Innovation, Sciences et Développement économique Canada
Office de la Propriété Intellectuelle du Canada

Innovation, Science and Economic Development Canada
Canadian Intellectual Property Office

CA 3062745 C 2023/09/05

(11)(21) 3 062 745

(12) BREVET CANADIEN
CANADIAN PATENT

(13) C

(86) Date de dépôt PCT/PCT Filing Date: 2017/11/20
(87) Date publication PCT/PCT Publication Date: 2018/06/07
(45) Date de délivrance/Issue Date: 2023/09/05
(85) Entrée phase nationale/National Entry: 2019/10/25
(86) N° demande PCT/PCT Application No.: IB 2017/001770
(87) N° publication PCT/PCT Publication No.: 2018/100435
(30) Priorité/Priority: 2016/11/20 (US62/424,489)

(51) Cl.Int./Int.Cl. *H04L 65/1066* (2022.01), *G06F 17/00* (2019.01), *H04L 12/12* (2006.01), *H04L 65/401* (2022.01), *H04L 67/10* (2022.01), *H04L 69/16* (2022.01), *H04L 67/02* (2022.01)

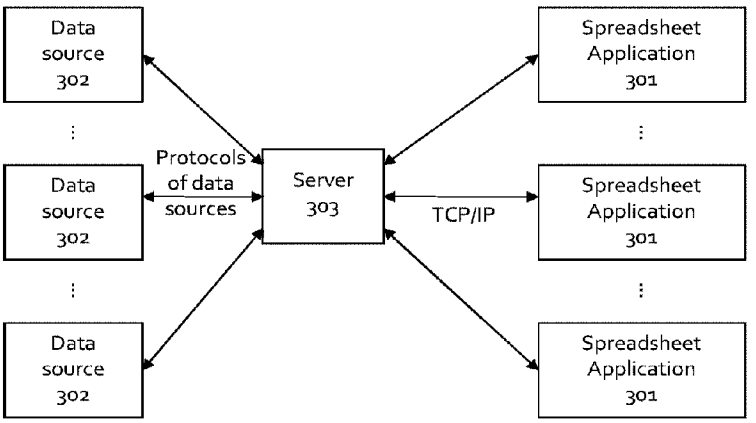
(72) Inventeur/Inventor:
THOMAS, ANDREW, CA

(73) Propriétaire/Owner:
REAL INNOVATIONS INTERNATIONAL LLC, CA

(74) Agent: THOMAS, PAUL E.

(54) Titre : ECHANGE BIDIRECTIONNEL DE DONNEES EN TEMPS REEL DANS UN RESEAU A L'AIDE D'UNE APPLICATION DE TABLEUR

(54) Title: BIDIRECTIONAL NETWORKED REAL-TIME DATA EXCHANGE USING A SPREADSHEET APPLICATION



(57) Abrégé/Abstract:
Systems and methods and methods for providing real-time data to a spreadsheet applications (SSAPPs) are disclosed. In an example, a spreadsheet application (SSAPP) obtains subscribed data from a server through the persistent connection between the SSAPP and the server via a TCP socket. The subscribed data can be propagated to the server from a data source. The SSAPP can perform an action on the subscribed data, such as presenting a representation based on the subscribed data to a user. When the data source propagates updated data to the server, the server can send the updates to the SSAPP in real time over the TCP socket. The SSAPP can also send data do the server over the TCP socked by, for example, extracting contents from a set of cells, processing the contents to produce a result, and transmitting the result to the server via the persistent connection.

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
07 June 2018 (07.06.2018)

WIPO | PCT

(10) International Publication Number
WO 2018/100435 A3

(51) International Patent Classification:
H04L 29/06 (2006.01) *H04L 12/12* (2006.01)
G06F 17/00 (2006.01)

(21) International Application Number:
PCT/IB2017/001770

(22) International Filing Date:
20 November 2017 (20.11.2017)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
62/424,489 20 November 2016 (20.11.2016) US

(71) Applicant: **REAL INNOVATIONS INTERNATIONAL LLC** [CA/CA]; 162 Guelph Street, Suite 253, Georgetown, ON L7G 5X7 (CA).

