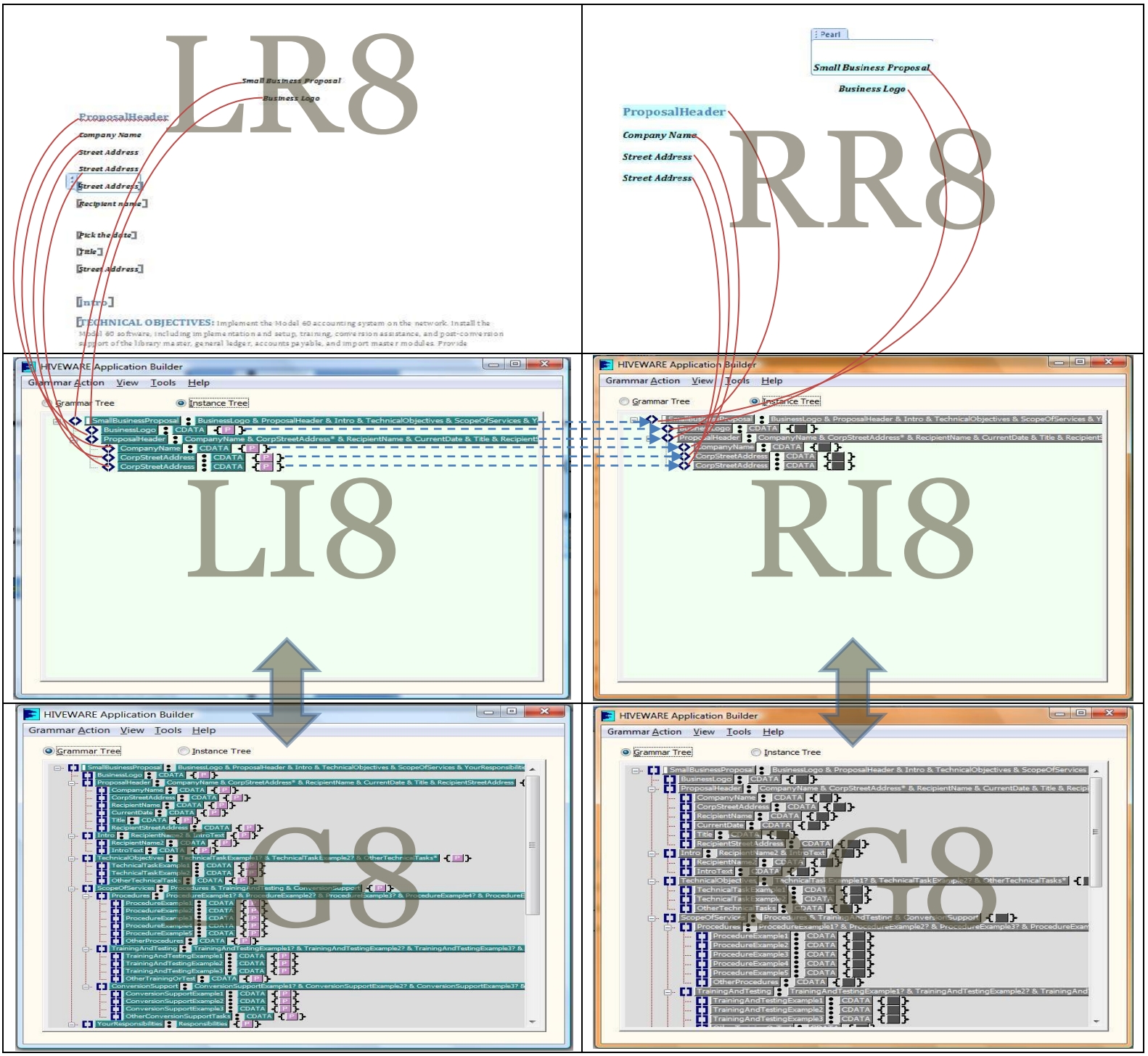
It is important to note that the latest state of the distributed document is kept in memory in the latently connected computers. If a hive member turns his machine off or it crashes, he merely has to reboot. Since the registry keeps three or more of its neighbor's end points, any one of them could be used to background-re-populate (9) his rebooted computer. As a last resort, a hive replicate (8) may Export and later Import a snap-shot of his document (see Export and Import ribbon choices in the Hiveware tab shown in Figure 2).

Populate (9) has created the RI8 and RG8 replicates (8) and their fragment DLLs (11) have created their corresponding representation (3) nodes (5) in RR8. There is now an independent direct socket connection, when needed, from each published author node in LR8 to RR8.

For both RI8 and RG8 their nodes are gray which indicates that each is owned by someone else on another machine. Gray also indicates they are un-changeable and un-editable.

