US 20070136698A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2007/0136698 A1**

Trujillo et al. (43) **Pub. Date: Jun. 14, 2007**

(54) **METHOD, SYSTEM AND APPARATUS FOR A PARSER FOR USE IN THE PROCESSING OF STRUCTURED DOCUMENTS**

(76) Inventors: **Richard Trujillo**, Austin, TX (US); **Bryan Dobbs**, Round Rock, TX (US); **Rakesh Bhakta**, Austin, TX (US); **Howard Tsoi**, Austin, TX (US); **Jack E. Randall**, Austin, TX (US); **Howard Liu**, Plano, TX (US); **Yongjian Zhou**, Santa Clara, CA (US); **Daniel M. Cermak**, Austin, TX (US)

Correspondence Address:
BLAKELY SOKOLOFF TAYLOR & ZAFMAN
12400 WILSHIRE BOULEVARD
SEVENTH FLOOR
LOS ANGELES, CA 90025-1030 (US)

(21) Appl. No.: **11/412,698**

(22) Filed: **Apr. 27, 2006**

Related U.S. Application Data

(60) Provisional application No. 60/675,349, filed on Apr. 27, 2005. Provisional application No. 60/675,347, filed on Apr. 27, 2005. Provisional application No. 60/675,167, filed on Apr. 27, 2005. Provisional application No. 60/675,115, filed on Apr. 27, 2005.

Publication Classification

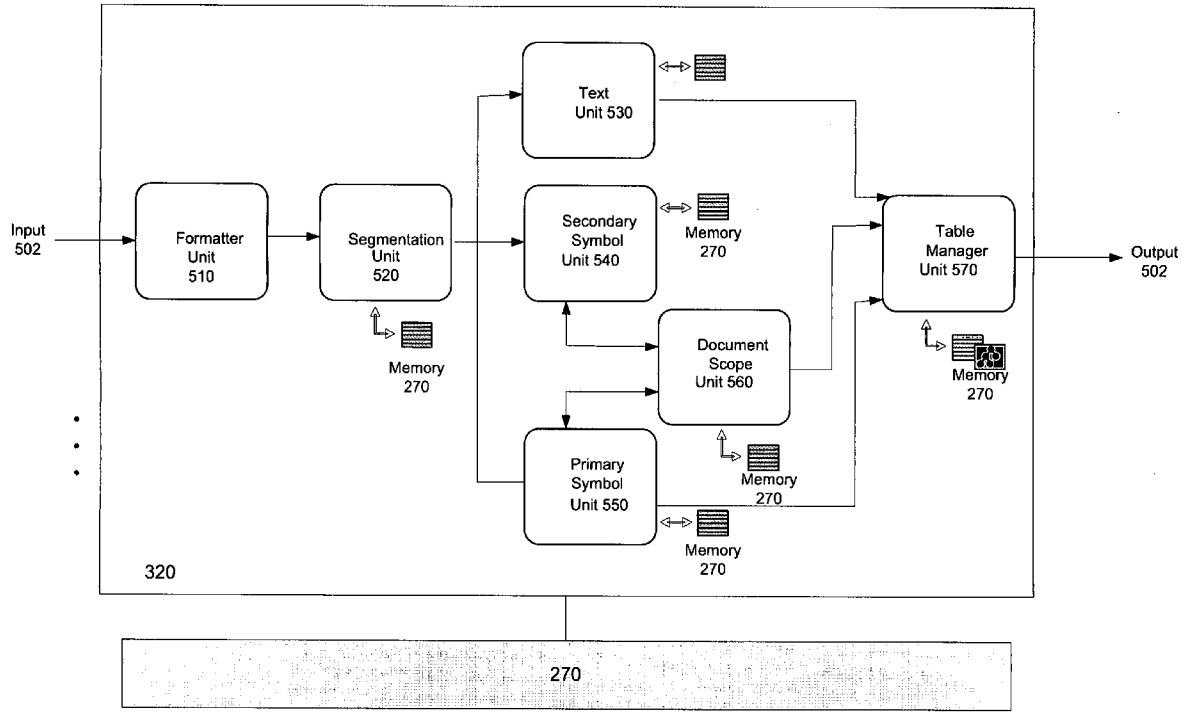
(51) **Int. Cl.**

G06F 17/50	(2006.01)
G06F 7/00	(2006.01)
G06F 17/00	(2006.01)

(52) **U.S. Cl.** **716/1**; 707/100; 707/102; 715/513

(57) **ABSTRACT**

Embodiments of systems, methods and apparatuses for a parser for generating one or more data structures representative of a structured document are disclosed. More specifically, embodiments of a parser may comprise hardware circuitry operable to receive a structured document, begin parsing the structured document as it is being received and generating the data structures representative of the structured document as it is being parsed.



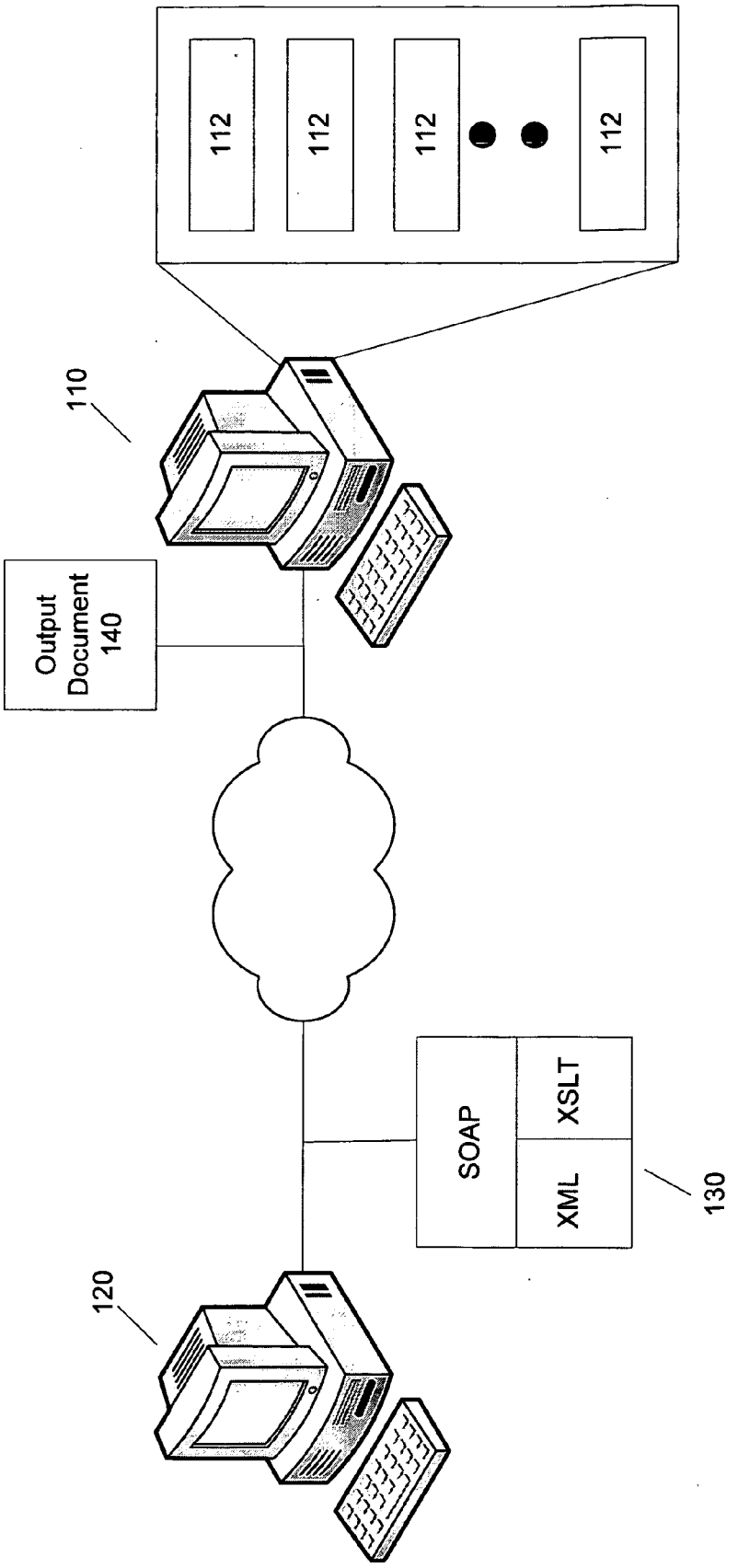


FIGURE 1

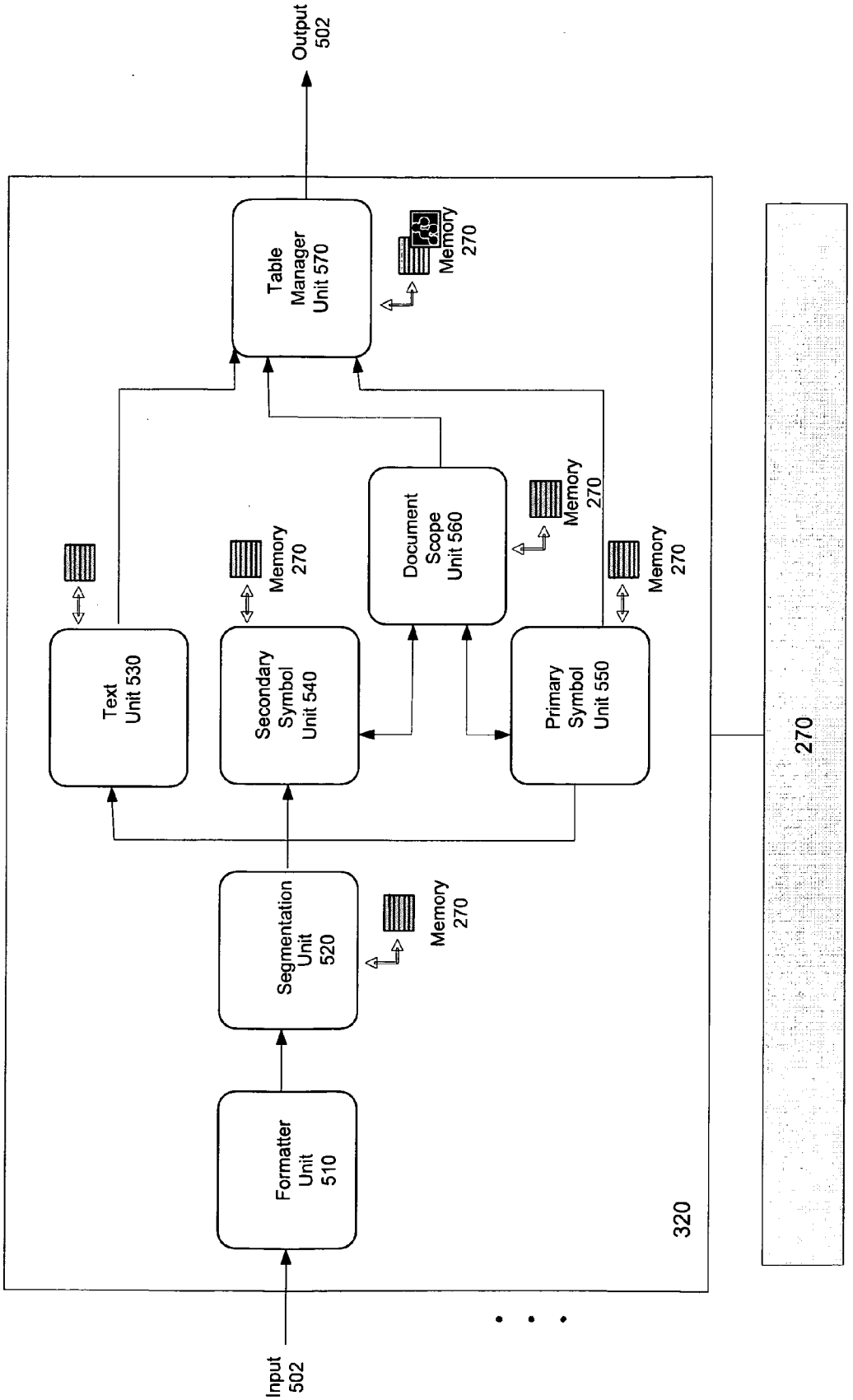

Output 502
Table Manager Unit 570
Memory 270
Document Scope Unit 560
Memory 270
Memory 270
Memory 270
Text Unit 530
Secondary Symbol Unit 540
Primary Symbol Unit 550
Segmentation Unit 520
Memory 270
Formatter Unit 510
320
270
Input 502

