US008782230B1

(12) **United States Patent**
Swedor et al.

(10) **Patent No.: US 8,782,230 B1**
(45) **Date of Patent: Jul. 15, 2014**

(54) **METHOD AND APPARATUS FOR USING A COMMAND DESIGN PATTERN TO ACCESS AND CONFIGURE NETWORK ELEMENTS**

(75) Inventors: **Olivier K. Swedor**, Auvernier (CH); **Tal I. Lavian**, Sunnyvale, CA (US); **Robert J. Duncan**, San Francisco, CA (US)

(73) Assignee: **Rockstar Consortium US LP**, Plano, TX (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1162 days.

(21) Appl. No.: **09/726,758**

(22) Filed: **Nov. 29, 2000**

Related U.S. Application Data

(60) Provisional application No. 60/213,107, filed on Jun. 21, 2000.

(51) **Int. Cl.**
G06F 15/16 (2006.01)

(52) **U.S. Cl.**
USPC **709/226**

(58) **Field of Classification Search**
USPC 715/513; 709/229, 223, 224; 370/401
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,848,233 A * 12/1998 Radia et al. 726/13
6,167,448 A * 12/2000 Hemphill et al. 709/224
6,292,489 B1 * 9/2001 Fukushima et al. 370/401
6,463,528 B1 * 10/2002 Rajakarunanayake et al. ... 713/1
6,546,419 B1 * 4/2003 Humpleman et al. 709/223
6,662,342 B1 * 12/2003 Marcy 715/513
6,880,005 B1 * 4/2005 Bell et al. 709/225
6,973,488 B1 * 12/2005 Yavatkar et al. 709/223
7,054,901 B2 * 5/2006 Shafer 709/203
2002/0032768 A1 * 3/2002 Voskuil 709/224
2004/0133776 A1 * 7/2004 Putzolu 713/153

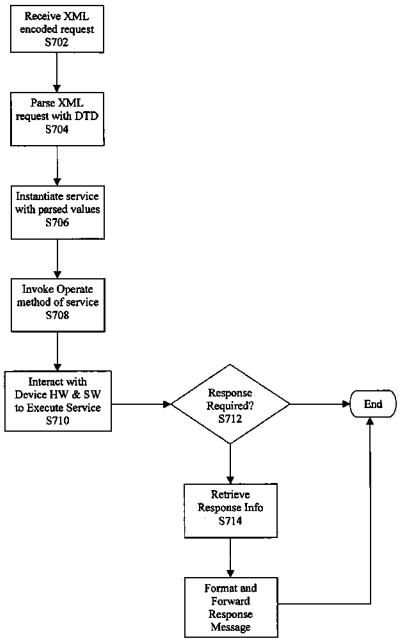
* cited by examiner

Primary Examiner — Tamara T Kyle
(74) *Attorney, Agent, or Firm* — David A. Dagg

(57) **ABSTRACT**

An XML accessible network device is capable of performing functions in response to an XML encoded request transmitted over a network. It includes a network data transfer service, coupled to a network, that is capable of receiving XML encoded requests from a client also connected to the network. A service engine is capable of understanding and parsing the XML encoded requests according to a corresponding DTD. The service engine further instantiates a service using parameters provided in the XML encoded request and launches the service for execution on the network device in accordance with a command design parameter. A set of device APIs interacts with hardware and software on the network device for executing the requested service on the network device. If necessary, a response is further collected from the device and provided to the client in a response message.

21 Claims, 4 Drawing Sheets



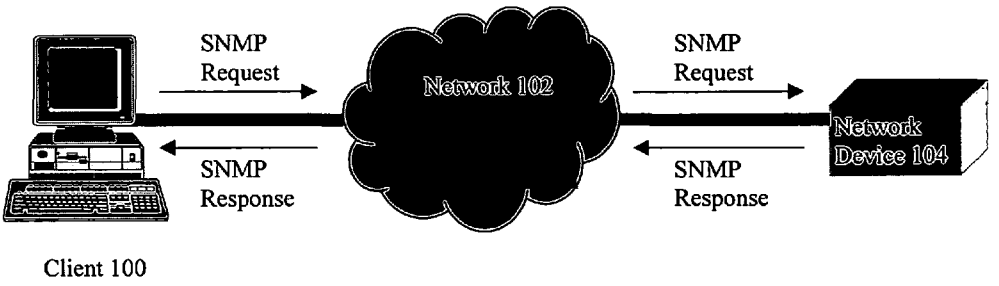


FIG. 1 (Prior Art)