[30] push: D1: page 8, lines 7-9; D1: page 8, lines 12-15

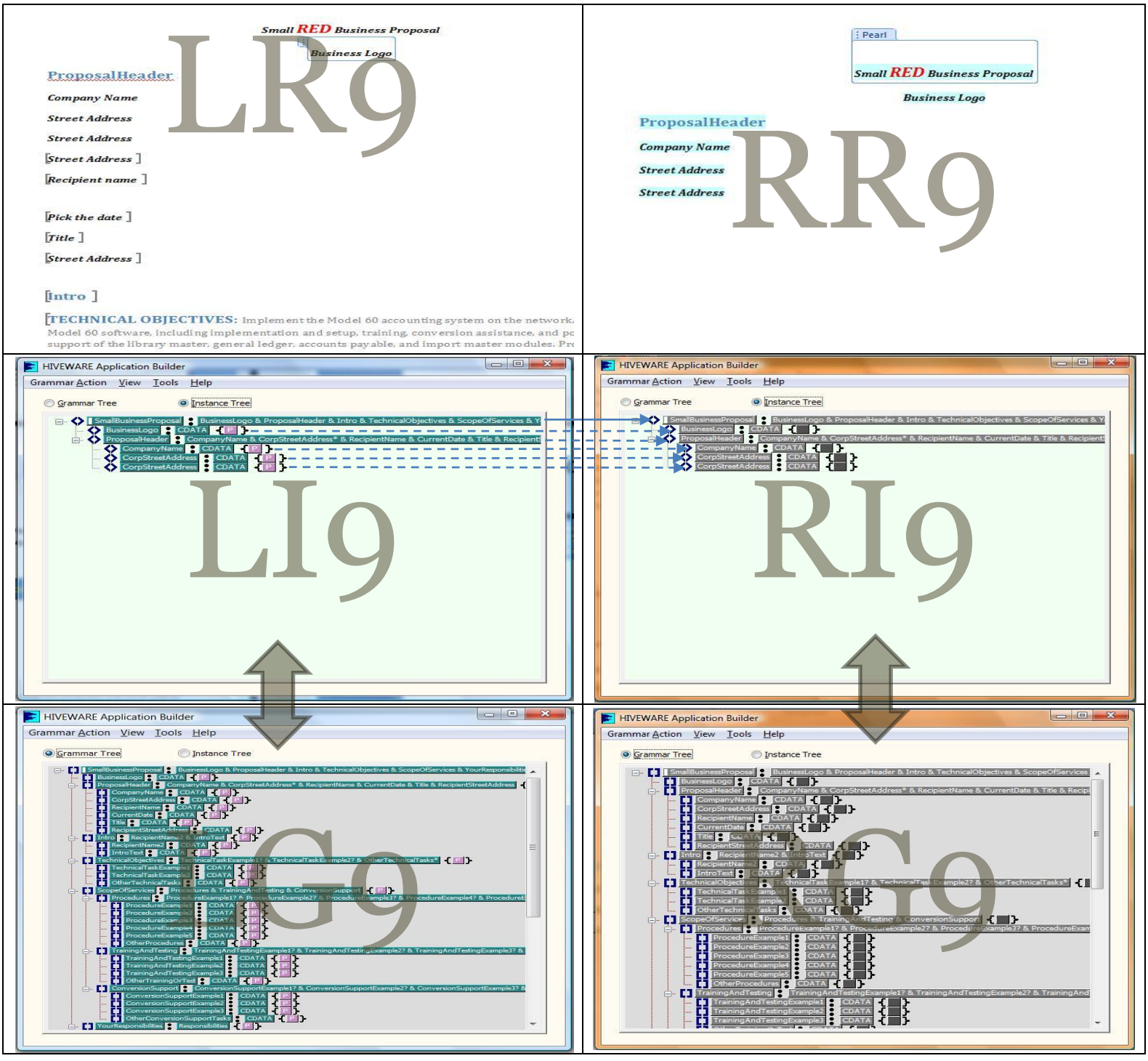
Table 8: Six L entries replicated on R



L9 R9

LR9 makes a content text change by adding the word **RED** to the root **Small Business Proposal** node (5). RR9 shows that this change was successfully pushed (30) to it. Note that as an added context informational cue (18), RR8's root content control has the label **Pearl** which is the name of the author making the change. Being able to show the node author's name allows participants to be aware of the document's diverse user environment. The solid blue line between LI8 and RI8 root node indicates that the connection is only used during the content pushing whereupon it is closed until next use.

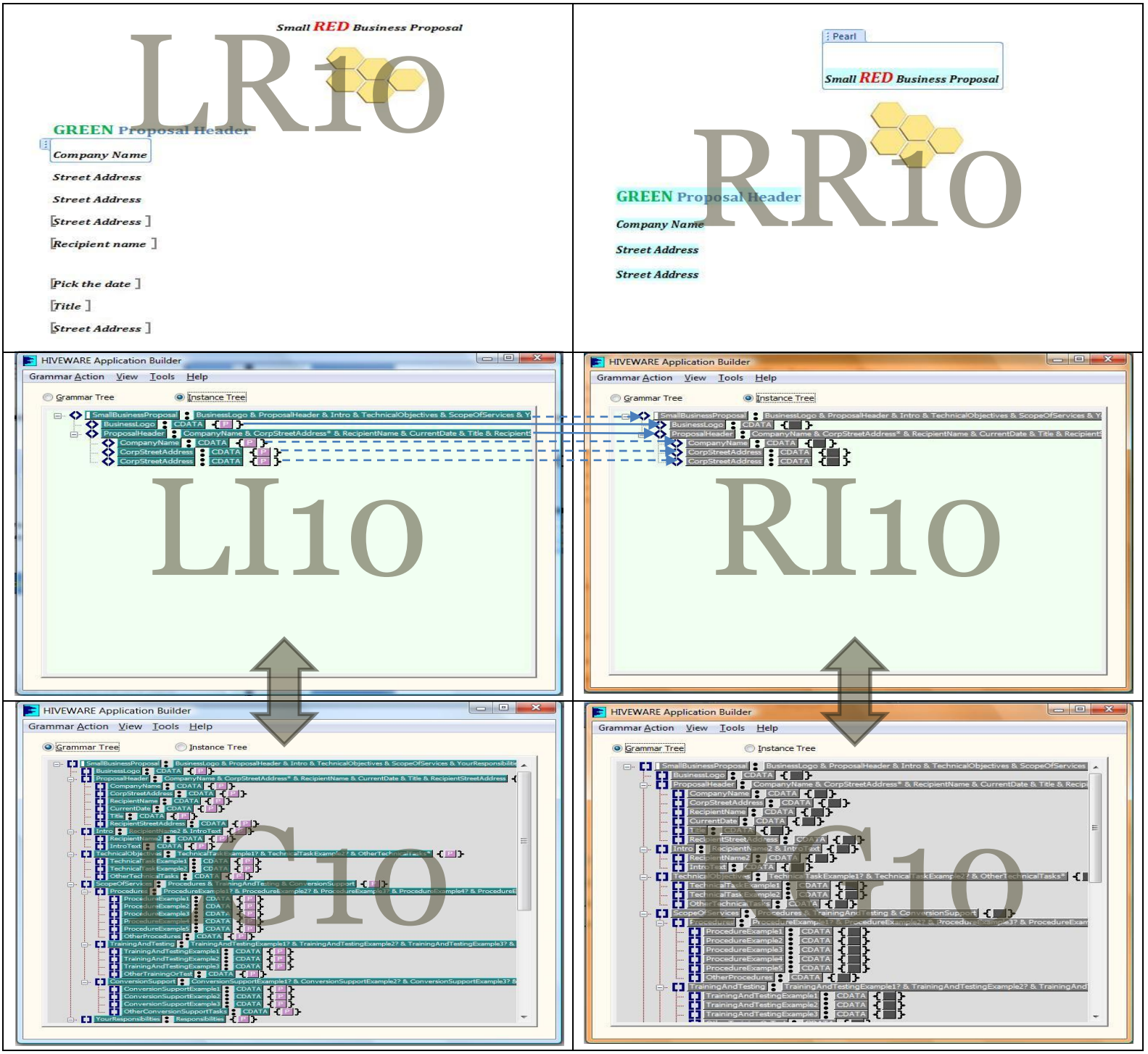
Table 9: Small RED Business Proposal content change



L10 R10

LR10 shows two other content entries, the Business Logo,  , and the addition of the word **GREEN** to Proposal Header. These changes are automatically pushed (30) to R as seen in RR10.