(72) Inventor: **THOMAS, Andrew**; 162 Guelph Street, Suite 253, Georgetown, ON L7G 5X7 (CA).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report (Art. 21(3))*
— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))*

(54) Title: BIDIRECTIONAL NETWORKED REAL-TIME DATA EXCHANGE USING A SPREADSHEET APPLICATION

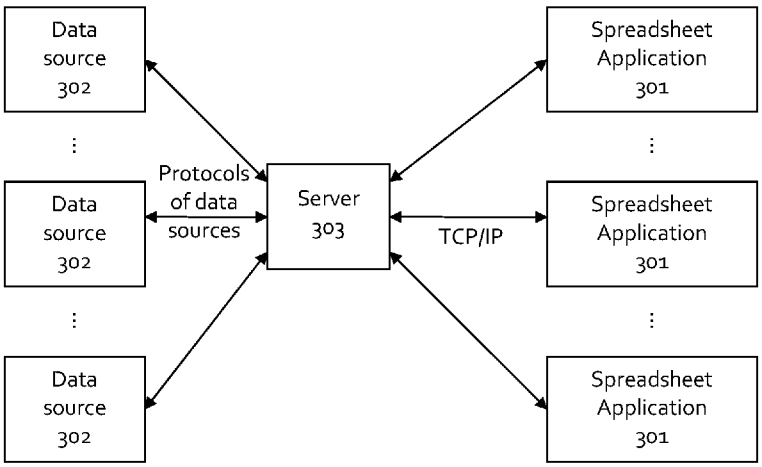


FIG. 3

(57) Abstract: Systems and methods and methods for providing real-time data to a spreadsheet applications (SSAPPs) are disclosed. In an example, a spreadsheet application (SSAPP) obtains subscribed data from a server through the persistent connection between the SSAPP and the server via a TCP socket. The subscribed data can be propagated to the server from a data source. The SSAPP can perform an action on the subscribed data, such as presenting a representation based on the subscribed data to a user. When the data source propagates updated data to the server, the server can send the updates to the SSAPP in real time over the TCP socket. The SSAPP can also send data do the server over the TCP socked by, for example, extracting contents from a set of cells, processing the contents to produce a result, and transmitting the result to the server via the persistent connection.

WO 2018/100435 A3

[Continued on next page]

WO 2018/100435 A3

(88) Date of publication of the international search report:
27 September 2018 (27.09.2018)

BIDIRECTIONAL NETWORKED REAL-TIME DATA EXCHANGE USING A SPREADSHEET APPLICATION

Copyright Notice

Cross-Reference to Related Applications

[0002] This application claims the benefit of U.S. Provisional Application No. 62/424,489, filed on November 20, 2016.

[0003] This application is related to Applicant's U.S. Patent Application No. 15/497,466, which was filed on April 26, 2017, which claims priority to U.S. Patent No. 9,667,689, which was filed on January 6, 2014, and which claims priority to U.S. Patent No. 8,661,092, which was filed on October 15, 2010, and which claims priority to U.S. Provisional Application No. 61/252,624, filed on October 16, 2009.

Background of the Invention

[0004] Real-time data refers to any digital or analog information that should be processed and/or transmitted within a certain amount of time after that data is originally created. The time elapsed from the moment that the data is created until it is processed and/or transmitted is known as latency. The maximum latency allowable for any particular real-time application is application-dependent. Applications where the maximum latency is a strict requirement can be referred to as "hard" real-time applications, while applications where the maximum latency is not a strict requirement can be referred to as "soft" real-time applications. Soft real-time applications need only

Date Reçue/Date Received 2021-11-19

satisfy an application-dependent, often subjective, measure of "fast enough". Non-real-time data is data that is not required to satisfy any particular latency requirement.