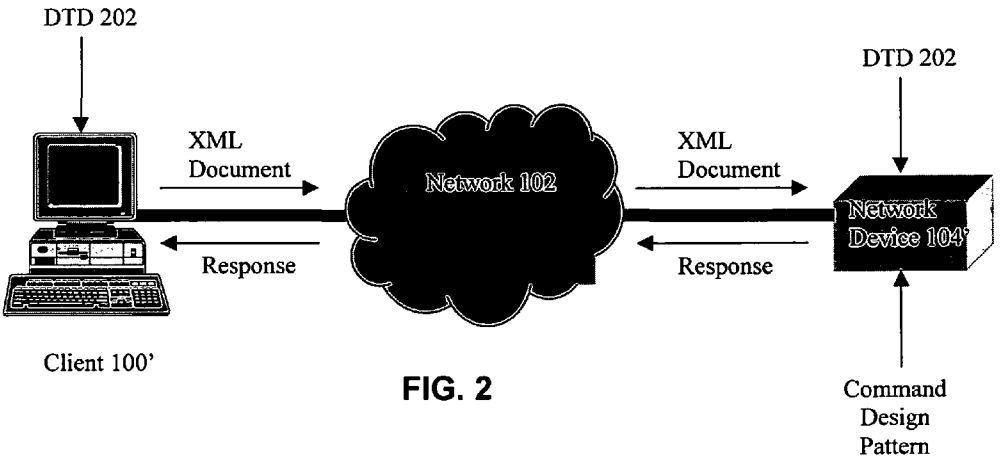


FIG. 2

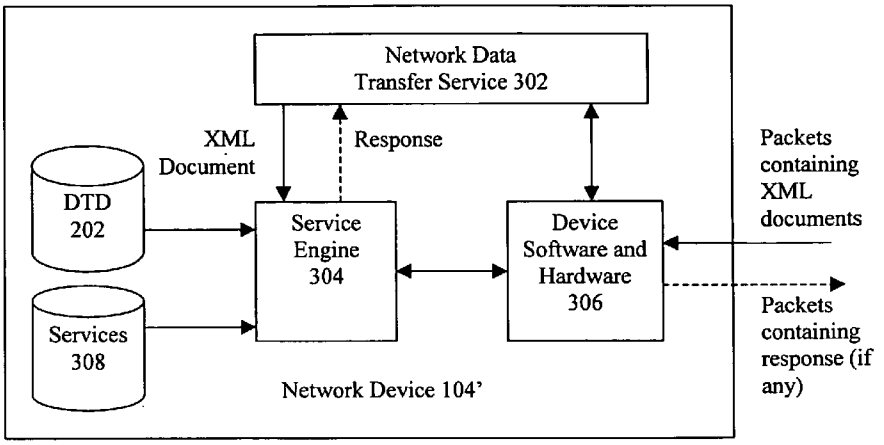

Network Data
Transfer Service 302
XML
Document
Response
DTD
202
Service
Engine
304
Device
Software and
Hardware
306
Packets
containing
XML
documents
Packets
containing
response (if
any)
Services
308
Network Device 104'

FIG. 3

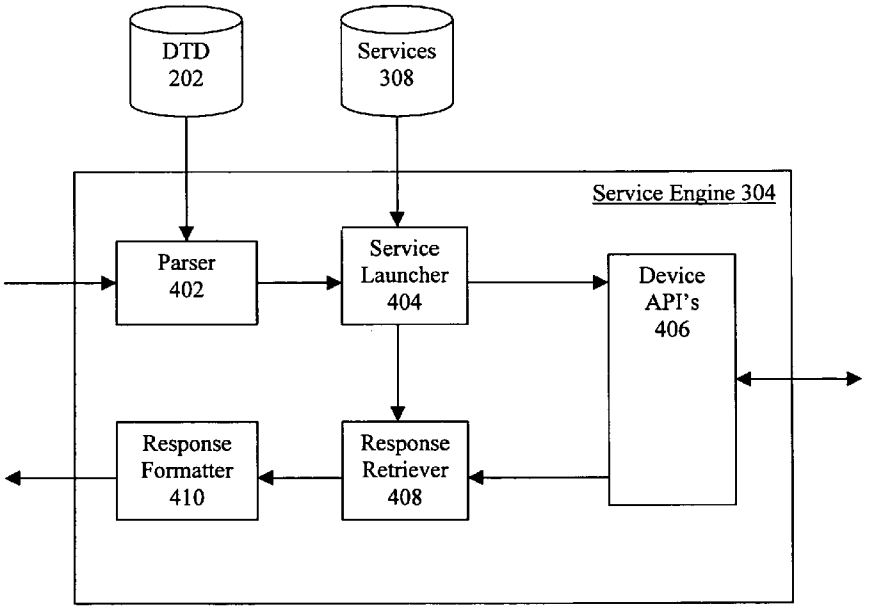

DTD
202
Services
308
Service Engine 304
Parser
402
Service
Launcher
404
Device
API's
406
Response
Formatter
410
Response
Retriever
408

FIG. 4

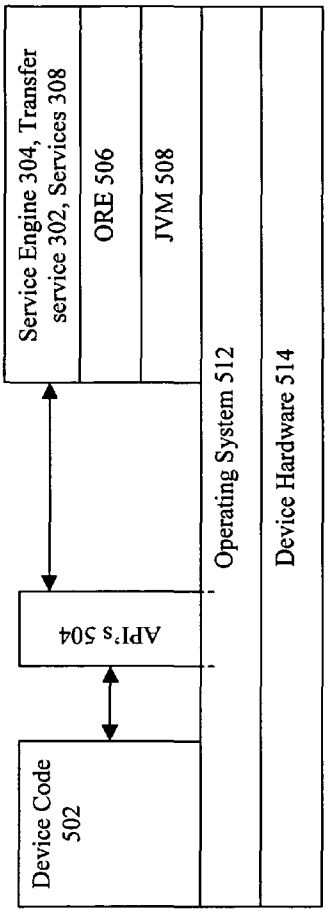

Service Engine 304, Transfer service 302, Services 308
ORE 506
JVM 508
Operating System 512
Device Hardware 514
APIs 504
Device Code 502