Table 10: and GREEN content changes sent by L and received by R



L11 shows Company Name and street addresses. Notice that Company Name is being content-delegated[31] to R. (Followed by the first Street Address and then the second Street Address which are not shown). RR11, the delegatee, no longer shows any blue shading behind these three fields. The entities on LR11 are now read-only and the corresponding content controls on RR11 are read-write. Since there is never any more than one author at a time that has read-write control over screen content, this illustrates the unambiguous control of DTD context pieces in general.

In all the tables so far the grammar structure created by the Save Outline action at the beginning means there has been no change to the DTD (i.e., LG and now RG). This is atypical of a document and certainly is not representative of how linguistics which hiveware emulates (2) works. In fact documents begun without a template will have no beginning structure except for the initial blank prompt. LI10 shows a **Change Outline** menu option. Its function is to create new sub-nodes[32] and add or modify content models to sub-nodes. Change Outline is the user representation (3) for context editing[33]. It is this functionality that demonstrates the evolving DTD (14) claim.

The greens and the grays of the nodes in the LI11 and RI11 instance trees reflect the mixed ownership.

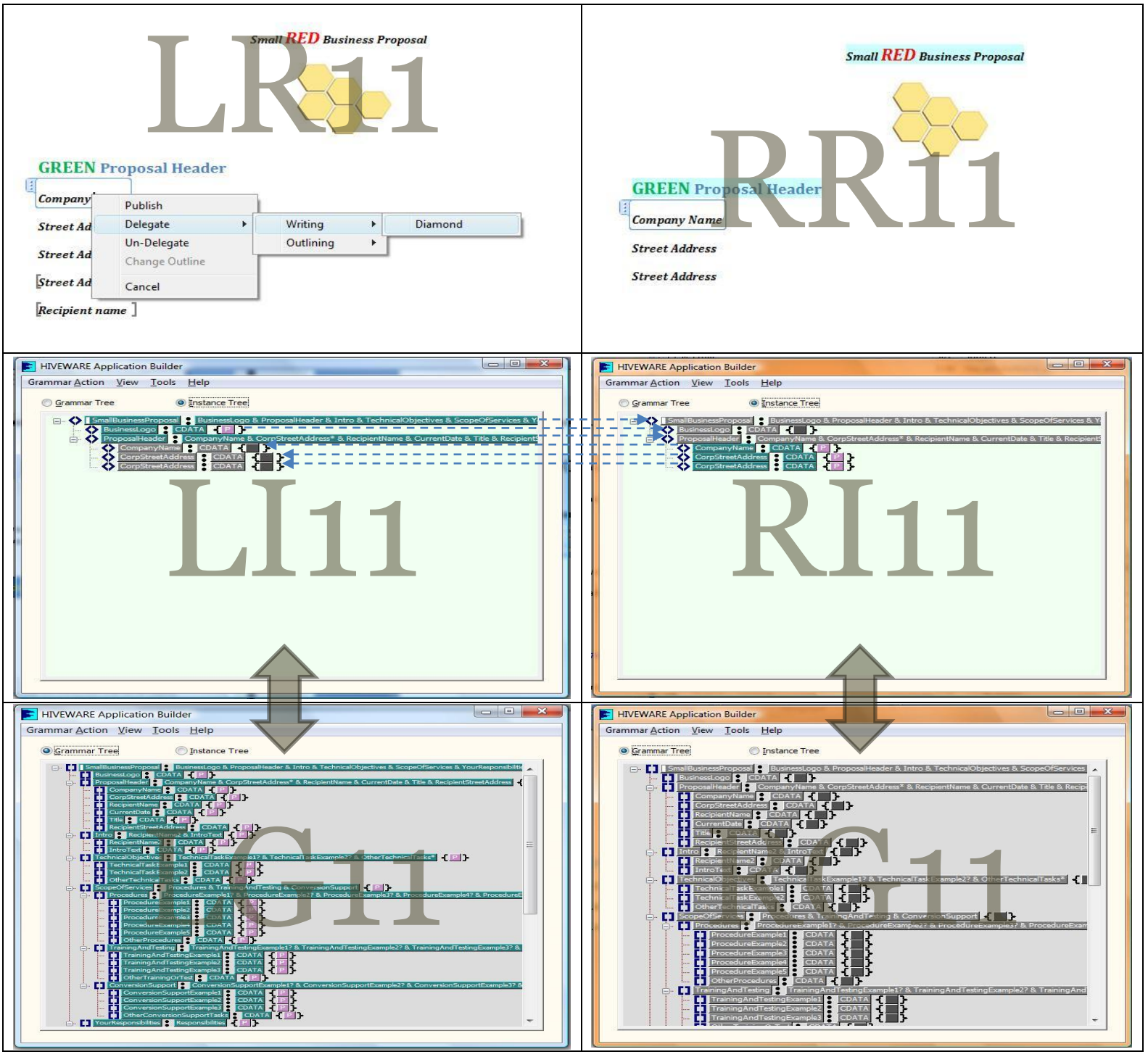
---

[31] delegate: D3: col. 23, line 6 to col. 24, line 24; D3: col. 24, line 34 to col. 26, line 21

[32] sub-node: D1: page 19, lines 13-15

[33] (root) context editor: D3: col. 23, lines 8-10; D3: col. 23, lines 56-58; D3: col. 24, lines 36-39; D3: col. 24, lines 56-59