[0005] The term "data" may refer to hard real-time, soft real-time or non-real-time data. "Real-time data" may refer to hard real-time or soft real-time data.

[0006] Real-time data is typically generated due to a physical process or a computer program external to the computer system that processed the data. For example, real-time data may include: information from an industrial process control system such as motor status, fluid tank level, valve position, conveyor speed and so on; prices, volumes, etc. for financial instruments such as stocks; user interface events such as an indication that a user has clicked on a button on a computer display; data entry by a human operator; and computer operating system status changes. Virtually any information that is changing over time can be treated as real-time data.

[0007] An originator of data may be described as a "data source". For example, data may originate as a physical process, measured electrically, and converted to a digital representation, or data may originate in a digital representation. Generally, data is made available in a digital computer as a digital representation, following zero or more steps to convert the data into a digital representation. A data source may comprise all of the components and steps necessary to convert the data to a digital form accessible by a computer program.

[0008] Analogous to a data source is a "data sink". A data sink consumes, or uses, data. Some examples of data sinks are: actuators in a process control system; trade processing software in a stock trading system; a user interface application; a database or other data storage system.

[0009] Many data sources are also data sinks. Accordingly, a data source may comprise a data source, a data sink, or both simultaneously. For example, when data is transmitted to a data source, the data source may also act as a data sink.

[0010] In computer applications, data is commonly managed by a "server". The server can act as either a data source or a data sink, or both together, allowing "client" applications to interact with the data that the server manages.

[0011] Generally, a client application must initiate a connection with a server in order to interact with data. That connection can be "short-lived", where the connection exists only for the duration of a single or few interactions with the data, or "long-lived", where the connection persists for many interactions with the data, and possibly for the duration of the client application's lifetime. Long-lived connections are also referred to as "persistent" connections.

[0012] Data sources provide data in one or more "data formats" that define the digital representation of the data. The data format may conform to a published standard or be particular to the data source. Similarly, data sinks may require data in a published standard format or in a format particular to the data sink.

[0013] Data sources provide access to data through one or more "transmission protocols". A transmission protocol specifies the mechanism by which data are transferred from a data source to a data sink. A transmission protocol may conform to a published standard or be particular to the data source. A data source may combine data formats and transmission protocols such that not all supported data formats can be transmitted via all supported transmission protocols. Generally, a "protocol" or "data protocol" refers to the combination of a particular data format transmitted via a particular transmission protocol.

[0014] A data sink must support at least one data protocol offered by a data source in order to use the data generated by the data source. Since a large number of data protocols exist, it is impractical for all data sources and data sinks to support all data protocols. As a result, client applications that make use of data are usually created only to support the most necessary protocols for their primary purpose. Similarly, data sources generally support only those protocols that are necessary for their primary purpose. So, for example, there is no way to directly connect a web browser that supports the HTTP protocol to a spreadsheet application that supports the DDE protocol.

[0015] A protocol conversion step must be performed to convert data from a protocol supported by a data source into a protocol supported by a data sink in order for the data sink to make use of the data offered by the data source. This conversion step can be performed by a "middleware" application. A primary purpose of a middleware

application may be to facilitate communication between a data source and a data sink, usually by converting data from one protocol to another such that data sources and data sinks can interact indirectly when they share no protocol in common.

[0016] A data source may transfer data to a data sink using at least two methods:

[0017] 1. On demand: the data source passively waits for a data sink to request some or all of the data available in the data source. When the data sink makes a request for data, the source responds with a result indicating the current state of the requested data. If the data sink needs to be informed of changes to the data, the data sink must repeat the request in order for the data source to respond with the updated data. This repeated request for the same data by the data sink is known as "polling". A data sink may create either a short-lived connection to the data source for each new request, or a persistent connection over which many repeated requests are transmitted.

[0018] 2. By subscription: the data sink creates a persistent connection to the data source, and subscribes to some or all of the data available from the data source. The data source transmits any changes to the data via the persistent connection as those changes occur. The data source will continue to send changes to the data until the data sink specifies otherwise or the connection is closed.