FIGURE 2

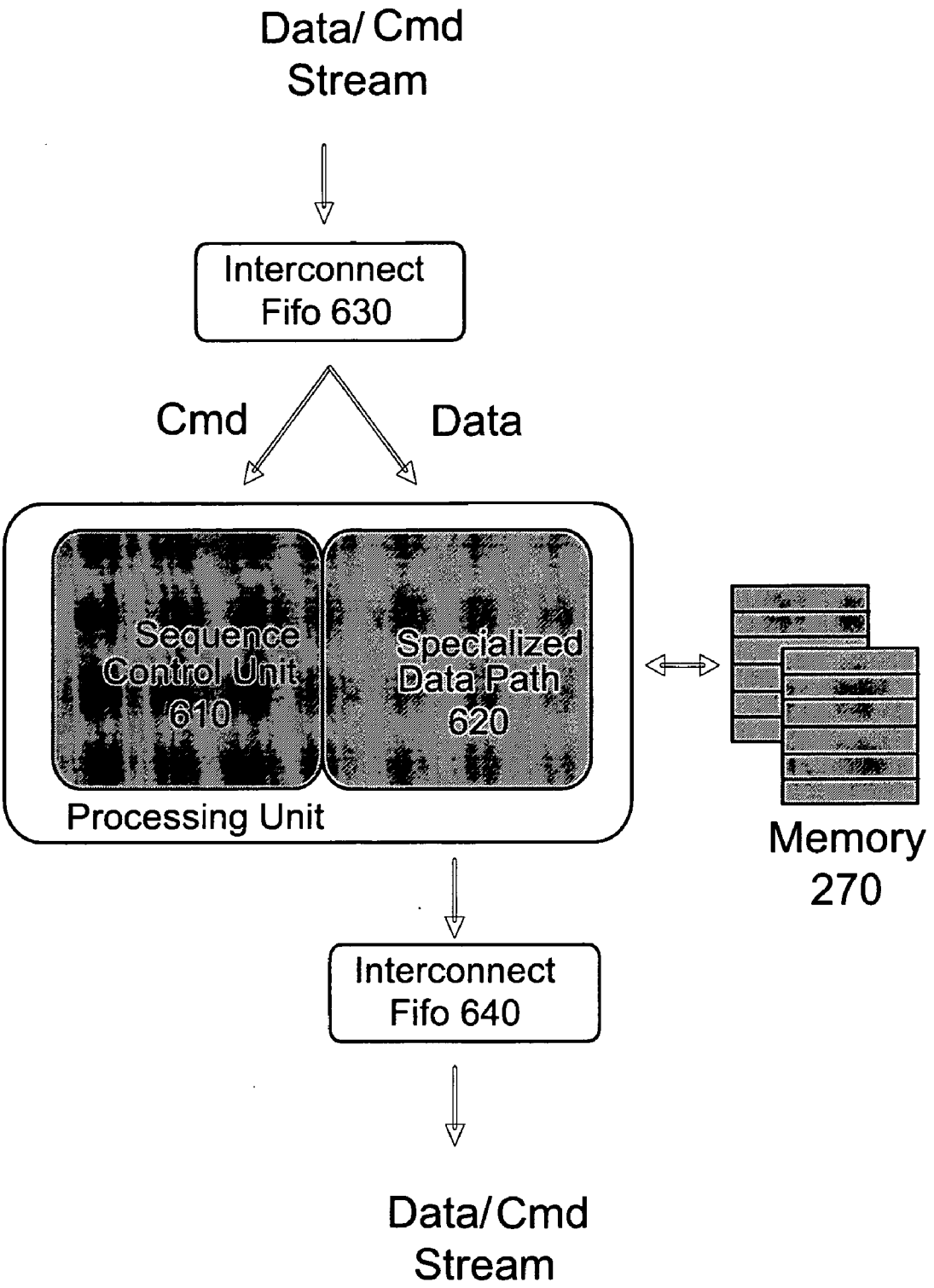

Data/ Cmd
Stream
Interconnect
Fifo 630
Cmd
Data
Sequence
Control Unit
610
Specialized
Data Path
620
Processing Unit
Memory
270
Interconnect
Fifo 640
Data/Cmd
Stream

FIGURE 3

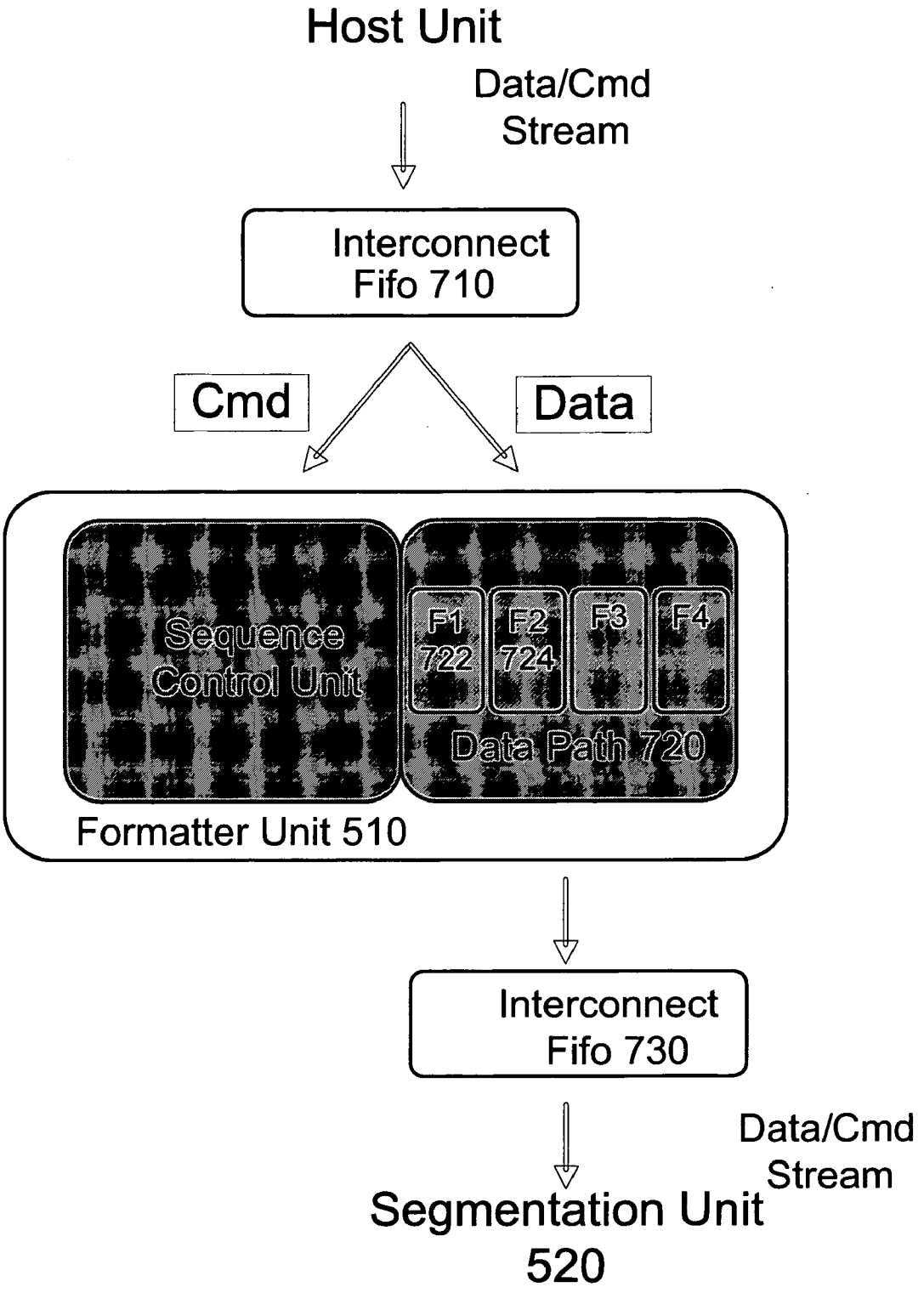

Host Unit
Data/Cmd Stream
Interconnect Fifo 710
Cmd
Data
Sequence Control Unit
F1 722
F2 724
F3
F4
Data Path 720
Formatter Unit 510
Interconnect Fifo 730
Data/Cmd Stream
Segmentation Unit 520

FIGURE 4

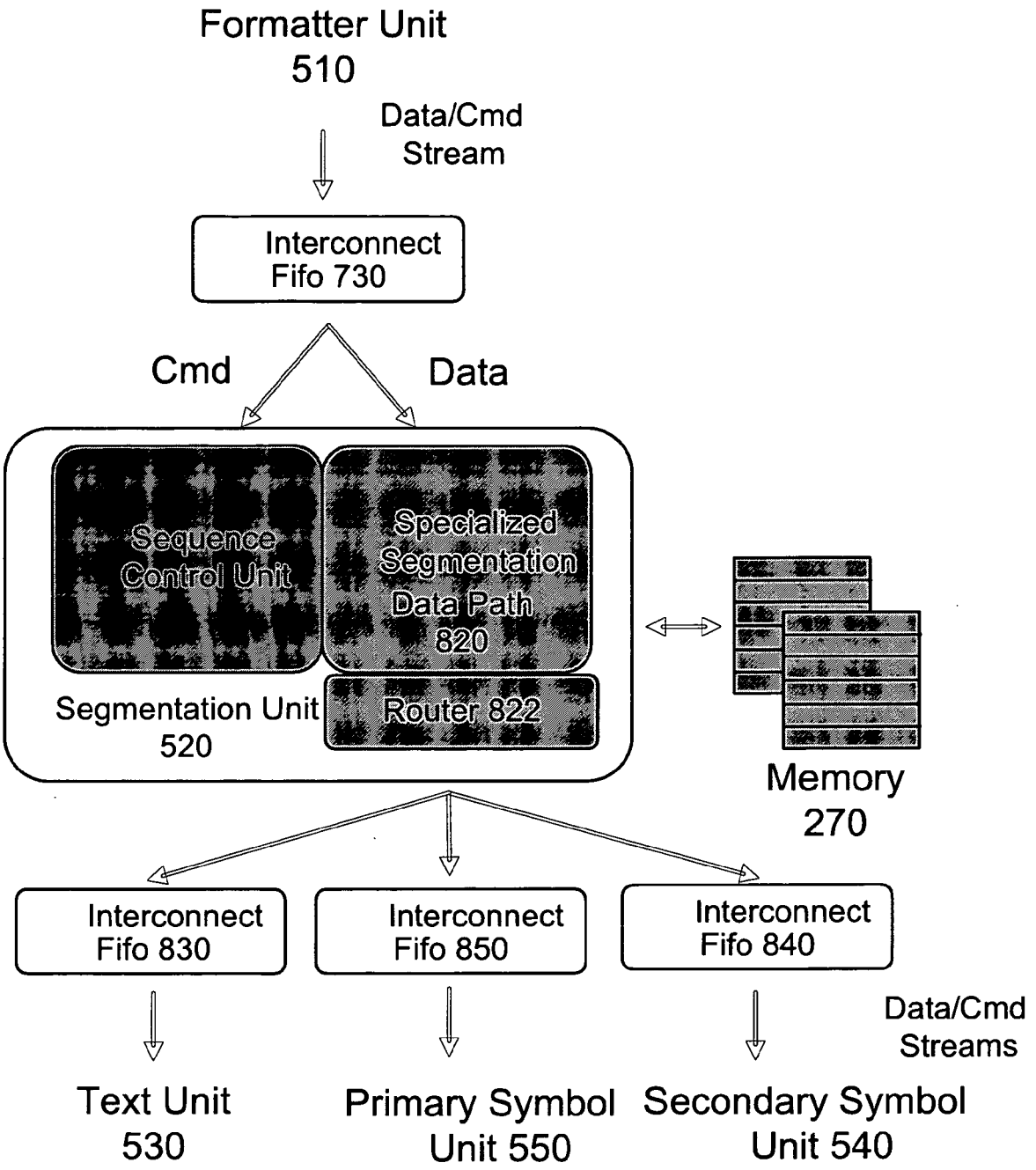

Formatter Unit
510
Data/Cmd
Stream
Interconnect
Fifo 730
Cmd
Data
Sequence
Control Unit
Specialized
Segmentation
Data Path
820
Router 822
Segmentation Unit
520
Memory
270
Interconnect
Fifo 830
Interconnect
Fifo 850
Interconnect
Fifo 840
Data/Cmd
Streams
Text Unit
530
Primary Symbol
Unit 550
Secondary Symbol
Unit 540