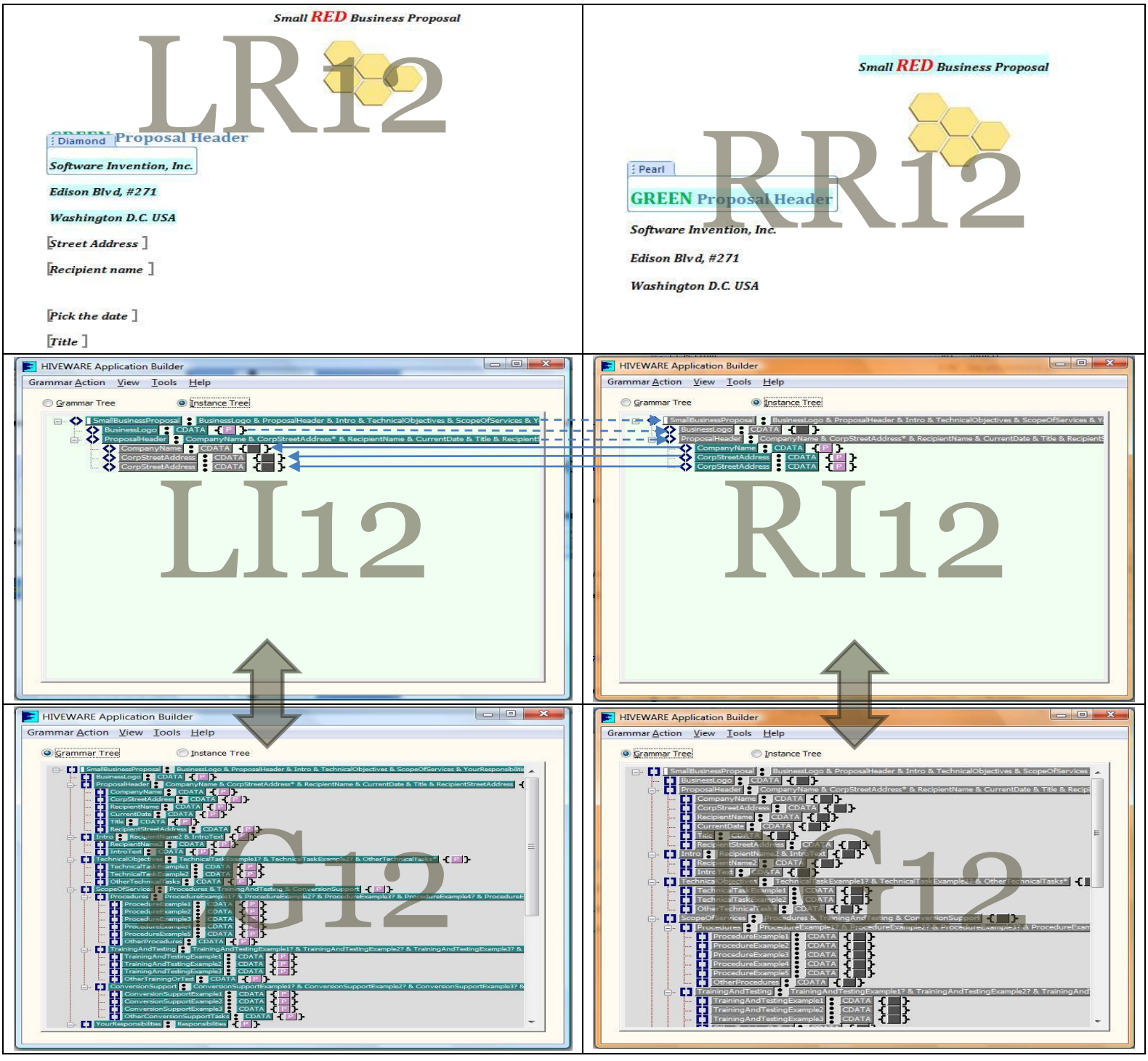
Table 11: Delegating content authoring of Company Name and Street Address to R



RR12 shows a content change by R which appears on L shown in LR12. In fact, content changes may now occur at will and contemporaneously (17) along these coordinated connections.

Context changes may also occur if the node author is either the root context author or is a context author (28). LI12 and RI12 show a mixed ownership allowing contemporaneous (17) content edits. Context editing, on the other hand, would result in the same sort of mixed ownership picture except the grammar trees would be affected as well. The only difference between content and context delegation is that the focus of the change is on the grammar tree instead of the instance trees. The end result is grammar trees with mixed and complementary ownership (greens and grays).

Table 12: R send address content to L. Read-only blue-shaded cues.



3. FOR HIVE DEVELOPERS[34]

A node's (5) code behavior (implementation) is alterable. The process for altering the code is shown below. In both the grammar and instance trees the DTD content model is shown using a format that is reminiscent of Chomsky's normal form except the resulting parsed tree complex rather than binary. For any production rule the item to the left of the colon is the non-terminal. To the right of the colon is the production rule with its syntax. Attached to each production rule is the semantic action. The semantic action is the block of code associated with the production rule that gives it its implementation. A parent's rule's implementation in hiveware is accessible which makes the implementation inheritable (16). This is a complete hiveware rule:

where the non-terminals in this rule are TechnicalObjectives, TechnicalTaskExample1 and 2 are adorned with the optional (?) occurrence indicator and the OtherTechnicalTasks which is adorned with the zero-or-more occurrence indicator. This production rule has two AnyOrder (&) connectors. And this rule is running a block of DLL code represented by the purple button. Figure 3 shows the complete set of rules for the SmallBusProposal hive. A running hive replicate (8) is a parsed tree whose nodes are connected (only when change takes place, however) to all other hive members for that particular node, which themselves are running parsed trees that are all replicates (8). The difference between hiveware and standard computer science methodology is that computer science methodology discards the parsed tree after compilation while hiveware always retains it. For hiveware, context and content changes are made to the tree and are regarded as fix-ups to the receiving trees. In other words, when an author anywhere makes a change, only his parse tree is up-to-date while all others for that node are temporarily invalid. It is the hiveware engine's task to push (30) that change to its subscribers, thus repairing the invalid tree and restoring it once again to replicate state.

The interface actions on the controls may occur directly in the HIVEWARE Application Builder either on the grammar or instance tree, or they may be projected to the representation (3). For an example of content change representation (3) see HfW's LR11 in Table 11. But for demonstration purposes, the builder will be used below to make a single code alteration on a single grammar node.

---

[34] It is assumed that for this developer section the reader is trained in the basic terminology for SGML and Computer Science.