[0019] It is understood that data transfer methods such as shared memory, message queues and mailboxes are variations on either the demand or subscription methods. It is also understood that the terms data transfer, data propagation, or data transmission all refer to the movement of data within a system, and these terms may be used interchangeably, as they relate to the specific data transfer method. It is further understood that these methods are independent of the underlying transmission protocol.

[0020] Computer applications dealing with real-time data must be reliable, responsive and easily connected to their data sources. This has meant that real-time data processing applications have historically been created as stand-alone applications connected directly or indirectly to the data source. This stand-alone architecture has also allowed the applications to take full advantage of the graphical capabilities of the computer to provide rich dynamic visualization of the real-time data. By contrast, applications based on web browser technology have proven unsuitable in terms of data

connectivity and graphical speed. The HTTP protocol is intended as a request-response communication method where each request-response pair requires a web client (typically a web browser) to open a new socket to a web server, perform the communication and then shut down the socket. This paradigm works well for communication that is infrequent and not particularly time-sensitive. The HTTP protocol further limits the types of transactions to data retrieval from the web server or data submission to the web server, but not both in the same transaction. Methodologies such as AJAX that are based on this model are expected to make relatively few transactions and tend to scale to higher speeds very poorly. The computational and networking costs of establishing and closing connections for each transaction act as a limit to the speed of such systems.

[0021] Consequently, widespread real-time data processing, as well as display in a web browser, has been unavailable. Some developer efforts have provided access to data driven displays using ActiveX components in a web browser, but these components are generally poorly supported by modern browsers and subject to limitations due to the security risks that they represent.

[0022] Efforts have been made to display changing data in a web browser using the built-in Javascript engine of the browser. This is generally achieved using a methodology called AJAX (Asynchronous Javascript and XML), where the web browser polls periodically for new data and then updates its display accordingly. This polling mechanism is highly inefficient, and suitable only for relatively small data sets or for relatively slow-moving data. Lowering the polling rate to conserve CPU or network bandwidth has the effect of raising data latency, which is unacceptable for real-time applications.

[0023] Efforts to improve on AJAX, through a mechanism called Streaming AJAX, take advantage of a side-effect of the browser page loading mechanism to cause a browser page to grow incrementally by adding Javascript commands to the page over time. Each Javascript command executes as it arrives, giving the impression of a continuous data stream. The web browser is effectively fooled into thinking that it is loading a very large web page over a slow network connection. This method has several drawbacks, including the fact that the web browser's memory and CPU usage can grow continuously over time due to the ever-larger page that is being transmitted. Holding an

HTTP connection open to collect multiple asynchronous messages from a specially designed web server like this effectively makes the short-lived HTTP connection into a long-lived streaming connection. This allows much faster updates from the server to the client, as new data can be transmitted from the server asynchronously and does not require the client to open and close a connection for each new message. However, it does nothing to speed up the communication from the client to the server. Effectively it creates a fast uni-directional channel from the server to the client, while still retaining the negative performance characteristics of HTTP when communicating from the client to the server.

[0024] Both AJAX and streaming AJAX methods suffer from a lack of quality display options within the web browser. Web browsers are generally designed for the display of static pages and web "forms", and do not offer high-speed or high quality graphic presentation options. Efforts to improve upon graphical display options have tended to be incompatible among web browsers, and generally very slow to execute.

[0025] All data transmission solutions based on built-in web browser capability are primarily targeted at receiving data in the web browser. The communication of data is uni-directional, in that the connection that receives data from a server cannot also be used to transmit data to the server. If the web browser needs to transmit data back to the server, it must do so by opening a new connection, transmitting an HTTP request, and then closing the connection again. Consequently, solutions such as Streaming AJAX are very slow to transmit data back to the data server, because of the large overheads and latencies incurred by having to emit a new HTTP request for every data transmission.