FIGURE 5

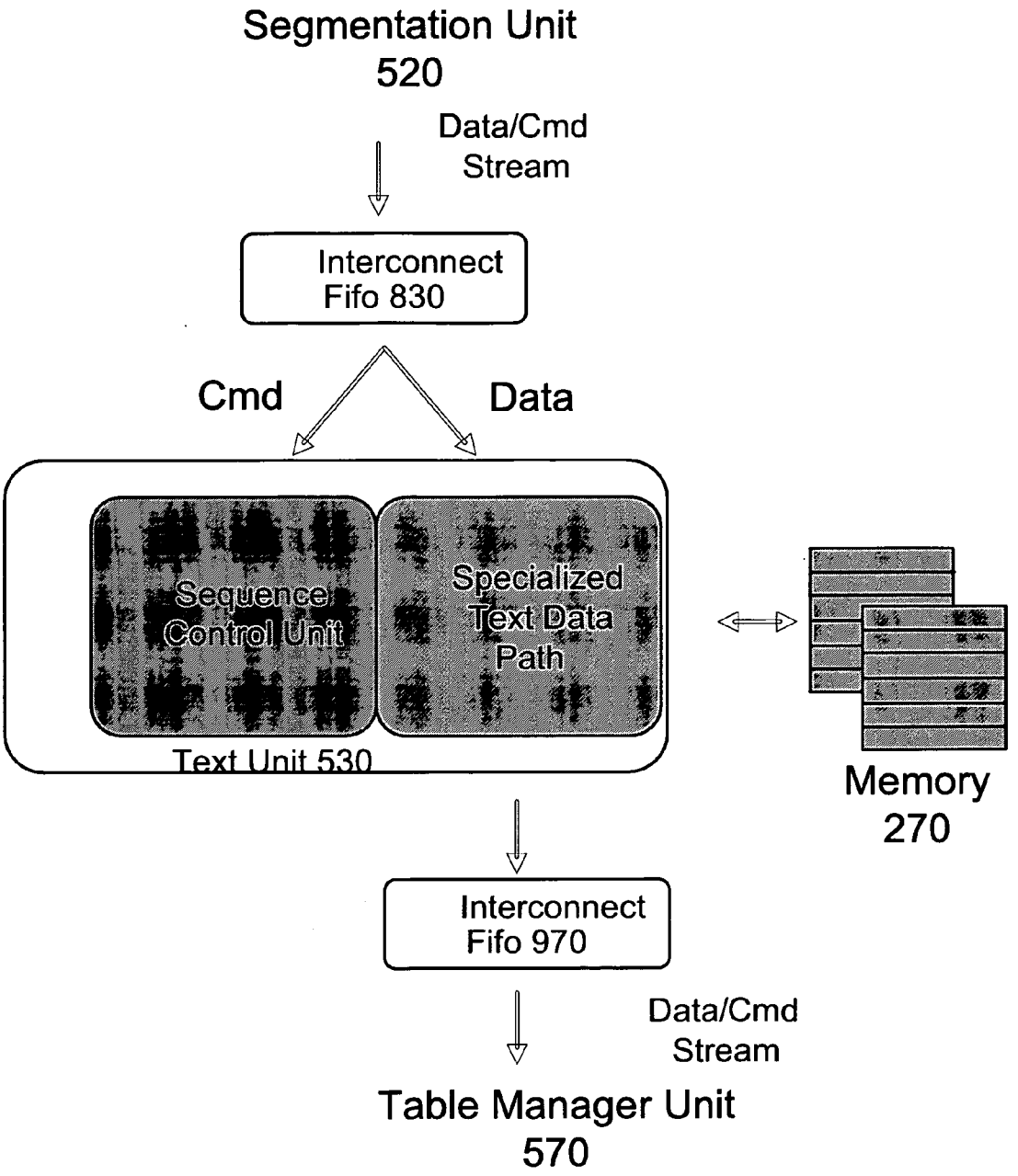

Segmentation Unit
520
Data/Cmd
Stream
Interconnect
Fifo 830
Cmd
Data
Sequence
Control Unit
Specialized
Text Data
Path
Text Unit 530
Memory
270
Interconnect
Fifo 970
Data/Cmd
Stream
Table Manager Unit
570

FIGURE 6

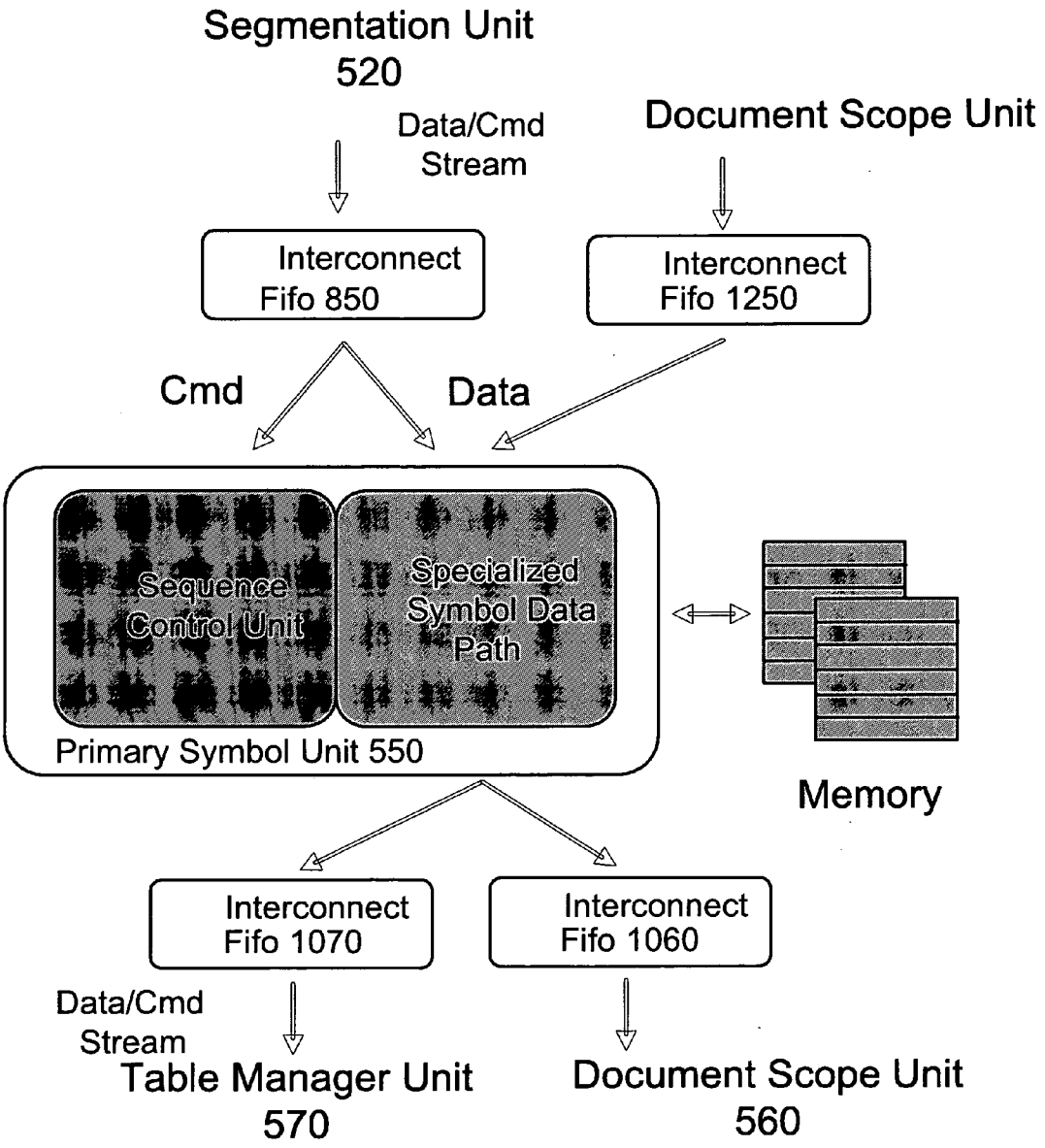

Segmentation Unit
520
Data/Cmd
Stream
Document Scope Unit
Interconnect
Fifo 850
Interconnect
Fifo 1250
Cmd
Data
Sequence
Control Unit
Specialized
Symbol Data
Path
Primary Symbol Unit 550
Memory
Interconnect
Fifo 1070
Interconnect
Fifo 1060
Data/Cmd
Stream
Table Manager Unit
570
Document Scope Unit
560

FIGURE 7

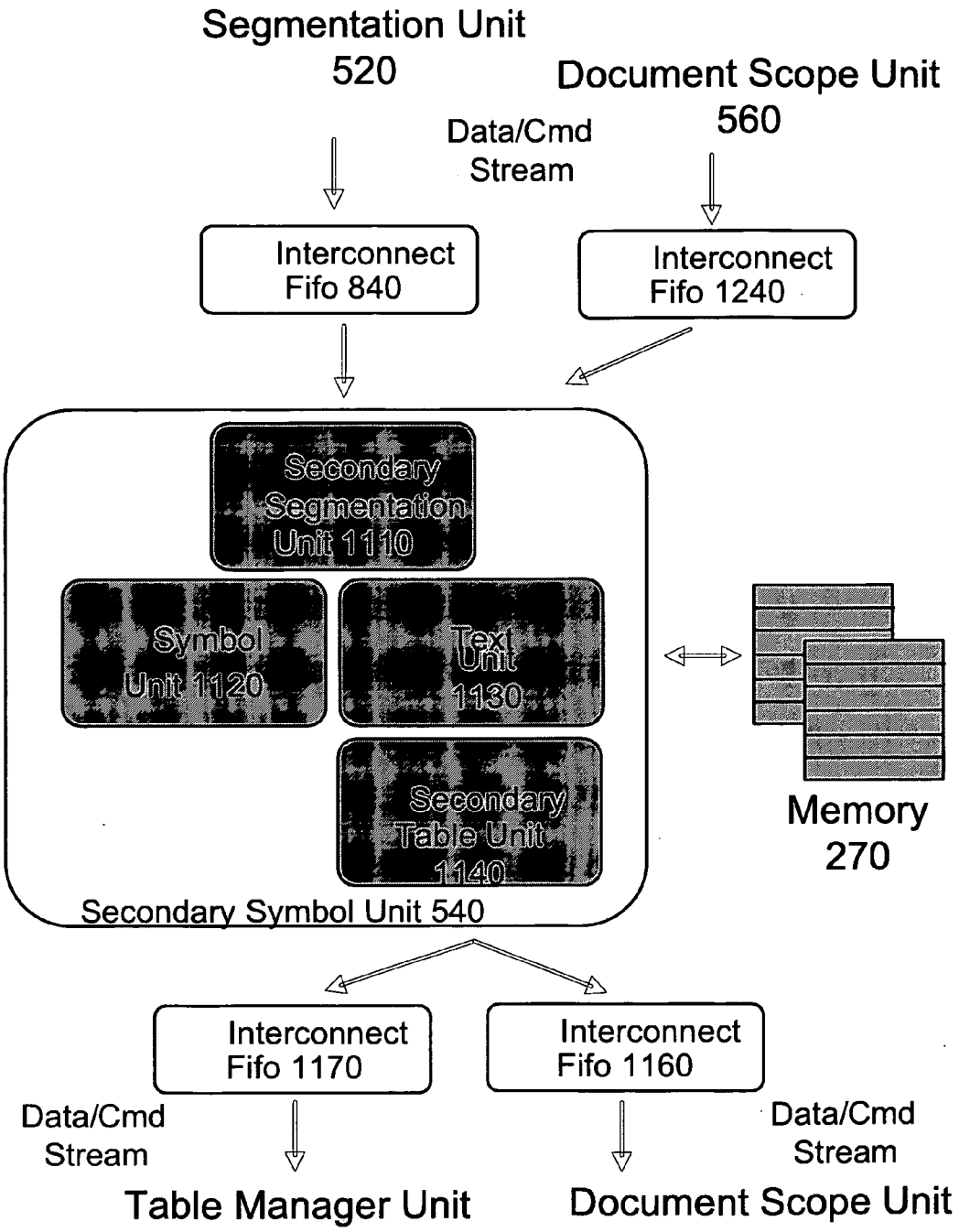

Segmentation Unit
520
Document Scope Unit
560
Data/Cmd
Stream
Interconnect
Fifo 840
Interconnect
Fifo 1240
Secondary
Segmentation
Unit 1110
Symbol
Unit 1120
Text
Unit
1130
Secondary
Table Unit
1140
Secondary Symbol Unit 540
Memory
270
Interconnect
Fifo 1170
Interconnect
Fifo 1160
Data/Cmd
Stream
Data/Cmd
Stream
Table Manager Unit
Document Scope Unit

FIGURE 8

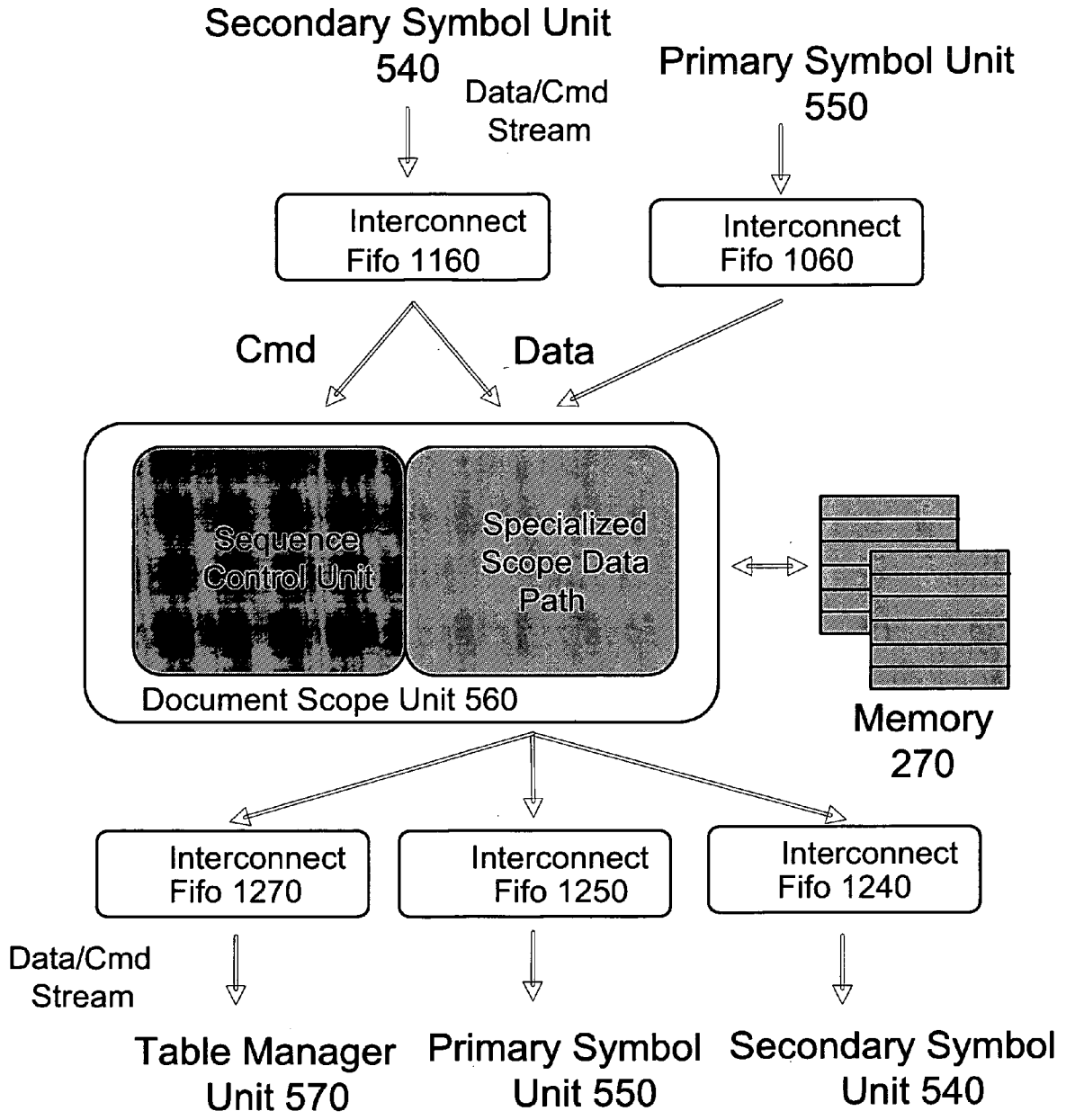

Secondary Symbol Unit 540
Data/Cmd Stream
Primary Symbol Unit 550
Interconnect Fifo 1160
Interconnect Fifo 1060
Cmd
Data
Sequence Control Unit
Specialized Scope Data Path
Document Scope Unit 560
Memory 270
Interconnect Fifo 1270
Interconnect Fifo 1250
Interconnect Fifo 1240
Data/Cmd Stream
Table Manager Unit 570
Primary Symbol Unit 550
Secondary Symbol Unit 540