FIG. 5

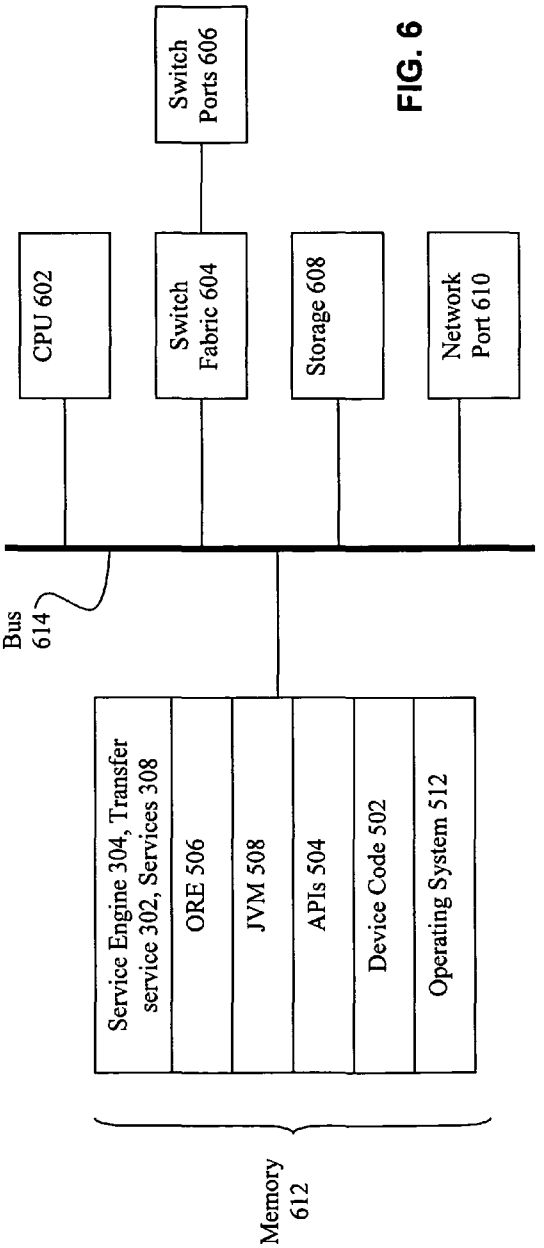

Switch Ports 606
CPU 602
Switch Fabric 604
Storage 608
Network Port 610
Bus 614
Service Engine 304, Transfer service 302, Services 308
ORE 506
JVM 508
APIs 504
Device Code 502
Operating System 512
Memory 612

FIG. 6

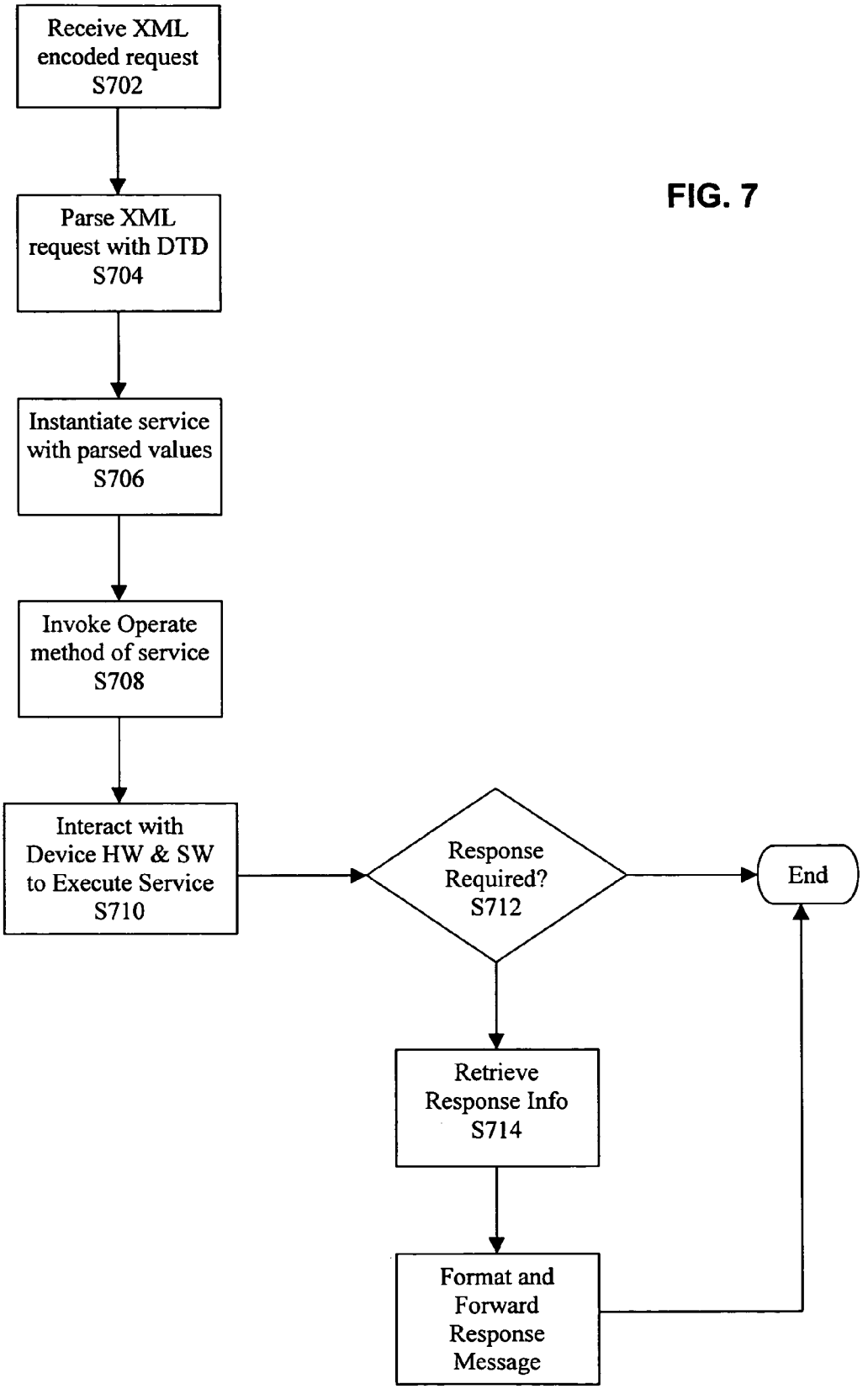

Receive XML encoded request S702
Parse XML request with DTD S704
Instantiate service with parsed values S706
Invoke Operate method of service S708
Interact with Device HW & SW to Execute Service S710
Response Required? S712
End
Retrieve Response Info S714
Format and Forward Response Message

FIG. 7

METHOD AND APPARATUS FOR USING A COMMAND DESIGN PATTERN TO ACCESS AND CONFIGURE NETWORK ELEMENTS

CROSS-REFERENCE TO RELATED APPLICATIONS

The present application is based on, and claims priority from U.S. Provisional Appln. No. 60/213,107, filed Jun. 21, 2000. The present application is also related to U.S. application Ser. No. 09/692,949 (NOR-12520BA) and U.S. Appln. Ser. No. 09/727,341 (NOR-12675HU), both commonly owned by the assignee of the present application.