[0026] Some efforts at web-based data visualization attempt to improve the user experience by presenting slow-moving (high latency) data as if it were faster. This is achieved by displaying interpolated data in the web browser at higher frequency than the data is actually arriving. For example, a circular gauge representing a speedometer might receive the values 1 and 100, separated in time by 5 seconds. The web page could then draw the gauge dial 5 times per second, changing the value by 4 each time. This would give the viewer an impression of a smoothly changing speed, even though the underlying data delivery contains no such information. That is, such a display of interpolated data can be entirely misleading to the viewer. This kind of interpolation obscures the true

behaviour of the underlying data, and is usually unacceptable in real-time applications such as process control and stock-market trading.

[0027] Rich Internet Application ("RIA") frameworks such as Adobe Flash™ and Microsoft Silverlight™ offer improved platforms for both data processing and graphical display within a web browser. These RIA frameworks also support direct TCP/IP communications within the RIA. Surprisingly, the combination of these features makes it possible to process and display real-time information in a web browser. This processing and display capability has not been translated into real-time data systems due to a perception in the software industry that RIAs are suited primarily to video, advertising and games.

[0028] A common alternative to HTTP is to provide a secondary communication socket for high-speed data alongside the HTTP communication channel. Effectively, the web client communicates via HTTP for the presentation information, and via a separate dedicated socket for high-speed bi-directional data communication. This solves the speed issue, but introduces other issues:

[0029] A separate communication socket requires a separate TCP port to be open on the server. This means another opening in the corporate firewall, which IT departments commonly resist.

[0030] Rich Internet Application (RIA) frameworks, such as Flash or Silverlight, commonly implement limits on socket communication that require yet another well-known port to be open to act as an access policy server. This introduces a further opening in the corporate firewall, further limiting the usefulness of the technique.

[0031] An RIA framework operating within a browser (e.g., Silverlight) may not implement its own SSL layer, relying instead on the browser's HTTPS implementation for encryption. In such a case, a dedicated socket implemented by an RIA will not be secure.

[0032] - Dedicated sockets will not pass through web proxies.

[0033] The advent of high-speed or real-time data processing over the Internet has created a need for long-lived high-speed socket communication. This need has driven the RIA implementers to offer such sockets, but with the limitations described above. There remains an unmet need for long-lived bi-directional socket communication over HTTP or, more preferably, HTTPS to a web server.

[0034] The HTML5 specification includes a draft specification called WebSockets. This intends to provide two-way communication between a client and server using a HTTP-mediated socket. Although WebSockets are not universally supported at this time, they provide the possibility of creating bi-directional connections through forward and reverse web proxies. The current invention enables real-time data connectivity through WebSockets, providing successful connectivity even in instances where the data source or end user are isolated from the Internet via proxy servers and are unable to make a connection via an arbitrary TCP/IP port. This significantly broadens the set of network topologies on which the current invention may be usefully implemented while allowing an additional potential level of security on the client networks.

[0035] The present invention is suitable to augment industrial Supervisory Control And Data Acquisition ("SCADA") systems. SCADA systems comprise data collection hardware such as sensors and other devices, communication networks, central processing systems, and display units to allow plant operators and engineers to view the data in their industrial processes in real time. SCADA systems often comprise interfaces that supports a supervisory level of coordination and control, such as uploading new recipes to a candy-making machine, changing global settings on a wind turbine, or acknowledging a high pressure alarm for a boiler.

[0036] SCADA systems have evolved over time. The first generation systems were "monolithic", running on individual computers, connecting to field devices directly. The second generation allowed "distributed" processing, using multiple computers communicating with each other over a local-area network ("LAN") and communicating with the field devices over proprietary control networks. The current, "networked", generation uses personal computers and open standards such as TCP/IP and open protocols for local-area networking. Thus it is now possible to access SCADA systems

and data from the Internet, although there are fundamental questions about security that are limiting the broad adoption of such capabilities.