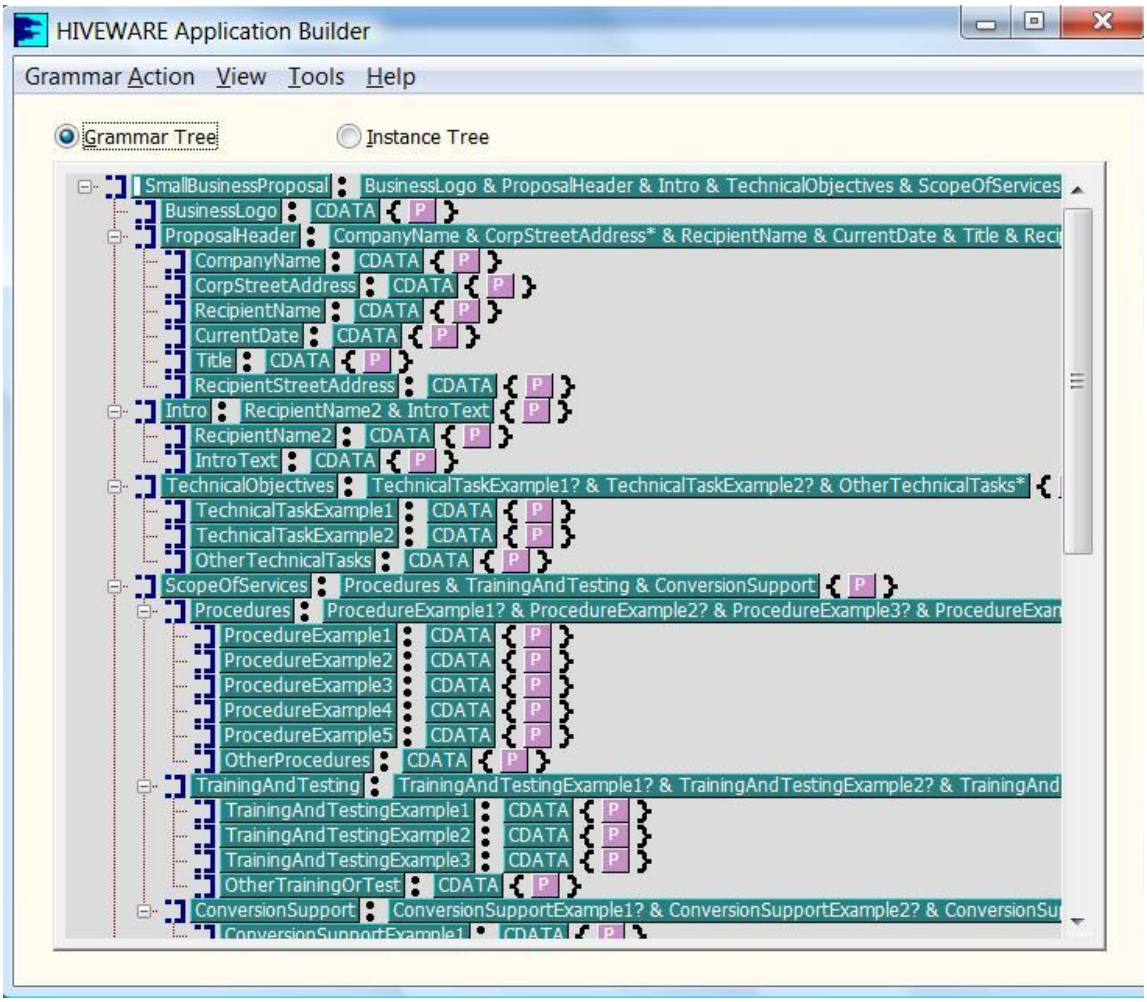


Figure 3: SmallBusProposal grammar tree

Figure 4 shows the SmallBusProposal's content model (SGML term)/production rule(computer science term) being manually typed into an edit box.

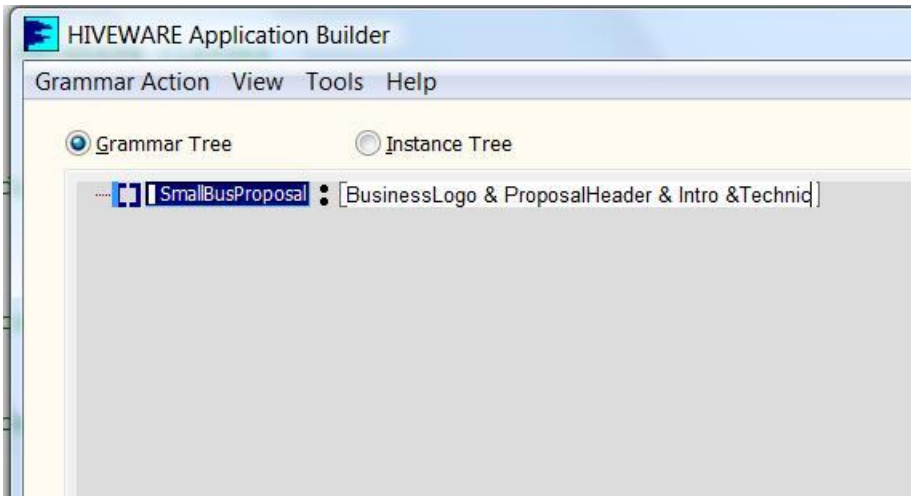


Figure 4: SBP's content model typed in

After a carriage return (CR), the production rule is built. The non-terminals in the root node now become non-terminals below the root. In Figure 5 there are three remaining non-terminals to terminate: ProposalHeader, Intro and TechnicalObjectives. The button legend is shown and indicates whether there source code has been generated, compiled and whether the DLL has been developed or not.