FIELD OF THE INVENTION

The present invention relates to network device configuration and monitoring, and more particularly, to a method and apparatus for accessing, configuring and controlling a device stationed on a network using a command design pattern and documents written in a markup language such as XML.

BACKGROUND OF THE INVENTION

Computer networks continue to proliferate. As they do so, they become increasingly complex and difficult to manage. This problem is exacerbated when a variety of network devices, computers, and software are combined together to integrate large intranets with the Internet.

As shown in FIG. **1**, when a client **100** wants to learn information regarding a remote network device **104** stationed on a network **102**, code executing on client **100** formats a message requesting such information and sends it to the network device **104**. Network device **104** must be preprogrammed with functionality for communicating in the protocol required by client **100**'s message and for knowing exactly how to get the information requested. If so, network device **104** can then respond with the requested information.

Simple network management protocol (SNMP) is one example of a network protocol that allows clients to learn information about remote network devices. SNMP allows network devices **104** to send alerts to a manager **102**, or to send statistical information about traffic, but it limits the kind of information that can be sent to that which is pre-defined in the management information blocks (MIBs) coded into the network device. Accordingly, a new MIB needs to be redefined each time a new type of information is maintained or is needed about the device, thus making network management and performance even more problematic.

SUMMARY OF THE INVENTION

The present invention relates to an apparatus and method for more efficiently accessing, configuring and controlling a network device using a common design pattern and documents written in a markup language such as the Extensible Markup Language (XML).

In accordance with one aspect of the invention, an XML accessible network device is capable of performing functions in response to an XML encoded request transmitted over a network. It includes a network data transfer service, coupled to a network, that is capable of receiving XML encoded requests from a client also connected to the network. A service engine is capable of understanding and parsing the XML encoded requests according to a corresponding document type definition (DTD). The service engine further instantiates a service using parameters provided in the XML encoded request and launches the service for execution on the network device using a command design parameter. A set of device APIs interacts with hardware and software on the network device for executing the requested service on the network device. If necessary or desired, a response is further collected from the device and provided to the client in a response message.

In accordance with another aspect of the invention, a method for causing a network device to locally perform a service comprises the steps of receiving at the network device a document written in accordance with a markup language and a corresponding document definition, parsing by the network device the received document in accordance with the corresponding document definition, and executing the service on the network device in accordance with the parsed document and a command design parameter.

In accordance with a further aspect of the invention, a network device for locally performing a service in response to a remote request comprises means for receiving at the network device a document written in accordance with a markup language and a corresponding document definition, means for parsing by the network device the received document in accordance with the corresponding document definition, and means for executing the service on the network device in accordance with the parsed document and a command design parameter.

In accordance with a further aspect of the invention, a network device for locally performing a service in accordance with a received document written in a document markup language comprises a parser that is adapted to parse the received document in accordance with a document definition, a service engine coupled to the parser that is adapted to instantiate an object corresponding to the service in accordance with the parsed received document, and to execute the service in accordance with the instantiated object and a command design parameter.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing and other features, aspects, and advantages of the present invention will become more apparent from the following detailed description when read in conjunction with the following drawings, wherein:

FIG. **1** illustrates a conventional architecture for accessing, configuring and controlling a network device using standard network protocols;

FIG. **2** is a functional overview of an apparatus for accessing, configuring and controlling a network device using XML encoded data and a command design parameter in accordance with the present invention;

FIG. **3** further illustrates an example of a network device that is configured in accordance with the present invention;

FIG. **4** further illustrates a service engine that can be included in a network device according to the invention such as that illustrated in FIG. **3**;

FIG. **5** is an architectural view of a network device that is configured in accordance with the present invention;

FIG. **6** is an example implementation of a network device in accordance with the invention and the architecture depicted in FIG. **5**; and

FIG. **7** illustrates a process for accessing, configuring and controlling a network device using XML encoded data in accordance with the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention will now be described in detail with reference to the accompanying drawings, which are provided

as illustrative examples of preferred embodiments of the present invention. Notably, the implementation of certain elements of the present invention may be accomplished using software, hardware or any combination thereof, as would be apparent to those of ordinary skill in the art, and the figures and examples below are not meant to limit the scope of the present invention. Moreover, where certain elements of the present invention can be partially or fully implemented using known components, only those portions of such known components that are necessary for an understanding of the present invention will be described, and detailed descriptions of other portions of such known components will be omitted so as not to obscure the invention. Further, the present invention encompasses present and future known equivalents to the known components referred to herein by way of illustration.

FIG. **2** is a functional overview of one embodiment of the present invention. As shown, a client **100'** is connected in the conventional manner to a network such as the Internet or an intranet (i.e. LAN/WAN/MAN) **102**, which in turn is connected to a network device **104'** (e.g. router, switch, hub or similar device capable of processing fixed-length or variable-length packets in a network). It should be noted that although the features and advantages of the present invention are particularly well suited to routers, switches and hubs, and will be described in more detail below with reference to such devices, other network-aware devices can be adapted for use in the present invention, and so the invention is not to be limited to these particular illustrative devices. For example, network device **104'** may also include gateways, multiplexers and other known or future equivalents, including those having a packet forwarding architecture.

As is apparent from FIG. **2**, in contrast with the prior art, the present invention provides communications for accessing and configuring network elements using XML documents and a common design pattern. As is known, XML is a metalanguage built upon the Standard Generalized Markup Language (SGML), and is a tool for defining text markup languages, defined by the World Wide Web Consortium (W3C). An XML-defined language comprises a set of tags, attributes, and constraints on how to use them. XML is a simple, open, portable, extensible means of representing data. Unlike HTML, XML tags tell what the data means, instead of just how to display it. Further information regarding XML can be found from the W3C web pages at http://www.w3.org/XML.