FIGURE 9

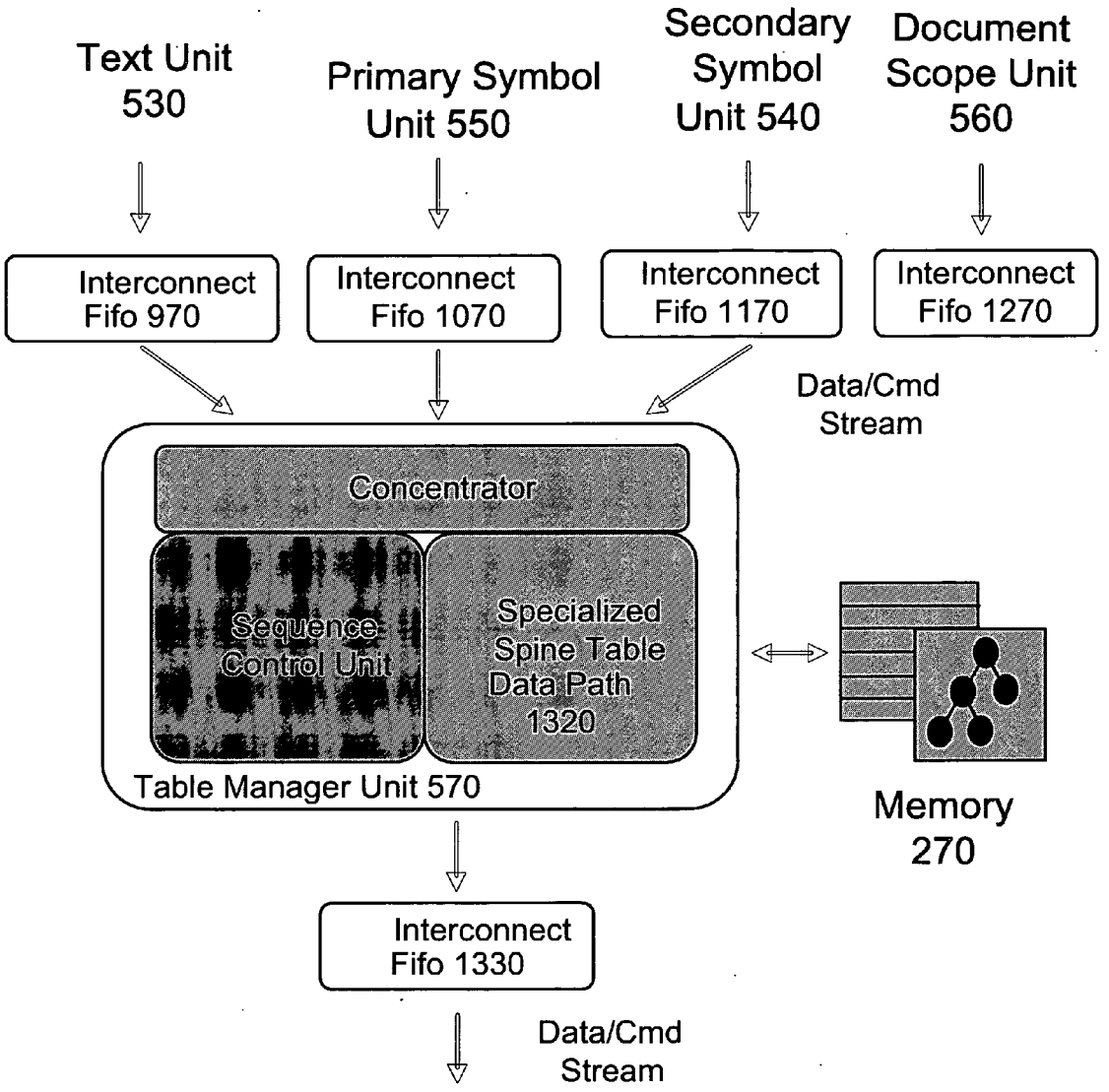

Text Unit 530
Primary Symbol Unit 550
Secondary Symbol Unit 540
Document Scope Unit 560
Interconnect Fifo 970
Interconnect Fifo 1070
Interconnect Fifo 1170
Interconnect Fifo 1270
Data/Cmd Stream
Concentrator
Sequence Control Unit
Specialized Spine Table Data Path 1320
Table Manager Unit 570
Memory 270
Interconnect Fifo 1330
Data/Cmd Stream
Parse-time Expression Processor 330

FIGURE 10

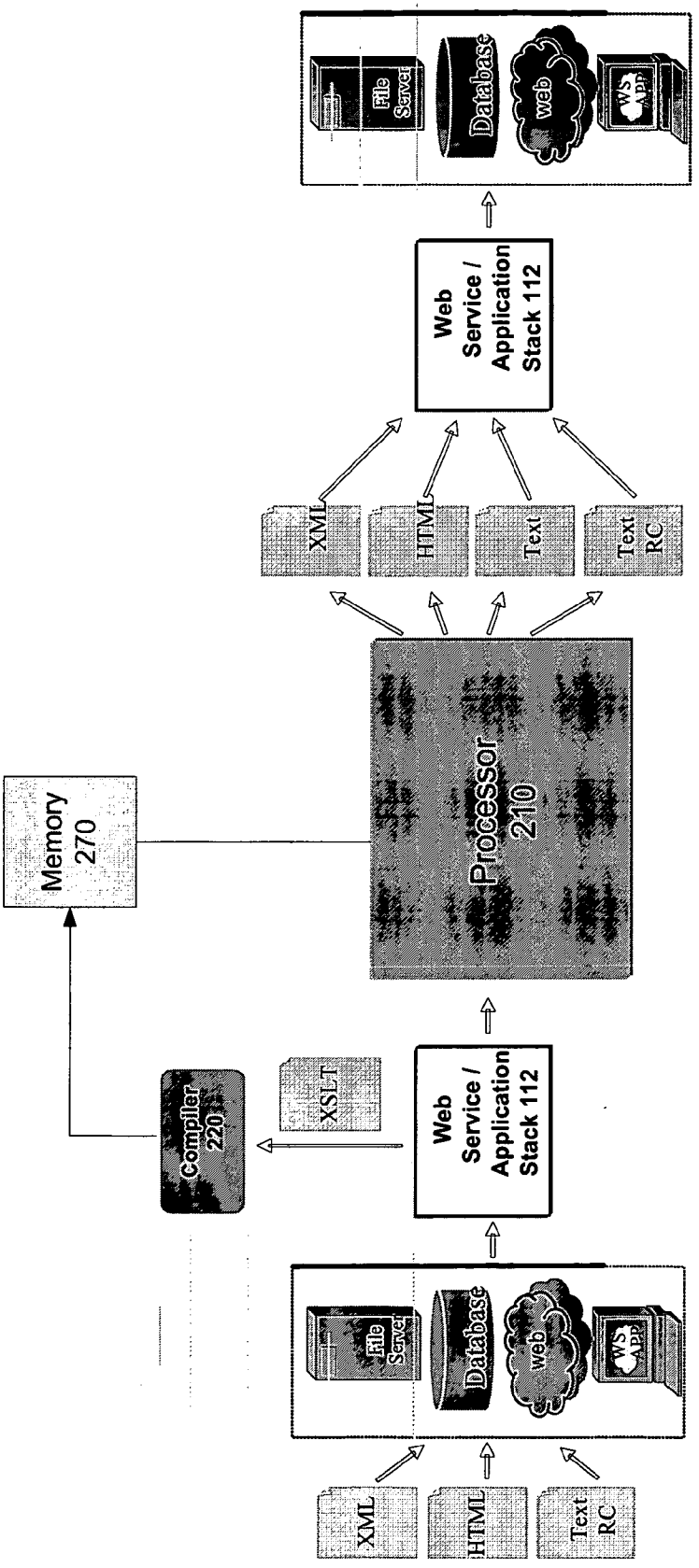

File Server
Database
web
WS APP
Web Service / Application Stack 112
XML
HTML
Text
Text RC
Processor 210
Memory 270
Compiler 220
XSLT
Web Service / Application Stack 112
File Server
Database
web
WS APP
XML
HTML
Text RC

FIGURE 11

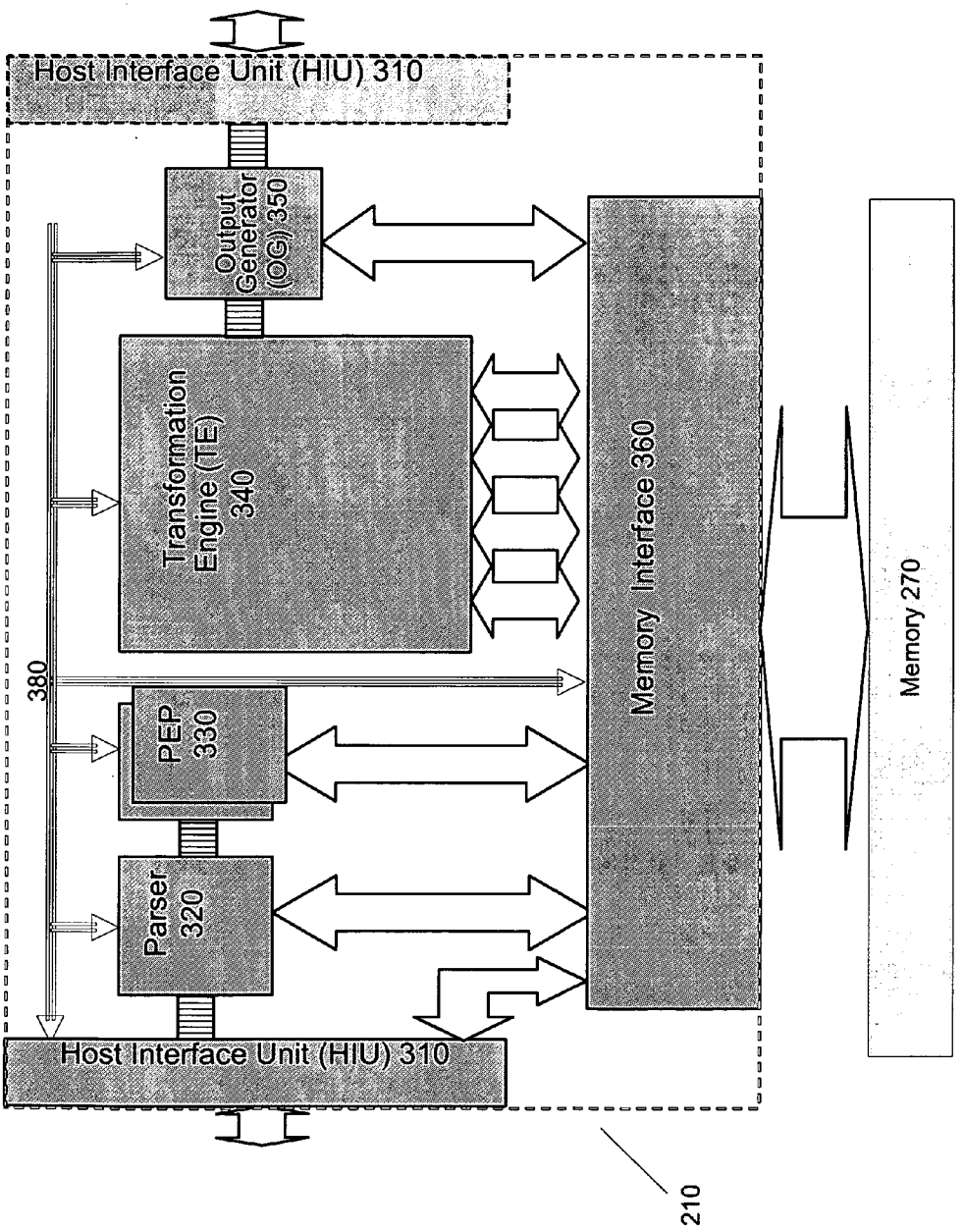

Host Interface Unit (HIU) 310
Output Generator (OG) 350
Transformation Engine (TE) 340
380
PEP 330
Parser 320
Host Interface Unit (HIU) 310
Memory Interface 360
Memory 270
210