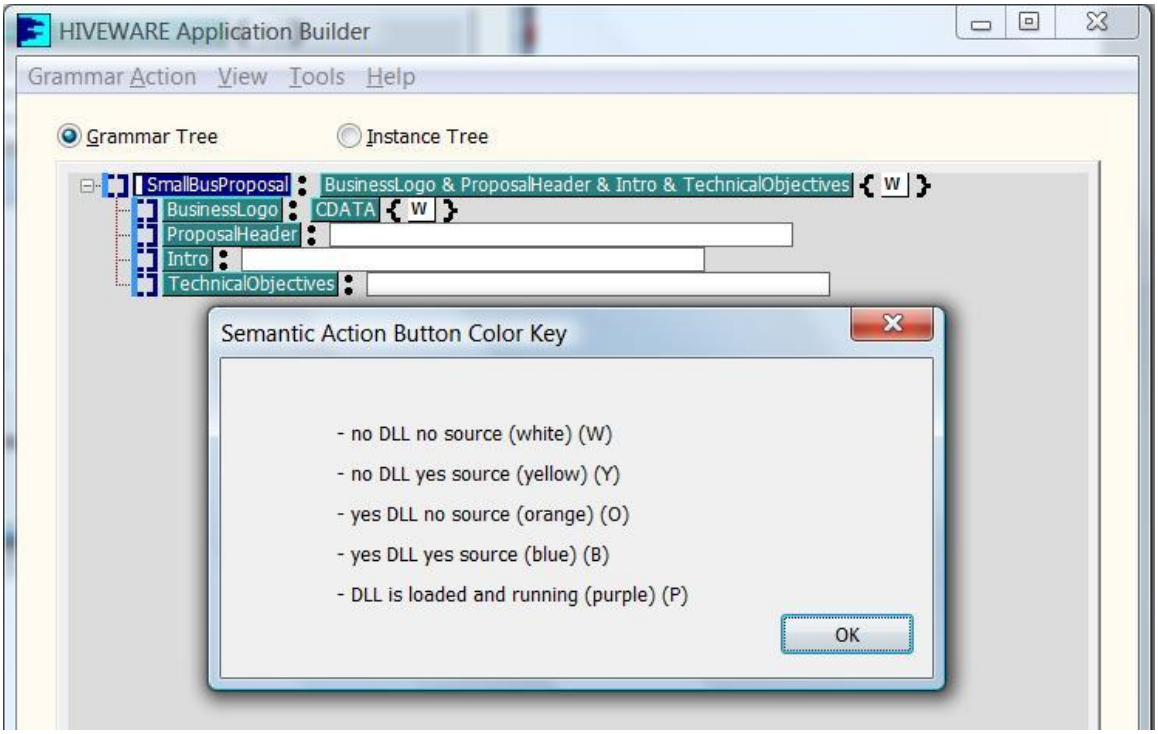


Figure 5: Semantic Action Color Key

The yellow button indicates that the source code has been clicked once and the source code generated for the BusinessLogo sub-node (32).

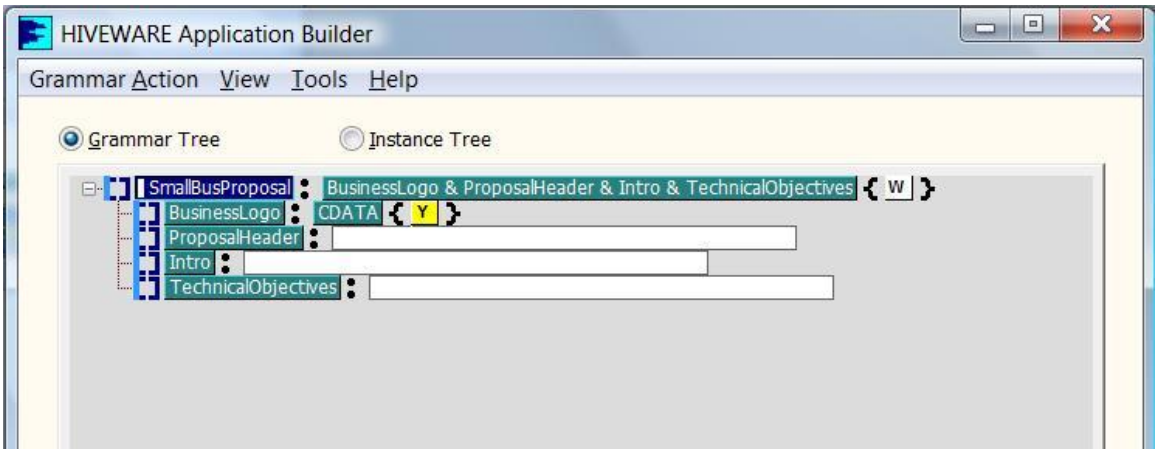


Figure 6: Yellow code button meaning generated source

Clicking on the yellow semantic action button brings up an instance of the Visual Studio C++ compiler loaded with the generated BusinessLogo node code[35].

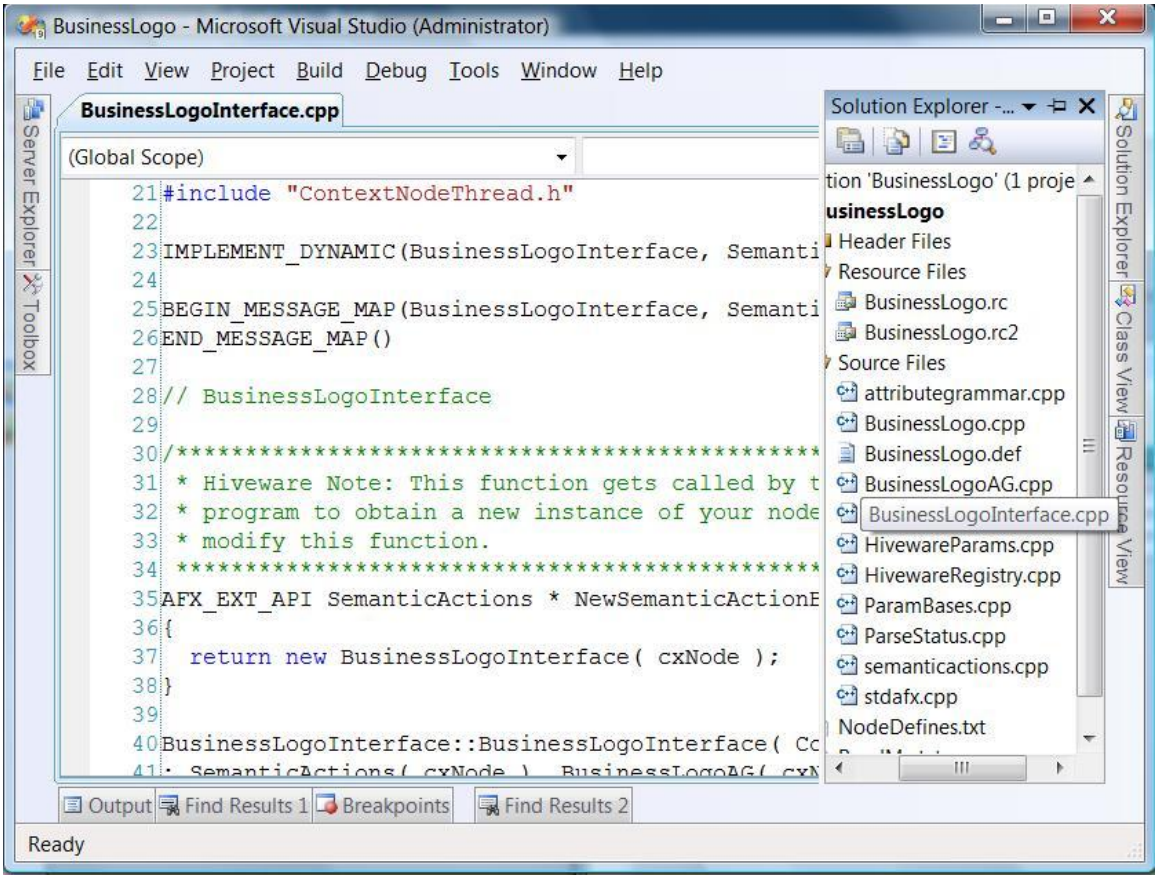


Figure 7: Yellow button gets compiler for BusinessLogo code

Changes are made to the code, compiled and debugged until the result is satisfactory to the developer whereupon the compiler is exited. Following the exit, the main program is moved back into the foreground. The other hive nodes in this hive did not quit at any point and their potential interactions with their concomitant nodes across the network were never suspended.