Those skilled in the art, after being taught by the present disclosure, will appreciate that there are many flavors of XML and SGML and that many markup languages are equivalent to XML for adaptation and implementation according to the present invention. XML is described in detail in this application because of its wide acceptance and adoption. However, equivalents to XML that are within the scope of the invention can include, for example, XSL, XSLT, XPath, XLink, XPointer, HyTime, DSSSL, CSS, SPDL, CGM, ISO-HTML, and others.

As is further known, the format of an XML document is defined by a Document Type Definition (DTD). Accordingly, the present invention includes local copies of DTD **202** at the sending and receiving ends of the XML document. There can be just one DTD **202** that defines all communications for all applications, or there may be different DTDs **202** for different types of applications, and those skilled in the art will understand the possible variations. Further, there are many ways known in the art that such local copies of DTD **202** can be retrieved and obtained by network device **104'**, and the details thereof will not be presented so as not to obscure the invention.

Generally, when the client computer **100'** wants to access or configure network device **104'** (i.e., cause network device **104'** to perform a service locally on the network device **104'**), it creates an XML request identifying the service to be performed, which XML document is encoded in the format defined by DTD **202** but with parameters specific to the desired access or configuration. Client **100'** sends the document over the network **102**, which document is received by the network device **104'** in the form of data packets as is conventionally done. In contrast to the conventional network device **104**, the network device **104'** has been adapted to include functionality necessary to decode the XML encoded request and to identify the service to be performed. Using a common design pattern, the network device **104'** allows the service to perform tasks necessary to complete the request, including, if required, interacting with the hardware and software of the device to perform a service locally on the network device **104'**. If further needed or desired by the client computer, network device **104'** may create and format a response to client computer **100'**, which may or may not also be an XML encoded document.

FIG. **3** illustrates a network device **104'** in accordance with the present invention in further detail. As shown in FIG. **3**, network device **104'** comprises, in part, a network data transfer service **302**, a service engine **304** that accesses local copies of DTD **202** and services **308**, and device software and hardware **306**. It should be apparent that network device **104'** can contain or include other components for performing conventional switching and routing functions, for example. However, the details of such additional components are not presented here so as not to obscure the present invention. Further, although DTD **202** and services **308** are shown as local storages, it should be apparent that such storage need not be permanent. For example, DTDs and services may be retrieved from a remote server via a URL, and just a temporary representation can be resident on device **104'** as needed by service engine **304** according to techniques well understood by those of skill in the art.

Device hardware and software **306** represents conventional switch or router functionality that has been adapted for use in the present invention. In one possible implementation, where network device **104'** is an Accelar/Passport family router switch from Nortel Networks, device hardware and software **306** includes an ASIC-based forwarding architecture, with switch ASICs comprising most of the device's switch fabric and handling most forwarding tasks among switch ports in accordance with resident forwarding rules. For such a device **104'**, device hardware and software **306** further includes a CPU and associated memory coupled to the switch fabric that runs the VxWorks real-time OS, and existing applications stored in memory and executed by the CPU that run as VxWorks "tasks" for monitoring and controlling and configuring the ASIC forwarding hardware via a switch-specific API. It should be apparent that other types of switches and routers may be used in accordance with the invention, and that other operating systems such as Linux, PSOS, Vertex and RMX may comprise the device's operating system.

Services in storage **308** preferably include applications that enhance the network management capabilities of the device **104'** above and beyond that which is possible with a conventional network device **104**. Such applications may include means for setting and reporting system variables that are not limited by pre-configured MIBs. Such applications may further include means for configuring the forwarding architecture so as to enable the device to filter network traffic containing packets generated from activities not essential to a company's business, such as browsing the Internet. Other

examples of services can include event loggers and monitors, means for establishing different levels of quality of service in packet forwarding decisions, and the like. Although the discussion below will center on services that are launched using a command design parameter in accordance with the invention, it should be noted that device **104'** may also include functionality for executing similar remotely downloaded, installed and managed services, which additional similar services may also be stored in storage **308**.

In one example of the invention, device hardware and software **306** also includes an Oplet Runtime Environment (ORE™, a trademark of Nortel Networks), which is a platform for secure downloading, installation, and safe execution of services on a network device such as a switch or router, the downloaded services (i.e. Oplets) being locally stored in services storage **308**. In such an example of the invention, the network data transfer service **302** and service engine **304** may actually be implemented as one or more services (i.e. Oplets) managed by the ORE™ (not necessarily having a command design parameter). The ORE™ is described in more detail in other publications, including publications posted at the website www.openetlab.org, and so will not be described in detail here so as not to obscure the invention. Although use of the invention in network devices equipped with an ORE™ is considered one preferred implementation, the invention is not so limited. Further, the functionalities provided by the ORE™ that are useful for the present invention will be apparent to those skilled in the art after being taught by the present specification and can be separately provided.

Network data transfer service **302** is, for example, an HTTP server such as one provided by Apache. This is because, in one example of the invention, the XML encoded requests and device responses (if any) are exchanged using the HTTP protocol. As is known, HTTP is an application-level protocol that is generic, stateless, and can be used for many tasks other than transferring hypertext, which is its most widely known use. In one example of the invention, the HTTP communications take place over TCP/IP connections. The most widely used HTTP methods for handling communications are GET, HEAD and POST. The service engine **304** is registered with the HTTP server (by port number, for example) so that when XML encoded requests according to the invention are received by service **302**, service engine **304** can be activated and provided with the XML encoded request. The HTTP server keeps handles for all received requests pointing to the requesting client's address (perhaps with a timeout), and when responses from service engine **304** are received along with the handle, the HTTP server provides the response back to the requesting client using HTTP methods.

It should be apparent that other techniques for sending XML files through the HTTP protocol could be used. Moreover, in another alternative of the invention, responses may be forwarded back to the requesting client in various presentation alternatives such as providing HTML pages for browser access.