[0037] Networked SCADA systems are designed using a client/server model. A server (device or software application) contains a collection of data items. These data items are made available to a client (device or software application) upon request by the client. The implicit assumption is that the server is the authoritative source of the data values, and has a-priori knowledge of which data values it will supply. The client is non-authoritative, and determines which data items it may use by querying the server. For clarity, the authoritative source of data has the responsibility to determine which data items it will contain and make available to its clients, and the data values held in the authoritative source are presumed to be correct and current. The client cannot determine which data items exist, and may only affect the values and/or properties of the data items defined within the server.

[0038] Importantly, the server is simultaneously the authoritative data source and also a listener for incoming connections from the client. In a networked system, this means that any client that uses the data must be able to initiate a connection to the server. In a SCADA system, this would mean, for example, that an operator workstation (acting as a client) must be able to make a connection to the SCADA server. This in turn requires that the SCADA server be reachable via the network from the client's location. In the case of an Internet-based or cloud-based system, this means that the SCADA server must be reachable from the Internet, posing an unacceptable security risk. For clarity, the terms "cloud" and "Internet" may be used interchangeably throughout this disclosure.

[0039] When the topic of cloud computing is raised among process control engineers, there are many justifiable concerns about security. SCADA and other manufacturing and control systems often support high-value production lines, where any interference or foul play could cost thousands or millions of dollars. Although recently some shop floors have begun to make their process data available to the rest of the company on corporate LANs, there is strong resistance to opening ports in plant firewalls to allow incoming connections from the Internet.

[0040] On the other hand, cloud systems generally require Internet access, typically using a web browser HMI ("Human Machine Interface") or RIA or other kind of client to connect to a server on the process side. Until the present invention, this meant that a port had to be opened in the factory firewall to allow the web browser to connect. And this is a security risk that few plant engineers are willing to take. The primary source of security exploits is firewalls permitting inbound connections. Unless these are removed, the plant is exposed to attack.

[0041] Due to the mission-critical nature of SCADA systems, engineers and managers responsible for industrial processes are reluctant to expose them directly to the Internet, running behind secure firewalls to keep intruders and hackers at bay. Compounding the problem is that the architecture of most installed industrial systems was not developed with the Internet in mind.

[0042] Spreadsheet applications are computer software applications commonly used to analyze and generate information. A spreadsheet application ("SSAPP") presents data as a multi-dimensional table of data, where each data item is presented as a cell within that table. Each cell can contain a value or a formula that references other cells to produce a computation from their values. A cell may also contain formatting that determines how that cell is displayed to a user, where the formatting could also be a formula that computes the formatting based on the result of a computation. Examples of a spreadsheet application include Microsoft Excel™ (Microsoft Corp.), Google Sheets™ (Google Inc.) and Open Office™ Calc (Apache Software Foundation). SSAPPs are typically used in scenarios where decision-making and analytical logic is encoded in the cells and then presented to a user. Users can then modify certain cell values to have others recalculate, thereby updating the content of the spreadsheet to reflect the newly entered information. SSAPPs are frequently used to produce dashboards of non-real-time information. In some environments, SSAPPs are used to collect real-time information to be used as part of the cell computations (e.g., stock trading applications).

[0043] Spreadsheet applications commonly provide mechanisms to share information among spreadsheets and other applications running on a single computer. For example, Excel has two inter-process communication mechanisms: DDE (Dynamic Data Exchange) and RTD (Real-Time Data). These communication mechanisms

provide a means to analyse data originating from real-time data sources such as SCADA systems. However, these communication mechanisms have limitation, where, for example, DDE is a simple data exchange protocols based on Windows messages. This allows Excel to share information in real time with other applications using a simple (tag, value) representation. DDE is a client/server architecture where the server transmits data to the client based on subscriptions configured by the client. In the context of data from SCADA and/or networked real-time systems, this architecture suffers from a number of limitations, both by design and, in particular, by implementation in Excel, for example:

1. DDE is not a networked protocol. Excel and the communicating application must run on the same computer.

2. DDE does not transmit time stamp, data quality or data type information. This limits the data's usefulness when interacting with time sensitive data, with data whose type is not known a-priori and data where quality information is important.