---

[35] fragment editor generator executable: D1: page 21, lines 1-5

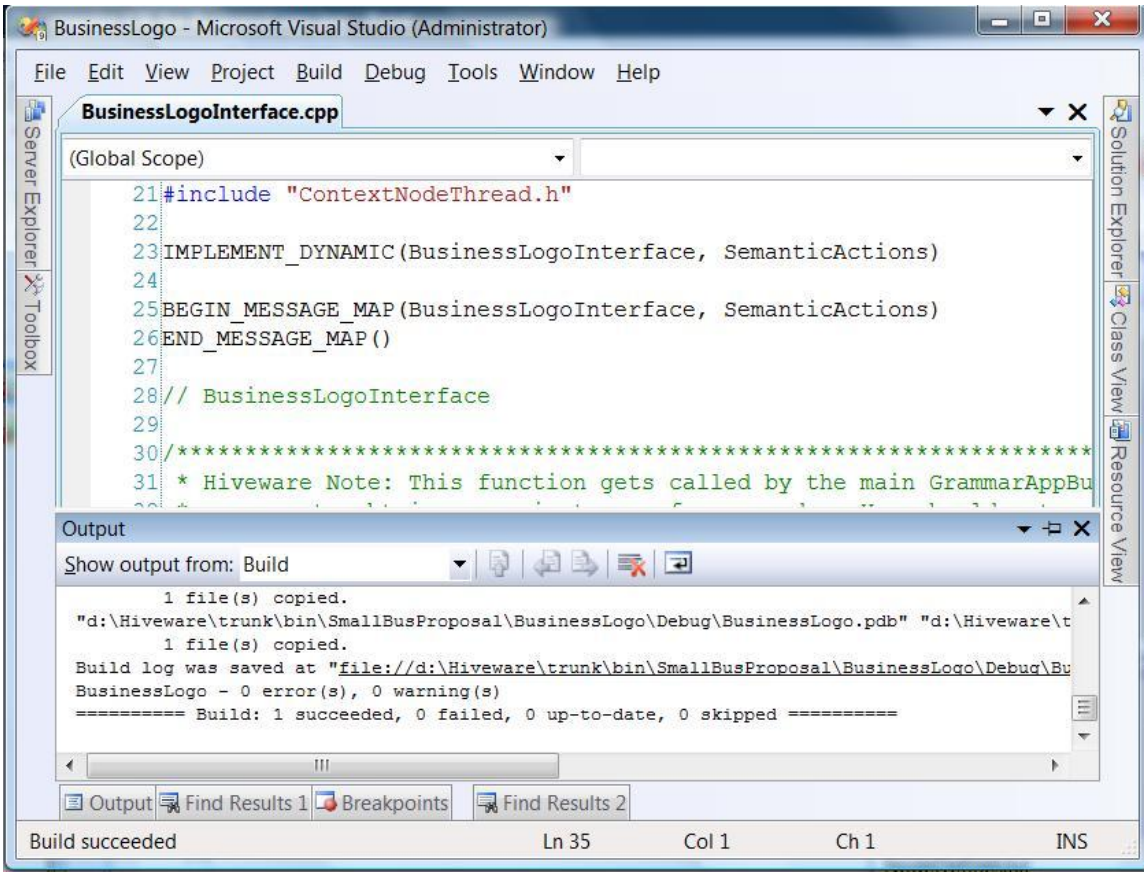


Figure 8: Successful BusinessLogo compile

The semantic action button is now blue indicating **yes DLL yes Source code**. The right-click popup dialog gives the programmer the choice either to proceed with (un)loading (i.e., hot-linking) the code into the running hive or propagate the code changes to the node's observers.

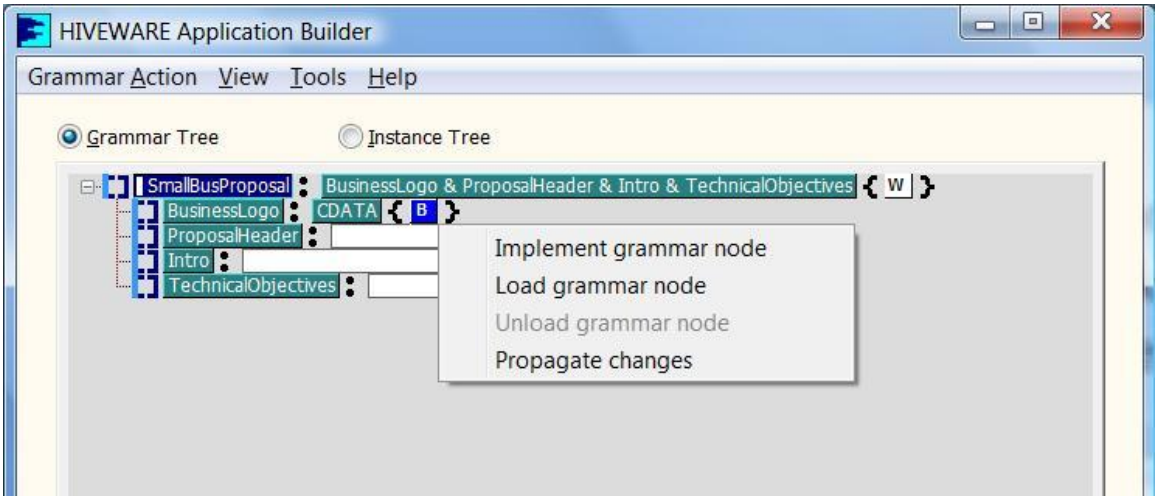


Figure 9: Ready to click on 'Load Grammar Node' to load (link) into running Hiveware executable

The developer chooses to load the code which results in the purple semantic action button indicating running code.