Device hardware and software **306** is adapted in accordance with the invention to forward packets using the HTTP protocol and addressed to network device **104'** to the network data transfer service **302**, if an HTTP server is not already provided in the device. This can be done in many ways known to those skilled in the art, and may depend on the particular packet forwarding architecture of the device. For example, where packets addressed to the device are automatically forwarded to the device's CPU, the device's kernel packet handling need only be aware of, and be able to communicate packets with, the HTTP server.

Service engine **304** is generally responsible for receiving XML encoded documents, for parsing the documents to identify the requested service and any specific run-time parameters, for causing the device **104'** to perform the requested service in accordance with the common design pattern and, if appropriate, obtaining and formatting a response to the requesting client.

Service engine **304** is further illustrated in FIG. **4**. As shown, it includes a parser **402**, a service launcher **404**, device APIs **406**, response retriever **408** and response formatter **410**. Although shown separately for clarity of the invention, the different blocks shown in FIG. **4** can be implemented in various combinations either together or separately. Moreover, some or all of the functionalities may be partially or fully included as functionalities of the ORE™ in the example of the invention where the ORE™ is included in the network device **104'**.

Parser **402** receives the XML document from the network data transfer service **302** (preferably along with a handle for the individual request), and retrieves the necessary DTD based on the required identifier in the XML document. Using the appropriate DTD, parser **402** performs processing based on the XML tags defined in the DTD and extracts out the values for each included in the XML document. These parsed-out values are provided to the service launcher **404**. There are several different conventionally available XML parsers that can be used to implement parser **402**, such as Document Object Model (DOM), Simple API for XML (SAX) and Java Document Object Model (JDOM). In one example of the invention, parser **402** is implemented by an Aelfred XML parser from OpenText. The Aelfred parser generates SAX events for each parsed XML tag. These parsed-out values and SAX events are supplied to service launcher **404**.

At a minimum, the DTD and the XML document should at least specify one of the services **308** to be performed. For example, the DTD may include a definition such as:

```
<!ELEMENT service>
<!ATTLIST service
   class CDATA #IMPLIED
   source CDATA #IMPLIED
   id ID #IMPLIED>
```

This allows an XML document to specify a "service" having "class," "source" and "id" parameters. Accordingly, a client wishing to launch a service on a remote device would create an XML document specifying a "service" with at least a desired "class." Such a corresponding XML document may include the following text (as well as an identifier of the DTD that defines its structure):

```
<service class="Address" id="ID_2">
</service>
```

This XML document requests the network device **104'** to launch a "service" having a class of "Address." Accordingly, in this example of the invention, when the network device **104'** receives the XML document, parser **402** will retrieve the DTD indicated by the identifier in the document. Using this DTD, it will understand that a "service" having a class of "Address" should be launched. Accordingly, it will send a message to service launcher **404** to launch a service defined by the class "Address," which can, for example, cause the physical address of the device to be set or reported.

It should be noted that the source or byte code corresponding to the service "class" may be originally available locally on the device **104'** or it may be remotely located and specified by a URL or other path descriptor to a class file containing such source or byte code. In one example of the invention where the Java programming language is used, the "class"

identifier is a class whose byte code can be included in a Java Archive (JAR) file. If the byte code corresponding to the specified "class" is located with a URL, the service engine **304** (perhaps in cooperation with transfer service **302**) downloads the file for local access in storage **308**. Additionally or alternatively, the service engine **304** may check whether an object corresponding to the requested service has already been instantiated using the "id" parameter.

If the service includes any runtime parameters, the DTD and XML documents should specify those as well, although the service should be able to execute using default parameters. Depending on the service requested, there may or may not be a requested response to be sent back to the client. For example, one requested service may be to provide traffic statistics of the device, for which a response would be collected from the device hardware and software and returned to the client. Meanwhile, another requested service may be to adjust priorities of certain traffic flows, for which a response from the device hardware and software would not necessarily be requested. As an example of a requested service with runtime parameters, a requested service may be to report on device throughput, collected a variable number of seconds apart, with the report repeatedly provided back to the client once per variable number of minutes.

Using the above example of the service of class "Address," the DTD may further include a definition such as:

```
<!ELEMENT property (value|null)*>
<!ATTLIST property
   name CDATA #REQUIRED>
<!ELEMENT value (#PCDATA)*>
<!ATTLIST value
   class CDATA #IMPLIED
   id ID #IMPLIED
   source IDREF #IMPLIED>
```

This allows a "property" with a "name" to be assigned a "value." A corresponding XML document may then be created by the requesting client that contains the following text:

```
<property name="city">
   <value id="ID_3">San Francisco</value>
</property>
<property name="country">
   <value id="ID_4">USA</value>
</property>
```

When these definitions and this text are combined with the previously-described XML-encoded request, the combined XML text could, for example, cause a service of class "Address" to set a city field and a country field in the device address system variables to San Francisco and USA, respectively.

In one example of the invention where the source code corresponding to a service is provided as Java classes, service launcher **404** includes a Java Virtual Machine (JVM) that is ported to the device **104'** and operates as a task on the device CPU under an operating system such as VxWorks. The JVM receives the byte code corresponding to the Java class from storage **308** and instantiates an object corresponding to the service using a no arguments constructor. Alternatively, service launcher **404** identifies an already-instantiated object corresponding to the service using an "id" parameter or the like. Further, during or before instantiating the object, service engine **404** may detect that certain other class files are needed and operate to download or access them as well.

Once instantiated, service engine **404** sets properties in the object using any parameters also provided in the XML document and parsed out by parser **402**. For example, for each property "name" there may be a corresponding "set" method (e.g. for the "city" property, there is a "setCity" method), which method the service engine **404** calls using the JVM to set the property to the "value" parsed from the XML document.

According to an aspect of the invention, all services do not continually run on the device **104'**. Rather, individual services are launched as requested by clients so as to perform functionality when needed. Moreover, services according to the invention are designed to be executed using a command design parameter so that the service engine **404** need not be aware of the internal code used to implement the service.