3. DDE bindings in Excel consume the formula of the cell(s) into which the DDE data is bound. This exposes the DDE binding to accidental deletion. In addition, it makes it impossible to bind the same cell both to receive data and to transmit it. The act of modifying the cell value deletes the DDE binding.

4. Since a DDE binding consumes the cell formula, any Excel cell can be a participant in at most one DDE binding. That is, a cell cannot be updated from multiple sources.

5. When Excel is acting as a DDE server, modifying any DDE-bound cell in an Excel spreadsheet causes all DDE bindings on the sheet to re-transmit their values even if they have not changed. This may not be important when communicating with applications on the same computer, but it would generate unacceptable network bandwidth utilization if DDE were extended to a network.

6. DDE values are transmitted by subscription, meaning that for Excel to emit data to another application that application must have subscribed to a cell or cells in Excel. When the cell content changes (a data change event), Excel emits the

new cell value. This means that configuration for the data communication must happen independently in two locations: in Excel for incoming data, and in the receiving application for outgoing data. Although Excel provides a means via scripting to "push" data via DDE to an application, this is challenging for a user to configure and produces "blocking" behaviour in Excel, which is highly undesirable.

7. There is no mechanism in Excel's implementation of DDE to reconnect to an application if the connection is lost. This produces a system that is not robust. The start-up order of Excel and the other application is crucial, resulting in a fragile system.

8. There is no mechanism in Excel to re-try a subscription for a DDE binding that did not exist at the time that the binding was originally attempted. When Excel starts it may attempt to subscribe to DDE items from another application. If the other application is not yet configured to satisfy that request, Excel will never re-try the request. The result is that some, but not all, of the DDE bindings in the Excel spreadsheet may be "dead".

9. DDE contains no data inspection mechanism. A DDE client cannot determine which tags are available in the DDE server.

10. DDE can present an attack vector for malware, so there are risks associated with leaving DDE features enabled. Disabling DDE features can mitigate the risk, but doing so can break functionality and prevent data from updating.

[0044] More recently, Excel has been modified to support RTD, a communication protocol based on COM (Component Object Model). Networking is supported in RTD using DCOM (Distributed COM). RTD is a client/server architecture where the client makes "read" calls to the server when it wants new data. Data is not transmitted by subscription. It uses a (tag, value) representation of the data. Although RTD provides some advantages over DDE, it still suffers from several limitations, particularly as it is implemented in Excel, for example:

1. DCOM networking is difficult to configure. This commonly results in insecure network configuration in an effort to make the connection succeed.

2. DCOM networking does not support network proxies.

3. DCOM networking is a blocking protocol. If the application to which Excel is connected is slow to respond, Excel will freeze waiting for a response. If the network communication is slow, Excel will similarly freeze.

4. RTD does not support cell ranges. Only individual values can be transmitted via RTD. RTD can transmit more than one value in a single message, but this does not substitute for cell ranges.

5. RTD is uni-directional. Excel cannot transmit data via RTD. In order to retrieve data from Excel, another application must use an alternate protocol, namely DDE, to subscribe to that data. This means that although RTD can operate on a network it cannot transmit data bi-directionally.

6. RTD provides a notification mechanism to let Excel know that data changes have occurred. Excel must then re-read all data values that it is interested in, to determine which specific data values have changed. This may be practical for communication within the same computer or on a LAN, but it is impractical on limited bandwidth connections or on connections where bandwidth usage carries a significant cost.

7. RTD bindings in Excel consume the formula of the cell(s) into which the RTD data is bound. This exposes the RTD binding to accidental deletion. In addition, it makes it impossible to bind the same cell both to receive data and to transmit it. The act of modifying the cell value deletes the RTD binding.