FIGURE 12

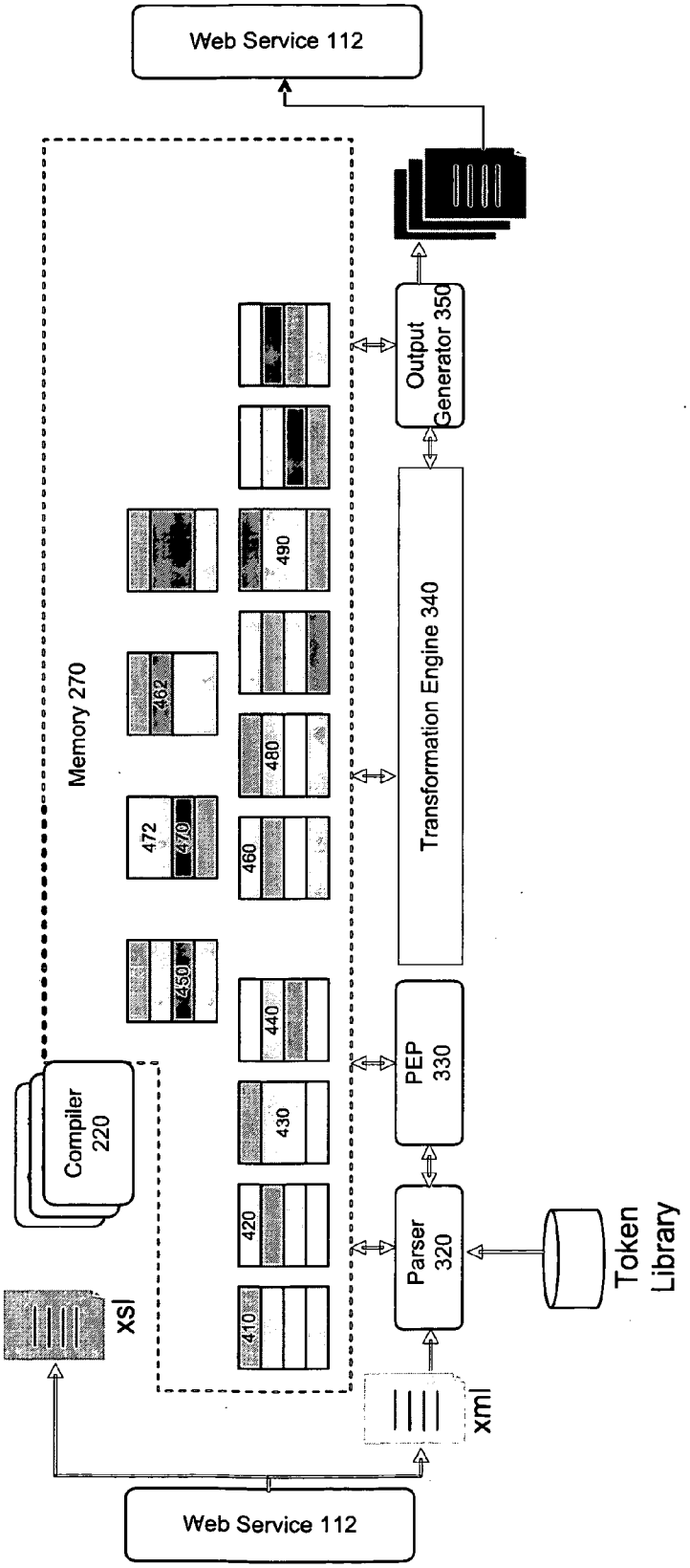

Web Service 112
Output Generator 350
Transformation Engine 340
PEP 330
Parser 320
Token Library
Compiler 220
Memory 270
xsl
xml
Web Service 112
410
420
430
440
450
460
470
472
480
462
490

FIGURE 13

METHOD, SYSTEM AND APPARATUS FOR A PARSER FOR USE IN THE PROCESSING OF STRUCTURED DOCUMENTS

RELATED APPLICATIONS

[0001] This application claims a benefit of priority under 35 U.S.C. § 119(e) to U.S. Provisional Patent Application Nos. 60/675,349, by inventors Howard Tsoi, Daniel Cermak, Richard Trujillo, Trenton Grale, Robert Corley, Bryan Dobbs and Russell Davoli, entitled "Output Generator for Use with System for Creation of Multiple, Hierarchical Documents", filed on Apr. 27, 2005; 60/675,347, by inventors Daniel Cermak, Howard Tsoi, John Derrick, Richard Trujillo, Udi Kalekin, Bryan Dobbs, Ying Tong, Brendon Cahoon and Jack Matheson, entitled "Transformation Engine for Use with System for Creation of Multiple, Hierarchical Documents", filed on Apr. 27, 2005; 60/675,167, by inventors Richard Trujillo, Bryan Dobbs, Rakesh Bhakta, Howard Tsoi, Jack Randall, Howard Liu, Yongjian Zhou and Daniel Cermak, entitled "Parser for Use with System for Creation of Multiple, Hierarchical Documents", filed on Apr. 27, 2005 and 60/675,115, by inventors John Derrick, Richard Trujillo, Daniel Cermak, Bryan Dobbs, Howard Liu, Rakesh Bhakta, Udi Kalekin, Russell Davoli, Clifford Hall and Avinash Palaniswamy, entitled "General Architecture for a System for Creation of Multiple, Hierarchical Documents", filed on Apr. 27, 2005 the entire contents of which are hereby expressly incorporated by reference for all purposes.

TECHNICAL FIELD OF THE INVENTION

[0002] The invention relates in general to methods and systems for processing structured documents, and more particularly, to the design and implementation of efficient parsers for use in the processing, transformation or rendering of structured documents.

BACKGROUND OF THE INVENTION

[0003] Electronic data, entertainment and communications technologies are growing increasingly prevalent with each passing day. In the past, the vast majority of these electronic documents were in a proprietary format. In other words, a particular electronic document could only be processed or understood by the application that created that document. Up until relatively recently this has not been especially troublesome.

[0004] This situation became progressively more problematic with the advent of networking technologies, however. These networking technologies allowed electronic documents to be communicated between different and varying devices, and as these network technologies blossomed, so did user's desires to use these networked devices to share electronic data.

[0005] Much to the annoyance of many users, however, the proprietary formats of the majority of these electronic documents prevented them from being shared between different platforms: if a document was created by one type of platform it usually could not be processed, or rendered, by another type of platform.

[0006] To that end, data began to be placed in structured documents. Structured documents may be loosely defined as any type of document that adheres to a set of rules. Because the structured document conforms to a set of rules it enables the cross-platform distribution of data, as an application or platform may process or render a structured document based on the set of rules, no matter the application that originally created the structured document.

[0007] The use of structured documents to facilitate the cross-platform distribution of data is not without its own set of problems, however. In particular, in many cases the structured document does not itself define how the data it contains is to be rendered, for example for presentation to a user. Exacerbating the problem is the size of many of these structured documents. To facilitate the organization of data intended for generic consumption these structured documents may contain a great deal of meta-data, and thus may be larger than similar proprietary documents, in some cases up to twenty times larger or more.

[0008] In many cases, instructions may be provided for how to transform or render a particular structured document. For example, one mechanism implemented as a means to facilitate processing XML is the extensible stylesheet language (XSL) and stylesheets written using XSL. Stylesheets may be written to transform XML documents from one markup definition (or "vocabulary") defined within XML to another vocabulary, from XML markup to another structured or unstructured document form (such as plain text, word processor, spreadsheet, database, pdf, HTML, etc.), or from another structured or unstructured document form to XML markup. Thus, stylesheets may be used to transform a document's structure from its original form to a form expected by a given user (output form).

[0009] Typically, structured documents are transformed or rendered with one or more software applications. However, as many definitions for these structured languages were designed and implemented without taking into account conciseness or efficiency of parsing and transformation, the use of software applications to transform or render these structured documents may be prohibitively inefficient.

[0010] Thus, as can be seen, there is a need for methods and systems for an architecture for the efficient processing of structured documents.

BRIEF DESCRIPTION OF THE DRAWINGS

[0011] The drawings accompanying and forming part of this specification are included to depict certain aspects of embodiments of the invention. A clearer impression of embodiments of the invention, and of the components and operation of systems provided with embodiments of the invention, will become more readily apparent by referring to the exemplary, and therefore nonlimiting, embodiments illustrated in the drawings, wherein identical reference numerals designate the same components. Note that the features illustrated in the drawings are not necessarily drawn to scale.

[0012] FIG. **1** depicts an embodiment of an architecture for the implementation of web services.

[0013] FIG. **2** depicts an embodiment of an architecture for a parser.

[0014] FIG. **3** depicts an embodiment of an architecture for logical units of a parser.

[0015] FIG. **4** depicts one embodiment of a formatter unit.

[0016] FIG. **5** depicts one embodiment of a segmentation unit.

[0017] FIG. **6** depicts one embodiment of a text unit.

[0018] FIG. **7** depicts one embodiment of a primary symbol unit.

[0019] FIG. **8** depicts one embodiment of a secondary symbol unit.

[0020] FIG. **9** depicts one embodiment of a document scope unit; and

[0021] FIG. **10** depicts one embodiment of a table manager unit.

[0022] FIG. **11** depicts the processing of a structured document.

[0023] FIG. **12** depicts one embodiment of an architecture for a device for the processing of structured documents.

[0024] FIG. **13** depicts one embodiment of an architecture for the processing of structured documents utilizing the embodiment of the device depicted in FIG. **12**.

DETAILED DESCRIPTION

[0025] Embodiments of the invention and the various features and advantageous details thereof are explained more fully with reference to the nonlimiting embodiments that are illustrated in the accompanying drawings and detailed in the following description. Descriptions of well known starting materials, processing techniques, components and equipment are omitted so as not to unnecessarily obscure the invention in detail. Skilled artisans should understand, however, that the detailed description and the specific examples, while disclosing preferred embodiments of the invention, are given by way of illustration only and not by way of limitation. Various substitutions, modifications, additions or rearrangements within the scope of the underlying inventive concept(s) will become apparent to those skilled in the art after reading this disclosure.

[0026] Reference is now made in detail to the exemplary embodiments of the invention, examples of which are illustrated in the accompanying drawings. Wherever possible, the same reference numbers will be used throughout the drawings to refer to the same or like parts (elements).

[0027] Before describing embodiments of the present invention it may be useful to describe an exemplary architecture for a web service. Although web services are known in the art, a description of such an architecture may be helpful in better explaining the embodiments of the invention depicted herein.

[0028] FIG. **1** depicts an embodiment of one such architecture for implementing a web service. Typically, web services provide a standard means of interoperating between different software applications running on a variety of platforms and/or frameworks. A web service provider **110** may provide a set of web services **112**. Each web service **112** may have a described interface, such that a requestor may interact with the web service **112** according to that interface.

[0029] For example, a user at a remote machine **120** may wish to use a web service **112** provided by web service provider **110**. To that end the user may use a requestor agent to communicate message **130** to a service agent associated with the desired web service **112**, where the message is in a format prescribe by the definition of the interface of the desired web service **112**. In many cases, the definition of the interface describes the message formats, data types, transport protocols, etc. that are to be used between a requester agent and a provider agent.

[0030] The message **130** may comprise data to be operated on by the requested web service **112**. More particularly, message **130** may comprise a structured document and instructions for transforming the structured document. For example, message **130** may be a SOAP (e.g. Simple Object Access Protocol) message comprising an eXtensible Markup Language (XML) document and an eXstensible Style Sheet Language Transformation (XSLT) stylesheet associated with the XML document. It should be noted that, in some cases, transformation instructions (e.g. a Document Type Definition (DTD), schema, or stylesheet) may be embedded in a structured document, for example, either directly or as a pointer. In such cases the transformation instructions may be extracted from the document before being utilized in any subsequent method or process.

[0031] Thus, in some cases the provider agent associated with a particular web service **112** may receive message **130**; web service **112** may process the structured document of message **130** according to the instructions for transforming the structured document included in message **130**; and the result **140** of the transformation returned to the requester agent.

[0032] In some cases, many structured documents may be sent to a particular web service **112** with one set of transformation instructions, so that each of these documents may be transformed according to the identical set of instructions. Conversely, one structured document may be sent to a particular web service **112** with multiple sets of transformation instructions to be applied to the structured document.

[0033] Hence, as can be seen from this brief overview of the architecture for implementing web services **112**, it may be highly desired to process these structured documents as efficiently as possible such that web services **112** may be used on many data sets and large data sets without creating a bottleneck during the processing of the structured documents and processing resources of web service provider **110** may be effectively utilized.

[0034] More particularly, in order to process or transform these structured documents it may be necessary in some cases to parse these structured documents (e.g. both lexically and semantically) so that operations may performed on identified semantic elements of the structured document, a representation of the structured document created, the content of the structured document provided in an alternate format, etc. Moreover, as the parsing of these structured documents may be a possible bottleneck in the processing or transformation of these structured documents it is desirable to parse these structured documents in an efficient manner.

[0035] Attention is now directed to embodiments of systems and methods for an architecture for the efficient parsing of structured documents. Embodiments of the present invention may provide a parser which comprises hardware circuitry, for example a hardware processing device such as an

Application Specific Integrated Circuit (ASIC), for efficient parsing of a structured document. In other words, embodiments the present invention may provide the capability to parse a structured document substantially solely in hardware (e.g. substantially without the use of software). This hardware may generate data structures storing the parsed content, some of which may reflect the structure of the document. Additionally, the parser may generate output events comprising content of the structured document, or references to this content, such that these events may be utilized by other applications or hardware, for example to form a document object model (DOM) of the parsed content, or to match content of the document with certain expressions, etc.