Therefore, all services designed in accordance with the invention include an "operate" method which serves as the command design parameter. In the example of the invention where services are implemented using the Java programming language, this can be done by requiring all services to be based on, or to "implement" a standard interface class. For example, each service class may implement an interface class defined as:

```
public interface Operation
{
   public Object operate( );
}
```

Each service class thus provides its own implementation of the "operate" method of the Operation interface. After instantiating the service class (perhaps also after "setting" various properties of the service object using parameters from the parsed document), service launcher **404** calls this specific "operate" method. Accordingly, service launcher **404** can cause various types of services to be performed without needing to know their implementation. This is akin to putting the code needed to process the requested service in the request itself.

Device APIs **406** includes functionality to interact with device hardware and software **306** to perform the requested service and to receive any responses from the device's device hardware and software. For example, where a service requests a network parameter of the device such as the device's name, the device APIs **406** will interact with the device hardware and software to retrieve the name (e.g. a string) from the device's system variables (e.g. MIB) and provide it to response retriever **408**, along with an identifier of the service that requested the parameter.

It should be appreciated that the actual implementation of device APIs **406** depends on the code used to implement services **308**, as well as the device hardware and software. In one example of the invention, all services **308** use a common code language such as C/C++. In such an example, device APIs **406** comprises APIs that provide a common interface for all such services to the existing code running on the device **104'**. Accordingly, services **308** can be designed to execute on various platforms, with a known set of APIs providing a common interface to the services, while providing a variable interface with the existing code, depending on the device. The design and implementation of such APIs are within the skill of those in the art given the existing code, the device operating system and the design of services **308**.

In another example, device APIs **406** communicate with existing conventional applications of network device **104** through a loopback address of the device. For example, the service requested by the received XIVIL document may request access to network parameters of the device. Specifically, the requested service launched by the service engine can access the network parameters of the device by specifying the loopback address, which can then access the parameters through a network protocol stack such as an SNMP stack.

It should be apparent that the above two examples are not necessarily mutually exclusive. Further, other example

implementations of device APIs **406** are possible. Moreover, it should be apparent that some services **308** need not require APIs to fully operate.

Response retriever **408** keeps track of the services that require responses from the device hardware and software and initiates response messages to the requesting client when responses are received. It receives from service launcher **404** identifiers of the services that have been launched, as well as handles to the XML encoded request that caused the service to be launched. When responses are received from device APIs **406**, they are preferably received along with the identifier of the service. Response retriever **408** can then correlate the response to the XML encoded request to whom the response belongs and forward the response to response formatter **410** along with the handle to the XML encoded request.

Response formatter **410** formats response messages to be sent back to the requesting client. It receives from response retriever **408** a response along with an identifier of the XML encoded request, formats a response message for transmission back to the requesting client, and forwards the response message to the network data transfer service **302**. Network data transfer service **302** can then send the message back to the requesting client by using the handle of the XML encoded request to retrieve the header information contained in the packets carrying the XML encoded request. In one example of the invention, responses are also XML encoded documents that will instantiate a response object on the client. In this example, response formatter **410** may also access DTDs such as DTD **202** for formatting a response. However, it should be apparent that many other variations of formatting a response other than using XML documents are possible.

It should be further apparent that there are many possible ways of implementing response retriever **408** and response formatter **410**, and that they may be omitted altogether. For example, response retriever **408** and/or response formatter **410** may be partially or fully implemented by either or both of services **308** and device APIs **406**.

FIG. **5** is an architectural view of an example of network device **104'** in accordance with the principles of the present invention.

As shown in this example, interaction with the device hardware **514** (e.g. switch fabric, ports, memory, etc.) is performed through the device operating system **512** (e.g. VxWorks). The device code (e.g. routing software, etc.) **502** interacts with the device operating system **512**. Application programming interfaces (API's) **504** (e.g. Java, C/C++, etc.) interact directly with the device hardware **514** and/or via device operating system **512**. API's **504** may further interact with device hardware and operating system through device drivers (not shown). Java Virtual Machine (JVM) **508** preferably includes all functionality provided by a conventional JVM and is ported to the device **104'** and operating system **512**. Oplet Runtime Environment™ (ORE) **506** interacts with the JVM to coordinate the downloading, management and execution of services **308**. Service engine **304** interacts with ORE **506** for execution of services **308**. Service engine **304** and transfer service **302**, during operation, may also interact with API's **504**, which further interact with device code **502**.

FIG. **6** illustrates an example implementation of a network device **104'** having the architecture illustrated in FIG. **5**.

As shown, network device **104'** includes a CPU **602**, switch fabric **604**, storage **608**, network port **610** and memory **612** all communicating via a bus **614**. Switch fabric **604** further communicates with switch ports **606**. Storage **608** can comprise memory for storing program and device data. Network port **610** can provide a loopback address for access by services and other network management applications as described above. Memory **612** includes code for execution by CPU **602**.

It should be apparent that components corresponding to CPU **602**, switch fabric **604**, switch ports **606**, storage **608**, network port **610**, memory **612** and bus **614** are also provided in a conventional network device **104**. Accordingly, as should be further apparent from FIG. **6**, adapting a conventional network device **104** in accordance with the invention merely requires updating memory **612** to include executable software corresponding to the above-described functionality of the invention.

FIG. **7** illustrates an example of a process by which an XML encoded request received by the network device **104'** is fulfilled in accordance with the present invention.

As shown in FIG. **7**, when interaction with the network device **104'** is desired, the client computer **100'** encodes the request by constructing an XML encoded document corresponding to the request and in accordance with a corresponding DTD. This XML document is sent across the network **102** using a standard network protocol such as HTTP and received by network device **104'** (block S**702**). The XML document is received by the network data transfer service **302** running on the network device **104'** and provided to the service engine **304**. The service engine **304** parses the XML document using the corresponding DTD identified in the document (block S**704**). The parsed document corresponds to one of the services provided or retrieved for local access in storage **308**. Accordingly, service engine **304** instantiates a copy (or identifies an already-instantiated copy) of the service with the properties specified in in the parsed document (block S**706**). The instantiated service is then launched for execution by invoking the "operate" method of the service or similar command design parameter (block S**708**). The requested service may require interaction with the device software and hardware for execution, as indicated in block S**710**. If a response message back to the requesting client **100'** is required by the service (determined in block S**712**), the response from the device hardware and/or software is retrieved (block S**714**), and a response message is formatted and forwarded to network data transfer service **302** for transmission back to client **100'** (block S**716**).