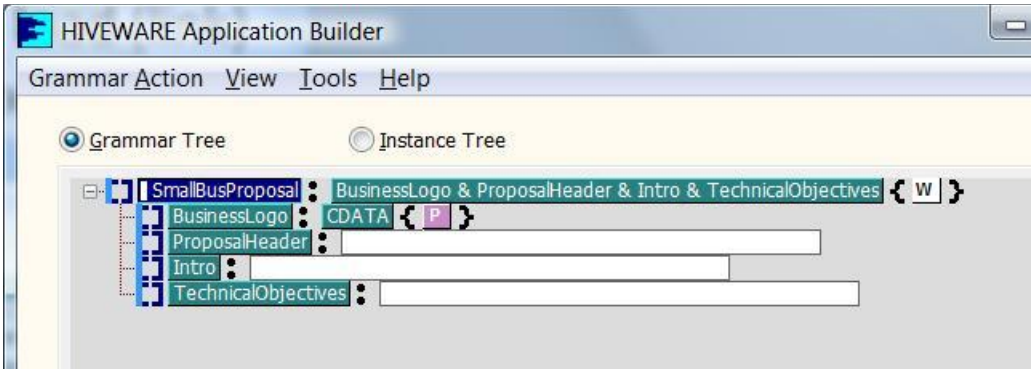


Figure 10: BusinessLogo code now running

Hiveware inherits implementations (16). The HfW embodiment came from the running DTD hive whose grammar is shown in Figure 11. Note that the HivewareForWord node consists of the CorpSiteLicenseHive and WorkGroupLicenseHive nodes and that there may be any number of them. The implementation associated with CorpSiteLicenseHive is the ability to generate any number of hives and the WorkGroupLicenseHive is associated with hive generation that comes from ancestors (see 4).

```
<!ELEMENT SdcdcWord  - - (HivewareWordAddin , HivewareForWord )>
<!ELEMENT HivewareWordAddin  - - CDATA -- development for Word Addin code -->
<!ELEMENT HivewareForWord  - -  (HivewareForWordKinds & HiveOperations) >
<!ELEMENT HivewareForWordKinds  - -  ( CorpSiteLicenseHive | WorkGroupLicenseHive | FreeTrialHive | DevelopmentHive )* >
<!ELEMENT CorpSiteLicenseHive  - -  HivewareForWord* | CDATA >
<!ELEMENT WorkGroupLicenseHive  - -  ( HivewareForWord | CDATA ) >
<!ELEMENT FreeTrialHive  - -  CDATA >
<!ELEMENT DevelopmentHive  - -  CDATA >
<!ELEMENT HiveOperations  - - CDATA >
```

Figure 11: SGML HivewareForWord ancestor to SmallBusProposal produced Word Addin

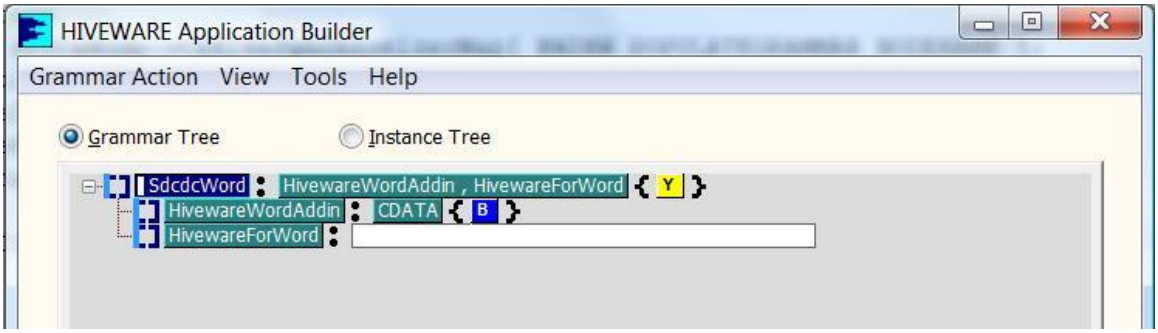


Figure 12: SdcdcWord node SGML that produced Z9 code

Figure 12 shows the DTD that produced the SdcdcWord ancestor hive node.

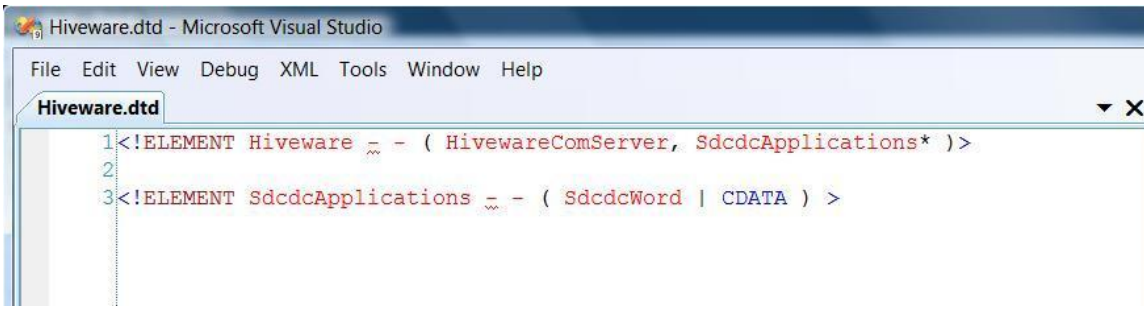


Figure 13: Hiveware DTD that produced SdcdcWord code

4. WHERE A HIVE COMES FROM AND ROOT NODE GENERATORS

COSMIC ROOT NODE GENERATOR

A hive's inception begins with the subscription process (see Table 6). Similar to natural language (2) the building of hives can have any semantic depth or height.

HFW CREATES MANY ROOT NODE GENERATORS[36]

HfW offers a particular kind of license agreement that allows the user to generate new hives at will by using the common Word practice of templates. This form of licensed HfW user makes each licensee a root context generator. The piece of HivewareForWord Addin fragment editor executable (11) in conjunction with the Hiveware engine contains this capability as well as the standard capabilities that each node acquires and or develops. Figure 1 shows the fragment editor executables (11) and communications architecture for HfW. The context and content communications activities on each node are independent of one another.

Typically, an embodiment is not its own root context generator (36), but receives the first defining DLL from an ancestor outside of its working sphere. That is, the ancestor defines the implementation and representation arena for its descendents.

[36] root context generator: D1: page 18, lines 5-9