[0036] More specifically, embodiments of the present invention may provide a parser circuit operable to receive a document in a streaming fashion. As the structured document is received, data from the document may be segmented into semantically meaningful groups, these groups classified as one or more types, and routed to logical units of the parser circuit based on their type. Each of these logical units may, in turn, be operable to process the type of data routed to it. By processing differing types of data from a structured document using distinct logical processing units, where each of these logical processing units may operate substantially in tandem with one another, parsing of a structured document may be accomplished in a quick and efficient manner. More specifically, the parsing of the structured document and the creation of one or more data structures representative of the structured document may be accomplished substantially simultaneously (it will be understood that for purposes of this disclosure that the occurrence of two events substantially simultaneously indicates that each of the two events may at least partially occur before the completion of the other event).

[0037] Additionally, by providing events, or otherwise identifying data of the structured document to other applications or hardware, while a structured document is in the process of being parsed the overall efficiency of applications or hardware which may utilize embodiments of this parser may operate more quickly and efficiently as well,

[0038] One particular embodiment of parser **320** is depicted in FIG. **2**. Parser **320** can receive one or more commands to operate on data, followed by the data (e.g. corresponding to a structured document) at input **502**. Document data identified by parser **320** may be communicated to applications or other hardware components through output **504**. Parser **320** may comprise one or more interfaces with memory **270** such that data structures may be filled, accessed, or built in memory **270** through this interface.

[0039] Parser **320** may comprise formatter unit **510**, segmentation unit **520**, text unit **530**, secondary symbol unit **540**, primary symbol unit **550**, document scope unit **560**, and table manager unit **570**. Formatter unit **510** may be coupled to input **502** and segmentation unit **520** which, in turn may be coupled to text unit **530**, secondary symbol unit **540** and primary symbol unit **550**. Secondary symbol unit **540** and primary symbol unit **550** are both coupled to document scope unit **560**, while all three of secondary symbol unit **540**, primary symbol unit **550** and document scope unit **560** are coupled to table manager unit **570**. Additionally, logical units **510**, **520**, **530**, **540**, **550**, **560**, **570** may interface with memory **270**.

[0040] Communication between the logical processing units **510**, **520**, **530**, **540**, **550**, **560**, **570** may take place through one or more First In First Out (FIFO) queues, such that when a logical processing unit **510**, **520**, **530**, **540**, **550**, **560**, **570** wishes to send a communication to another logical processing unit **510**, **520**, **530**, **540**, **550**, **560**, **570** this communication may be placed in an output FIFO queue of the sender which also serves as an input FIFO queue for the intended recipient of the communication. In one embodiment, these FIFO queues may be implemented in hardware (e.g. on an ASIC comprising logical units **510**, **520**, **530**, **540**, **550**, **560**, **570**) while in other embodiments these FIFO queues may be implemented in memory **270**.

[0041] Formatter unit **510** may receive a command indicating that data to follow is to be processed, and the format of the data that is to follow, followed by a stream of the data itself. For example, one or more source documents may be presented to formatter unit **510** in a serial fashion, as opposed to a source document needing to be wholly present before processing may begin.

[0042] Formatter unit **510** transcodes received data into an internal format (e.g. a data format utilized by other logical processing units **520**, **530**, **540**, **550**, **560**, **570**) and forwards the data (in the internal format) to segmentation unit **520**. Thus, data may arrive at formatter unit **510** in a variety of formats such as Unicode Transformation Format (UTF), Big Endian, Little Endian, International Standards Organization (ISO) **8859**, Windows (Win) **1252**, etc. and be converted to an internal format such that each of the other logical units **520**, **530**, **540**, **550**, **560**, **570** can process the data according to the internal format. Formatter unit **510** may then communicate commands and data to segmentation unit **520**.

[0043] Segmentation unit **520** may partition incoming data from the formatter unit **510** into frames (logical groupings of data) based on the markup of the structured document(s), identify the type of the data frame and route data (e.g. data frames) to a logical processing unit **530**, **540**, **550** operable to process that type of data. The logical processing units **530**, **540**, **550** operable to handle these differing data types are text unit **530**, primary symbol unit **540** and secondary symbol unit **560**.

[0044] Text unit **530** processes data frames which have been identified as text strings to store these text strings to memory **270** and may mark or associate these text strings with other associated properties, for example, if the text string is a comment the stored text string may be associated with a comment tag, if the string is whitespace it may be associated with a whitespace tag, etc. Furthermore, text unit **530** may condition these text strings by adding or removing portions of the text string, for example comment notation may be removed before storing a text string, etc.

[0045] Primary symbol unit **550** processes data identified as a primary symbol by matching the symbol to reference data in memory or storing the symbol in memory and associating the symbol with a unique reference number.

[0046] Secondary symbol unit **560** processes data identified as secondary symbols or secondary data, for example sub-elements or attributes of elements, that may require further processing by segmenting these secondary symbols according to their type and processing these secondary symbols according to their type. Secondary symbol unit **560**

may also identify directives embedded in these received secondary symbols which may effect subsequent processing of a structured document.

[0047] While performing their respective processing, each of primary symbol unit **550** and secondary symbol unit **560** may interact with document scope unit **560** to track, reference or determine the order (e.g. scope) of data frames with respect to a source document. Document scope unit **560** is operable to maintain, construct and reference data structures in memory **270** pertaining to the scope of a source document and may provide scope information pertaining to symbols to primary symbol unit **550**, secondary symbol unit **560** or table manger unit **570**.

[0048] Table manager unit **570** assembles one or more data structures in memory **270** representative of a source document based on communications from text unit **530**, primary symbol unit **550**, secondary symbol unit **540** or document scope unit **560**. The data structure created by table manager unit **570** may represent the structure of a source document and reference one or more of the data structures created by text unit **530**, primary symbol unit **550**, secondary symbol unit **540** or document scope unit **560**.

[0049] In one embodiment, each of logical processing units **510**, **520**, **530**, **540**, **550**, **560**, **570** may be based on a similar architecture. A block diagram of one embodiment of a general architectural format for the implementation of one or more of logical units **510**, **520**, **530**, **540**, **550**, **560**, **570** is presented in FIG. **3**. Logical processing unit **600** may be operable to perform a specialized task and, as such may be operable to execute a set of opcodes associated with that specialized task. Logical processing unit **600** may have one or more input interconnect FIFO queues **630** a sequence control unit **610**, a data path **620** and one or more output FIFO queues **640** (which may also serve as input queues for another logical processing unit). Input FIFO queue **630**, which may be implemented in hardware or through shared memory, may serve to couple logical processing unit **600** to another logical processing unit (e.g. input FIFO queue **630** may be the output FIFO queue of another logical processing unit). Thus, commands and associated data may be received at logical processing unit **600**, data processed according to the commands, and commands and data placed in one or more output FIFO queues **640**. In other words, an incoming command/data stream may be conditioned for processing by downstream logical processing units, for example by removing commands from the stream, inserting commands into the command/data stream, changing arguments to commands, augmenting, stripping or replacing data with other data, etc.

[0050] During operation, a communication for logical processing unit **600** may be placed in input FIFO **630**. This communication may comprise one or more commands (e.g. opcodes) and data associated with those commands. Logical unit **600** may receive and process the commands and data in each of input FIFO queues **630** using data path **620** which may comprise hardware circuitry operable to accomplish at least portions of the specialized task for which logical processing unit **600** is operable. Furthermore, to facilitate the processing of the data, logical processing unit **600** may have an interface to memory **270** such that logical processing unit **600** may create data structures in memory **270**, or reference existing data structures in memory **270** (e.g. to read data from, or write data to these data structures).

[0051] To speed the processing of data by logical processing unit **600** sequence control unit **610** may be operable to order the execution of commands in input FIFOs queues **630** (and thus the processing of data associated with these commands) to take advantage of the operation of the architecture of the particular data path **620** of the logical processing unit **600**.

[0052] Turning to FIGS. **4-10** various embodiments of formatter unit **510**, segmentation unit **520**, text unit **530**, secondary symbol unit **540**, primary symbol unit **550**, document scope unit **560**, and table manager unit **570** according to an embodiment of the general architecture depicted in FIG. **3** are presented.

[0053] In particular, FIG. **4** depicts one embodiment of formatter unit **510**. A command may be placed in input FIFO **710** indicating that a data stream to follow is to be processed and that the data stream will be encoded in a certain format. Formatter unit **510** may retrieve this command from input FIFO **710** and, subsequently, as data is received, process this data such that the received data is transcoded from its original format into an internal format. Thus, in one embodiment, each token of data received is converted such that the same token is represented in the internal format.

[0054] To facilitate the transcoding of the incoming data stream, data path **720** may have one or more hardware circuits or hardware circuit paths designed for, or dedicated to, the conversion of one particular data format to the internal data format. For example, circuit path **722** may be hardware designed to convert a big endian representation to the internal format, circuitry path **724** may be hardware designed to convert UTF representation to the internal format, etc.

[0055] As it is processed this transcoded data may then be placed in output FIFO queue **730** with one or more commands for segmentation unit **520**. Thus, formatter unit **510** may substantially simultaneously be receiving data and commands, processing data, and placing data and commands in output FIFO queue **730**. By processing data as it arrives at formatter unit **510** (as opposed to waiting for an entire document to be present) more efficient processing of the data may be achieved.

[0056] Moving to FIG. **5**, one embodiment of segmentation unit **520** is represented. Segmentation unit **520** may receive data and commands from input FIFO queue **730** as they are placed there by formatter unit **510**, and output commands and associated data to one or more output FIFO queues **830**, **840** or **850** depending on the type of data associated with the command.

[0057] More specifically, segmentation unit **520** may scan the incoming data based on a definition of the language of the source document to identify certain tokens and keys for the type of document (e.g. special characters or character sets or sequences) and strip the markup and separators from the incoming data stream. This may be done by accessing a data structure in memory **270** or a library that defines the structure of the source document. Based on the markup (i.e. structure of the source document) the incoming data stream may be grouped into frames, each frame comprising a logical grouping or type of data, such as a text string or a symbol, and each data frame routed by router **822** to a logical processing unit **530**, **540**, **550** operable to process the data type of the frame.

[0058] Additionally, segmentation unit **520** may augment the data of a data frame. For example, segmentation unit **520** may perform entity substitution for entities or macros of a data frame by accessing a data structure in memory **270** defining an entity or macro and substituting that definition for the entity or macro in the data. To accelerate these operations (e.g. identification of characters or sequences, entity or macro substitution, etc.) data path **820** may have hardware circuitry to facilitate hexadecimal or decimal comparisons such that character recognition may be performed across multiple characters simultaneously.

[0059] Thus, each of the commands and associated data placed in output FIFO queues **830**, **840** or **850** by segmentation unit **520** may comprise one or more commands framing a particular type of data (which may have been augmented), where the output FIFO queue **830**, **840** or **850** into which the command and associated data frame is placed by router **822** depends on the type of the data frame. By splitting the data stream associated with a source document according to data type, parallel processing of data frames by downstream logical processing units **530**, **540**, **550**, **560**, **570** may be achieved.

[0060] As may be realized then, each of output FIFO queues **830**, **840**, **850** may be associated, respectively, with a logical processing unit **530**, **540**, **550** operable to process a particular data type. FIGS. **6-8** depict, respectively, text unit **530** operable to process text data (e.g. string data), primary symbol unit **550** operable to process symbols of a document, and a secondary symbol processing unit **540** operable to processes symbols in a data stream which are associated with a primary symbol (e.g. a secondary symbol may be a sub-element of a primary element or an attribute associated with an element, etc.).

[0061] Referring now to FIG. **6**, one embodiment of text unit **530** is represented. Text unit **530** may receive commands and data (e.g. data frames) from input FIFO queue **830** as they are placed there by segmentation unit **520** and output commands and associated data to output FIFO queue **970**. More specifically, a command and associated data received at text unit **530** may comprise a command to process a string while the data may comprise the string itself. Text unit **530** may process the string in order to classify the string (e.g. comment, whitespace, etc.). Text unit **530** may access or build tables in memory **270** with a string and the associated type of the string. In one embodiment, text unit may build, access or create a table in memory **270** for each type of string classification, thus one table may be built for whitespace, another may be built for comments, etc. Text unit **530** may then place one or more commands and an associated reference (such as a pointer) to a string in output FIFO queue **970**.

[0062] Now looking at FIG. **7**, one embodiment of primary symbol unit **550** is represented. Primary symbol unit **550** may receive commands and data (e.g. data frames) from input FIFO queues **850**, **1150** as they are placed there by segmentation unit **520** or document scope unit **560** and output commands and associated data to output FIFO queues **1070** and **1060**.

[0063] Primary symbol unit **550** may process symbols received from segmentation unit **520** and check to see if the symbols are valid, and may access a data structure in memory **270** to check to see if the symbol is in the data structure. If the symbol is recognized (e.g. in the data structure), primary symbol unit **550** may append a prefix to an associated symbol identifier to associate the symbol with a particular scope or namespace and store this identifier and symbol in a data structure in memory **270**. If the symbol is not recognized (e.g. not in a data structure in memory **270**) or a data structure referencing these symbols does not exist in memory **270**, the primary symbol unit **550** may create a unique identifier for the symbol (which may include a prefix designating the scope or namespace associated with the symbol), create a data structure in memory **270** to store this symbol data if one does not yet exist, and store the identifier and the associated symbol to the data structure in memory **270**.