Although the present invention has been particularly described with reference to the preferred embodiments, it should be readily apparent to those of ordinary skill in the art that changes and modifications in the form and details may be made without departing from the spirit and scope of the invention. It is intended that the appended claims include such changes and modifications.

What is claimed is:

1. A network device for locally performing a service, comprising:

a parser for receiving a service request from a requesting client, the service request being in a first format and including a markup language document and a request identifier, for retrieving a document type definition identified by the markup language document, and for extracting parameter values from and generating events for each tag in the markup language document defined in the document type definition, wherein the markup language document and document type definition specify a requested service using class, source, and identifier parameter values, wherein the identifier parameter value indicates the document type definition and wherein the class parameter value indicates a service class;

a service launcher for receiving the parameter values and events from the parser, determining whether an object corresponding to the requested service has already been

instantiated, receiving the service class from storage, and, in response to determining that the requested service has not already been instantiated, instantiating an object corresponding to the service based on the service class, and for calling an operate method of the instantiated object to launch the requested service, wherein the operate method is also implemented by each of a plurality of other service classes;

a response retriever for keeping track of services that require responses, receiving at least one response, and for correlating the at least one response to the requesting client; and

a response formatter for receiving the at least one response from the response retriever and formatting a response message based on the at least one response for transmission back to the requesting client, the response message being in a second format, different from the first format.

2. The network device according to claim **1**, wherein the requested service comprises an HTTP server.

3. The network device according to claim **1**, wherein the markup language comprises XML.

4. The network device according to claim **3**, wherein the document type definition comprises an XML DTD.

5. The network device according to claim **1**, wherein the service launcher identifies the byte code for the requested service from a plurality of classes responsive to the class parameter value.

6. The network device according to claim **1**, wherein the requested service configures a packet forwarding architecture in the network device to filter network traffic.

7. The network device according to claim **6**, wherein the packet forwarding architecture comprises a packet forwarding switch fabric.

8. The network device according to claim **7**, wherein the requested service causes changes in how packets are forwarded through the packet forwarding switch fabric.

9. The network device according to claim **7**, wherein the requested service monitors performance indicators of how packets are forwarded through the packet forwarding switch fabric.

10. The network device according to claim **1**, wherein the requested service accesses a MIB on the network device.

11. A method for causing a network device to locally perform a service, comprising:

receiving, by a parser from a requesting client, a service request being in a first format and including a markup language document and a request identifier;

retrieving, by the parser, a document type definition identified by the markup language document;

extracting, by the parser, parameter values from and generating events for each tag in the markup language document defined in the document type definition, wherein the markup language document and document type definition specify a requested service using class, source, and identifier parameter values, wherein the identifier parameter value indicates the document type definition and wherein the class parameter value indicates a service class;

receiving, by a service launcher from the parser, parameter values and events;

determining, by the service launcher, whether an object corresponding to the requested service has already been instantiated;

receiving, by the service launcher from storage, the service class;

instantiating, by the service launcher in response to determining that the requested service has not already been instantiated, an object corresponding to the service based on the service class;

calling, by the service launcher, an operate method of the instantiated object to launch the requested service, wherein the operate method is also implemented by each of a plurality of other service classes;

keeping track, by a response retriever, of services that require responses;

receiving, by the response retriever, at least one response;

correlating, by the response retriever, the at least one response to the requesting client;

receiving, by a response formatter, the at least one response from the response retriever; and

formatting, by the response formatter, a response message based on the at least one response, for transmission back to the requesting client, the response message being in a second format, different from the first format.

12. The method of claim **11**, wherein the requested service comprises an HTTP server.

13. The method of claim **11**, wherein the markup language comprises XML.

14. The method of claim **13**, wherein the document type definition comprises an XML DTD.

15. The method of claim **11**, further comprising identifying, by the service launcher, the byte code for the requested service from a plurality of classes responsive to the class parameter value.

16. The method of claim **11**, wherein the requested service configures a packet forwarding architecture in the network device to filter network traffic.

17. The method of claim **16**, wherein the packet forwarding architecture comprises a packet forwarding switch fabric.

18. The method of claim **17**, wherein the requested service causes changes in how packets are forwarded through the packet forwarding switch fabric.

19. The method of claim **17**, wherein the requested service monitors performance indicators of how packets are forwarded through the packet forwarding switch fabric.

20. The method of claim **11**, wherein the requested service accesses a MIB on the network device.

21. A network device for locally performing a service, comprising:

a parser for receiving a service request from a requesting client, the service request including a markup language document and a request identifier, for retrieving a document type definition identified by the markup language document, and for extracting parameter values from and generating events for each tag in the markup language document defined in the document type definition, wherein the markup language document and document type definition specify a requested service using class, source, and identifier parameter values, wherein the identifier parameter value indicates the document type definition and wherein the class parameter value indicates a service class;

a service launcher for receiving the parameter values and events from the parser, determining whether an object corresponding to the requested service has already been instantiated, receiving the service class from storage, and, in response to determining that the requested service has not already been instantiated, instantiating an object corresponding to the service based on the service class, and for calling an operate method of the instanti-

ated object to launch the requested service, wherein the operate method is also implemented by each of a plurality of other service classes;

a response retriever for keeping track of services that require responses, wherein, for a service requiring multiple responses, receiving multiple responses for the service and for correlating the multiple responses to the requesting client; and

a response formatter for receiving the at least one response from the response retriever and formatting a response message based on the at least one response for transmission back to the requesting client.

* * * * *