[0064] To assist in these processes, the data path **1020** of primary symbol unit **550** may have specialized hardware operable to perform range checking for characters of a symbol and a hash key generators such that a hash value can be generated from a symbol and used for rapidly generating a unique identifier for the symbol or for quickly searching for a symbol in a table in memory **270**.

[0065] During operation, primary symbol unit **520** may desire to resolve the scope or namespace of a symbol. To accomplish this task, primary symbol unit **520** may place a command and associated data in output FIFO queue **1060** (i.e. the input FIFO queue **1060** for document scope unit **560**), such that document scope unit **560** (elaborated on in more detail below) receives the command and data, process the data according to the command, and returns the desired scope or namespace information to primary symbol unit **550** through a command and associated data placed into input FIFO queue of primary symbol unit **550**.

[0066] Moving to FIG. **8**, one embodiment of secondary symbol unit **540** is represented. Secondary symbol unit **540** may receive commands and data (e.g. data frames comprising a symbol) from input FIFO queues **840**, **1240**, as they are placed there by segmentation unit **520** or document scope unit **560**, and output commands and associated data to output FIFO queues **1170** or **1160**.

[0067] Secondary symbol unit **540** may process the symbols received from segmentation unit **520**. This processing may be accomplished by segmenting received symbols according to data type and routing these symbols for processing according to their type. To accomplish this segmentation and processing, secondary symbol unit **540** may comprise secondary segmentation unit **1110**, symbol processing unit **1120** and text unit **1130**, each of may have a substantially similar architecture, and operate substantially similarly to segmentation unit **520**, primary symbol unit **550** and text unit **530**, respectively. Thus, secondary symbol unit **540** may receive a command and associated data from segmentation unit **520** through input FIFO queue **840**. This data may be classified by secondary segmentation unit **1110** and routed to text unit **1130** or symbol unit **1120** based upon the type of the data. Symbol unit **1120** or text unit **1130** can then process the data accordingly.

[0068] Thus, for secondary symbols, macro and character identification and replacement may be performed by secondary symbol unit **540** and one or more index structures accessed or constructed in memory **270** comprising unique identifiers for these secondary symbols. Furthermore, processing directives contained in a structured document, such as parse directives may be identified and routed accordingly.

[0069] Secondary table unit **1140** may operate substantially similarly to table manager unit **570** (elaborated on in more detail below), to construct one or more data structures in memory **270** which represent a portion of the structure of the structured document being processed, for example a subtree of a structured document where the root node of the subtree is a secondary symbol.

[0070] As discussed above, both secondary symbol unit **540** and primary symbol unit **550** may interface with document scope unit **560** to obtain scope or namespace information for one or more pieces of data. FIG. **9** depicts one embodiment of a document scope unit **560**. Document scope unit **560** may receive a command and associated data from secondary symbol unit **540** or primary symbol unit **550** through input FIFO queues **1170** or **1070**, respectively. These commands may be executable to locate data relating to the scope or namespace of the data. Document scope unit **560** may then process the data to obtain the scope or namespace information and return this scope or namespace information to the respective requestor (e.g. the primary symbol unit **550** or secondary symbol unit **540**). Additionally, scope or namespace information, such as unique identifiers for different scope, may be forwarded to table manger unit **570** through output FIFO queue **1270**, such that table manager unit **570** has access to these scope identifiers. To locate the scope or namespace information associated with a symbol, document scope unit **560** may build or access data structures comprising namespaces, and prefixes associated with those namespaces, in memory **270**. Additionally, document scope unit **560** may maintain a scope stack for keeping track of a current scope.

[0071] Data produced by each of text unit **530**, primary symbol unit **550**, secondary symbol unit **540** and document scope unit **560** may be utilized by table manager unit **570** to build a data structure in memory representing the structure of a document and identify document data for applications or other hardware components concurrently with the building of this data structure in memory **270**.

[0072] One embodiment of a table manager unit is depicted in FIG. **10**. Specifically, table manager unit **570** may receive commands and associated data from each of text unit **530**, primary symbol unit **550**, secondary symbol unit **540** and document scope unit **560** through, respectively, input FIFO queues **970**, **1070**, **1170** or **1270**. Concentrator **1310** may receive one or more of these commands and associated data and order these commands appropriately for processing by data path **1320**. Data path **1320** may be optimized for processing these commands to create a data structure in memory, each entry in this data structure corresponding or associated with data of the structured document such that the data structure reflects the structure of the structured document. Concurrently with the construction of the data structure in memory **270**, table manager unit **570** may identify data of the structured document with semantic meaning to applications or other hardware by this data in output FIFO queue **1330**.

[0073] While it should be understood that embodiments of the present invention may be applied with respect to almost any structured document (e.g. a document having a defined structure that can be used to interpret the content) whether the content is highly structured (such as an XML document, Hypertext Markup Language (HTML) document, .pdf document, word processing document, database, etc.) or loosely structured (such as a plain text document whose structure may be, e.g., a stream of characters), it may be useful to illustrate one particular embodiment of a parser in conjunction with an architecture for transforming XML or other structured documents utilizing a set of transformation instructions for the XML document (e.g. a stylesheet). While this illustration of the uses of one embodiment of a parser such as that described herein may helpful it will be apparent that, as discussed above, embodiments of a parser may be utilized in a wide variety of other architectures and may be applied to parse structured document with or without the use of transformation instructions or pre-generated data structures, etc.

[0074] Attention is now directed to an architecture for the efficient transformation or processing of structured documents in which an embodiment of a parser may be utilized. Embodiments of the architecture may comprise an embodiment of the aforementioned parser along with other logical components including a pattern expression processor, a transformation engine and an output generator, one or more of which may be implemented in hardware circuitry, for example a hardware processing device such as an Application Specific Integrated Circuit (ASIC) which comprises all the above mentioned logical components.

[0075] More particularly, transformation instructions may be compiled to generate instruction code and a set of data structures. The parser parses the structured document associated with the transformation instructions to generate structures representative of the structured document. The pattern expression processor (PEP) identifies data in the structured document corresponding to definitions in the transformation instructions. The transformation engine transforms the parsed document according to the transformation instructions and the output generator assembles this transformed data into an output document.

[0076] Turning to FIG. **11**, a block diagram for the transformation of structured documents using embodiments of the present invention is depicted. A structured document may be received at a web service **112** from a variety of sources such as a file server, database, internet connection, etc. Additionally, a set of transformation instructions, for example an XSLT stylesheet, may also be received. Document processor **210** may apply the transformation instructions to the structured document to generate an output document which may be returned to the requesting web service **112**, which may, in turn, pass the output document to the requester.

[0077] In one embodiment, compiler **220**, which may comprise software (i.e. a plurality of instructions) executed on one or more processors (e.g. distinct from document processor **210**) may be used to compile the transformation instructions to generate data structures and instruction code in memory **270** for use by document processor **210**. Document processor **210** may be one or more ASICs operable to utilize the data structures and instruction code generated by compiler **220** to generate an output document.

[0078] FIG. **12** depicts a block diagram of one embodiment of an architecture for a document processor operable to produce an output document from a structured document. Document processor **210** comprises Host Interface Unit (HIU) **310**, Parser **320**, PEP **330**, Transformation Engine

(TE) **340**, Output Generator (OG) **350**, each of which is coupled to memory interface **360**, to Local Command Bus (LCB) **380** and, in some embodiments, to one another through signal lines or shared memory **270** (e.g. a source unit may write information to be communicated to a destination unit to the shared memory and the destination unit may read the information from the shared memory), or both. Shared memory **270** may be any type of storage known in the art, such as RAM, cache memory, hard-disk drives, tape devices, etc.

[**0079**] HIU **310** may serve to couple document processor **210** to one or more host processors (not shown). This coupling may be accomplished, for example, using a peripheral component interconnect express (PCI-X) bus. HIU **310** also may provide an Applications Programming Interface (API) through which document processor **210** can receive jobs. Additionally, HIU **310** may interface with LCB **380** such that various tasks associated with these jobs may be communicated to components of document processor **210**.

[**0080**] In one embodiment, these jobs may comprise context data, including a structured document, data structures, and instruction code generated from transformation instructions by the compiler. Thus, the API may allow the context data to be passed directly to HIU **310**, or, in other embodiments, may allow references to one or more locations in shared memory **270** where context data may be located to be provided to HIU **310**. HIU **310** may maintain a table of the various jobs received through this API and direct the processing of these jobs by document processor **210**. By allowing multiple jobs to be maintained by HIU **310**, these jobs may be substantially simultaneously processed (e.g. processed in parallel) by document processor **210**, allowing document processor **210** to be more efficiently utilized (e.g. higher throughput of jobs and lower latency).

[**0081**] Parser **320** may receive and parse a structured document, identifying data in the structured document for PEP **330** and generating data structures comprising data from the structured document by, for example, creating data structures in shared memory **270** for use by TE **340** or OG **350**.

[**0082**] PEP **330** receives data from parser **320** identifying data of the structured document being processed and compares data identified by the parser **320** against expressions identified in the transformation instructions. PEP **330** may also create one or more data structures in shared memory **270**, where the data structures comprises a list of data in the structured document which match expressions.

[**0083**] Transformation engine **340** may access the data structures built by parser **320** and PEP **330** and execute instruction code generated by compiler **220** and stored in memory **270** to generate results for the output document. In some embodiments, one or more instructions of the instruction code generated by compiler **220** may be operable to be independently executed (e.g. execution of one instruction does not depend directly on the result of the output of the execution of another instruction), and thus execution of the instruction code by transformation engine **340** may occur in substantially any order.

[**0084**] Output generator **350** may assemble the results generated by transformation engine **340** in an order specified by the transformation instructions and may write the output document to shared memory **270**. The output document may then be provided to the initiating web service **112** through HIU **310**, for example, by signaling the web service **112** or a host processor that the job is complete and providing a reference to a location in memory **270** where an output document exists.

[**0085**] Moving now to FIG. **13**, an example application of one embodiment of the present invention to an XML document and an XSLT stylesheet is illustrated. It is noted that, while the description herein may include examples in which transformation instructions are applied to a single source document, other examples may include applying multiple sets of transformation instructions to a source document (either concurrently or serially, as desired) or applying a set of transformation instructions to multiple source documents (either concurrently with context switching or serially, as desired). Generally, an XML document is a structured document which has a hierarchical tree structure, where the root of the tree identifies the document as a whole and each other node in the document is a descendent of the root. Various elements, attributes, and document content form the nodes of the tree. The elements define the structure of the content that the elements contain. Each element has an element name, and the element delimits content using a start tag and an end tag that each include the element name. An element may have other elements as sub-elements, which may further define the structure of the content. Additionally, elements may include attributes (included in the start tag, following the element name), which are name/value pairs that provide further information about the element or the structure of the element content. XML documents may also include processing instructions that are to be passed to the application reading the XML document, comments, etc.

[**0086**] An XSLT stylesheet is a set of transformation instructions which may be viewed as a set of templates. Each template may include: (i) an expression that identifies nodes in a document's tree structure; and (ii) a body that specifies a corresponding portion of an output document's structure for nodes of the source document identified by the expression. Applying a stylesheet to a source document may comprise attempting to find a matching template for one or more nodes in the source document, and instantiating the structures corresponding to the body of the matching template in an output document.

[**0087**] The body of a template may include one or more of: (i) literal content to be instantiated in the output document; (ii) instructions for selection of content from the matching nodes to be copied into the output document; and (iii) statements that are to be evaluated, with the result of the statements being instantiated in the output document. Together, the content to be instantiated and the statements to be evaluated may be referred to as "actions" to be performed on the nodes that match the template.

[**0088**] The body of a template may include one or more "apply templates" statements, which include an expression for selecting one or more nodes and causing the templates in the stylesheet to be applied to the selected nodes, thus effectively nesting the templates. If a match to the apply templates statement is found, the resulting template is instantiated within the instantiation of the template that includes the apply templates statement. Other statements in

the body of the template may also include expressions to be matched against nodes (and the statements may be evaluated on the matching nodes).

[0089] The expressions used in a stylesheet may generally comprise node identifiers and/or values of nodes, along with operators on the node identifiers to specify parent/child (or ancestor/descendant) relationships among the node identifiers and/or values. Expressions may also include predicates, which may be extra condition(s) for matching a node. A predicate is an expression that is evaluated with the associated node as the context node (defined below), where the result of the expression is either true (and the node may match the expression node) or false (and the node does not match the expression). Thus, an expression may be viewed as a tree of nodes to be matched against a document's tree.

[0090] A given document node may satisfy an expression if the given document node is selected via evaluation of the expression. That is, the expression node identifiers in the expression match the given document node's identifier or document node identifiers having the same relationship to the given document node as specified in the expression, and any values used in the expression are equal to corresponding values related to the given document node.

[0091] A document node may also be referred to as a "matching node" for a given expression if the node satisfies the given expression. In some cases in the remainder of this discussion, it may be helpful for clarity to distinguish nodes in expression trees from nodes in a structured document. Thus, a node may be referred to as an "expression node" if the node is part of an expression tree, and a node may be referred to as a "document node" if the node is part of the document being processed. A node identifier may comprise a name (e.g. element name, attribute name, etc.) or may comprise an expression construct that identifies a node by type (e.g. a node test expression may match any node, or a text test expression may match any text node). In some cases, a name may belong to a specific namespace. In such cases, the node identifier may be a name associated with a namespace. In XML, the namespace provides a method of qualifying element and attribute names by associating them with namespace names. Thus, the node identifier may be the qualified name (the optional namespace prefix, followed by a colon, followed by the name). A name, as used herein (e.g. element name, attribute name, etc.) may include a qualified name. Again, while XSLT stylesheets may be used in one example herein of transformation instructions, generally a "transformation instructions" may comprise any specification for transforming a source document to an output document, which may encompass, for example, statements indented to identify data of the source document or statements for how to transform data of the source document. The source and output documents may be in the same language (e.g. the source and output documents may be different XML vocabularies), or may differ (e.g. XML to pdf, etc.).

[0092] Referring still to FIG. **13**, an XML document and an associated XSL stylesheet may be received by web service **112**. Web service **112** may invoke embodiments of the present invention to transform the received document according to the received stylesheet. More specifically, in one embodiment, compiler **220** may be used to compile the XSL stylesheet to generate data structures and instruction code for use by document processor **210**. Compiler **220** may assign serial numbers to node identifiers in the stylesheet so that expression evaluation may be performed by document processor **210** by comparing numbers, rather than node identifiers (which would involve character string comparisons).

[0093] Compiler **220** may also store a mapping of these node identifiers to serial numbers in one or more symbol tables **410** in memory **270**. Additionally, compiler **220** may extract the expressions from the stylesheet and generate expression tree data structures in memory **270** to be used by the document processor **210** for expression matching (e.g. one or more parse-time expression trees **420** comprising expression nodes). Still further, compiler **220** may generate an instruction table **430** in memory **270** with instructions to be executed for one or more matching expressions. The instructions in the instruction table **430**, when executed by document processor **210**, may result in performing the actions defined when an expression associated with the instruction is matched. In some embodiments, the instructions may comprise the actions to be performed (i.e. there may be a one-to-one correspondence between instructions and actions). The compiler may also generate whitespace tables **440** defining how various types of whitespace in the source document are to be treated (e.g. preserved, stripped, etc.), an expression list table **450**, a template list table **460** and one or more DTD tables **462** to map entity references to values or specify default values for attributes.

[0094] At this point, processing of the source document by document processor **210** may begin. Parser **320** receives the structured document and accesses the symbol tables **410**, whitespace tables **440**, or DTD tables **462** in memory **470** to parse the structured document, identify document nodes, and generate events (e.g. to identify document nodes parsed from the document) to PEP **330**. More particularly, parser **320** converts node identifiers in the source document to corresponding serial numbers in the symbol tables **410**, and transmits these serial numbers as part of the events to the PEP **330**. Additionally, parser **320** may generate a parsed document tree **470** representing the structure of the source document in memory. Nodes of the parsed document tree may reference corresponding values stored in one or more parsed content tables **472** created in memory by parser **320**.

[0095] PEP **330** receives events from the parser **320** and compares identified document nodes (e.g. based on their serial numbers) against parse-time expression tree(s) **420** in memory **270**. Matching document nodes are identified and recorded in template or expression match lists **480** in memory **270**.

[0096] Transformation engine **340** accesses the template or expression match lists **480**, the parsed document tree **470**, the parsed content tables **472** or the instruction table **430**. The transformation engine **340** executes instructions from the instruction table **430** in memory **270**. These instructions may be associated with one or more expressions. Transformation engine **340** may execute the instructions on each of the document nodes that matches the expression associated with the expression. Transformation engine **340** may store the results of the execution of these instructions in one or more tables in memory and forwards references to these results to output generator **350**.

[0097] Output generator **350** may access form output tables **490** from these references to construct an output

document. In some embodiments, output generator **350** may access a set of formatting parameters for the assembly of the output document. After the output document is assembled the output document may be returned to the proper web service **112**.

[**0098**] In one embodiment, a host may provide a source document as it is received or accessed and the parser **320** may transmit events to the PEP **330** as they are generated, as well. That is, as the portions of the source document are received by the host, the host passes the portion of the document to the parser **320** (e.g. through the HIU **310**). The parser **320** may thus begin parsing prior to the host receiving the entirety of the source document. Similarly, events are passed to the PEP **330** as they are identified or generated, allowing PEP **330** to process these events before parser **320** has completed the processing of the source document(s).

[**0099**] Since parser **320** may process source documents as they are presented (i.e. without having to have the entire document present before processing may begin), the throughput of parser **320** may be maximized and latency minimized. During the processing of these structured documents, parser **320** may operate to create and organize data structures in memory for efficient parallelized data processing by downstream logical components PEP **330**, transformation engine **340** or output generator **350** of processor **210**.

[**0100**] The operation of the particular embodiment of parser **320** for use with document processor **210** may be elaborated on again with reference to FIG. **5**. Formatter unit **510** may retrieve commands from input FIFO **710** and, subsequently, as data is received from host unit **310**, process this data such that the received data is transcoded from its original format into an internal format.

[**0101**] As it is processed this transcoded data may then be placed in output FIFO queue **730** with one or more commands for segmentation unit **520**. Segmentation unit **520** may receive data and commands from input FIFO queue **730** as they are placed there by formatter unit **510**, and output commands and associated data to one or more output FIFO queues **830**, **840** or **850** depending on the type of data associated with the command.

[**0102**] More specifically, segmentation unit **520** may scan the incoming data and strip the markup and separators from the incoming data stream. The incoming data stream of the XML document may then be grouped into frames, each frame comprising a text string or a symbol, and each data frame routed by router **822** to a logical processing unit **530**, **540**, **550** operable to process the data type of the frame.

[**0103**] Additionally, segmentation unit **520** may augment the data of a data frame. For example, segmentation unit **520** may perform entity substitution for entities or macros of a data frame by accessing a DTD table **462** to determine if an entity or macro is defined and, if it is, substituting that definition for the entity or macro in the data, or may access whitespace table **440** to determine how whitespace within the framed data is to be dealt with (e.g. stripped or preserved).

[**0104**] Thus, each of the commands and associated data placed in output FIFO queues **830**, **840** or **850** by segmentation unit **520** may comprise one or more commands framing a particular type of data (which may have been augmented), Text unit **530** may receive commands and framed string data from input FIFO queue **830**. More specifically, a command and associated data received at text unit **530** may comprise a command to process a string while the data may comprise the string itself. Text unit **530** processes the string in order to classify the string (e.g. comment, whitespace, etc.). Text unit **530** may access or build a parsed content table **472** (e.g. an element index table or element name/value table) in memory **270** with a string and an associated type of the string. In one embodiment, text unit may build, access or create a table in memory **270** for each type of string classification, thus one table may be built for whitespace, another may be built for comments, etc. Text unit **530** may then place one or more commands and an associated reference (such as a pointer) to a string in output FIFO queue **970**.

[**0105**] Primary symbol unit **550** may process the symbols received from segmentation unit **520** and check to see if the symbols are valid, and may access a symbol table **410** in memory **270** to check to see if the symbol is recognized. If the symbol is recognized primary symbol unit **550** may append a prefix to an associated symbol identifier to associate the symbol with a particular scope or namespace and store this identifier and symbol to a parsed content table **472** such as an element index table or element name/value table. If the symbol is not recognized (e.g. not in a symbol table **410** in memory **270**) primary symbol unit **550** may create a unique identifier for the symbol (which may include a prefix designating the scope or namespace associated with the symbol) and store this identifier and the associated symbol to a symbol table **410** in memory **270**.

[**0106**] Secondary symbol unit **540** may receive commands and data (e.g. data frames comprising a symbol) from input FIFO queues **840**, **1240**, as they are placed there by segmentation unit **520** or document scope unit **560**, and output commands and associated data to output FIFO queues **1170** or **1160**. As elaborated on above, secondary symbol unit **540** may comprise secondary segmentation unit **1110**, symbol processing unit **1120** and text unit **1130**, each of may have a substantially similar architecture, and operate substantially similarly to segmentation unit **520**, primary symbol unit **550** and text unit **530**, respectively. Thus, for secondary symbols macro and character identification and replacement may be performed by secondary symbol unit **540** and one or more parsed content table **472** (e.g. an element index table or element name/value table) accessed or constructed in memory **270** comprising unique identifiers for these secondary symbols. Furthermore, processing directives contained in the structured document, such as parse directives may be identified and routed accordingly.

[**0107**] Secondary table unit **1140** may operate substantially similarly to table manager unit **570** (as elaborated on above), to construct one or more data structures which represent a portion of the structure of the structured document being processed, for example a subtree of the structured document where the root node of the subtree is a secondary symbol.

[**0108**] Results produced by each of text unit **530**, primary symbol unit **550**, secondary symbol unit **540** and document scope unit **560** may be utilized by table manager unit **570** to build a data structure in memory representing the structure of a document and identify node information (e.g. events) for PEP **330** concurrently with the building of the data structure in memory **270**.

[0109] Specifically, table manager unit **570** may receive commands and associated data from each of text unit **530**, primary symbol unit **550**, secondary symbol unit **540** and document scope unit **560** through, respectively, input FIFO queues **970**, **1070**, **1170** or **1270**. Table manager unit may construct a parsed content table **472** reflecting the structure of the XML document, where each entry in this parsed content table **472** corresponds to a node of the XML document and comprises a node record with pointers or references to one or more entries in other parsed content tables **472** comprising information on the node, and order information such as references to parent nodes and child nodes such that the parsed content table **472** reflects the structure of the structured document. Concurrently with the construction of this data structure in memory **270**, table manager unit **570** may identify node information (e.g. events) for PEP **330** by placing commands and associated data in output FIFO queue **1330**.

[0110] In the foregoing specification, the invention has been described with reference to specific embodiments. However, one of ordinary skill in the art appreciates that various modifications and changes can be made without departing from the scope of the invention as set forth in the claims below. Accordingly, the specification and figures are to be regarded in an illustrative rather than a restrictive sense, and all such modifications are intended to be included within the scope of invention. For example, it will be apparent to those of skill in the art that although the present invention has been described with respect to a protocol controller in a routing device the inventions and methodologies described herein may be applied in any context which requires the determination of the protocol of a bit stream.

[0111] Benefits, other advantages, and solutions to problems have been described above with regard to specific embodiments. However, the benefits, advantages, solutions to problems, and any component(s) that may cause any benefit, advantage, or solution to occur or become more pronounced are not to be construed as a critical, required, or essential feature or component of any or all the claims.

What is claimed is:

1. An apparatus, comprising

a parser hardware circuit operable to receive a structured document, and parse the structured document to create a set of data structures, wherein the set of data structures are representative of the structured document.

2. The apparatus of claim 1, wherein the parser hardware circuit is operable to parse the structured document and create the set of data structures substantially simultaneously.

3. The apparatus of claim 2, wherein the parser hardware circuit comprises a formatter circuit operable to encode the structured document in an internal format.

4. The apparatus of claim 3, wherein the parser hardware circuit comprises a segmentation unit operable to segment the structured document into a set of data frames and route each data frame according to one of a set of data types associated with each data frame, wherein each data frame is routed to one of a set of hardware circuits operable to process one of the set of data types.

5. The apparatus of claim 4, wherein each of the data frames is associated with a first command

6. The apparatus of claim 5, wherein the parser circuit comprises the set of hardware circuits and the set of hardware circuits comprises a text circuit, a primary symbol circuit and a secondary symbol circuit, each of the hardware circuits operable to execute the first command in conjunction with the associated data frame.

7. The apparatus of claim 6, wherein the parser circuit comprises a document scope circuit operable to execute the second command to associate an order with a data frame.

8. The apparatus of claim 6, wherein the parser circuit comprises a table manager circuit operable to construct the set of data structures representative of the structured document.

9. A system, comprising:

a compiler operable to generate a first set of data structures from a set of transformation instructions corresponding to a structured document; and

a parser hardware circuit operable to receive the structured document, and parse the structured document, utilizing the first set of data structures, to create a second set of data structures, wherein the second set of data structures are representative of the structured document.

10. The system of claim 9, wherein the parser hardware circuit is operable to parse the structured document and create the second set of data structures substantially simultaneously.

11. The system of claim 10, wherein the parser is operable to generate a set of events associated with data of the structured document substantially simultaneously with the parsing of the structured document and the creation of the second set of data structures.

12. The system of claim 11, comprising a pattern expression processor circuit operable to create a third set of data structures based on the set of events, wherein the creation of the third set of data occurs substantially simultaneously with the parsing of the structured document and the creation of the second set of data structures.

13. The system of claim 12, wherein the parser circuit comprises a table manager circuit operable to construct the second set of data structures representative of the structured document and generate the set of events.

14. The system of claim 13, wherein the compiler is operable to create a set of instructions.

15. The system of claim 14, comprising a transformation engine circuit operable to access the first set of data structures, the second set of data structures or the third set of data structures to execute the set of instructions to transform the structured document according to the transformation instructions.

16. A method, comprising:

segmenting a received structured document into a set of data frames; and

processing the data frames based on one of a set of types associated with each data frame to create a first set of data structures representative of the structured document, wherein segmenting the structured document and processing the set of data frames occurs substantially simultaneously with the document being received.

17. The method of claim 16, wherein processing the set of data frames and creating the first set of data structures occurs substantially simultaneously.

18. The method of claim 17, comprising encoding the structured document in an internal format.

19. The method of claim 18, wherein the set of types comprise text, primary symbols and secondary symbols.

20. The method of claim 19, comprising associating an order with one or more of the data frames.

21. The method of claim 20, comprising generating a set of events associated with data of the structured document substantially simultaneously with the parsing of the structured document and the creation of the second set of data structures.

22. The method of claim 21, wherein processing the set of data frames utilizes a second set of data structures.

23. The method of claim 22, wherein the second set of data structures was generated by a compiling a set of transformation instructions corresponding to the structured document.

* * * * *