8. Since an RTD binding consumes the cell formula, any Excel cell can be a participant in at most one DDE binding. That is, a cell cannot be updated from multiple sources.

9. RTD does not transmit time stamp or data quality. This limits the data's usefulness when interacting with time sensitive data and data where quality information is important.

10. RTD contains no data inspection mechanism. An RTD client cannot determine which data tags are available in the RTD server.

[0045] Accordingly, there is a need for an improved network communication means that overcomes the limitation of both DDE and RTD, so that real-time data can be exchanged using a spreadsheet application, such as Excel, bidirectionally and in a more robust manner.

[0046] There are some existing attempts at sharing Excel data via a cloud service. They generally consist of an add-in to Excel that implements a web service interface such that information from the spreadsheet can be periodically published to an external server and polled from that server based on user interaction or a timer (e.g., iPushPull - https://www.ipushpull.com). This type of application fails in the same ways as an AJAX application – they demand trade-offs between latency, volume and server resources. Such a system is inappropriate for high-volume low-latency applications like control systems and financial trading and analysis. These applications are further limited by a dependence upon the cloud service for their operation.

[0047] There are other applications that attempt to mimic Excel with a web-based application that allow users to collaborate, with each user viewing a copy of the same spreadsheet in a browser window. The browsers transmit and receive updates to the sheet via polling using AJAX (e.g., Google Sheets - https://docs.google.com/spreadsheets). Although these applications include web-based interfaces and cloud storage, they rely on polling technology that is inappropriate for high-volume low-latency applications. In addition, these applications require a user to export the document to another format in order to see the data in a different SSAPP, such as Excel, thereby breaking the real-time linkage to the exported spreadsheet.

[0048] There are applications that attempt to bridge the gap between web-based spreadsheets (like Google Sheets) and desktop spreadsheet applications (like Excel) by providing automated file format translation through a shared network storage location

(e.g., Syncplicity). These applications simply automate an import and export step that would otherwise be performed manually. They do not address real-time data sharing.

[0049] None of the existing technologies provides a mechanism for high-speed, low-latency bidirectional communication between Excel and a cloud service, between Excel and an industrial control system, nor between two or more Excel worksheets.

Summary of the Invention

[0050] The present invention provides a system and method for use of the graphical and networking features of network clients such as web browsers, RIA frameworks and dedicated applications in conjunction with at least one real-time data server to provide low-latency, real-time data applications in a web browser. The invention overcomes the limitations of current AJAX and streaming AJAX while simultaneously dealing with data sources whose data is not usable within a web browser.

[0051] The present invention also provides a long-lived, bi-directional communication mechanism from a web client that may be performed entirely over HTTP or HTTPS, preferably using existing HTTP verbs (e.g. GET and POST) while being operable with existing browser and RIA technology. Throughout this disclosure, the terms "RIA", "Rich Internet Application", "Web Browser", "network client" and "client" are understood to refer interchangeably to any software or hardware application that communicates by means of the HTTP or HTTPS protocol.

[0052] The present invention also provides a system and method for secure, end-to-end data service enabling real-time data over the Internet. The invention provides real-time connectivity between sensors, devices, and machinery and the users of their data from any remote location that is connected to the Internet, with data throughput rates that may be over 25,000 data changes per second, preferably over 50,000 data changes per second, more preferably over 75,000 data changes per second, and most preferably over 100,000 data changes per second. The added latency of the data stream may be measured in milliseconds more than the latency of the connection over the Internet itself, preferably no more than 200 milliseconds, more preferably no more than 100 milliseconds, yet more preferably no more than 50 milliseconds, yet more preferably no more than 25 milliseconds, yet more preferably no more than 10 milliseconds, and most

preferably no more than 5 milliseconds. The present invention is particularly valuable for those working with real-time data from industrial systems, embedded devices, "smart" devices or financial systems.

[0053] The invention improves upon the state of the art in real-time data delivery to web browsers and network clients by reducing the data latency to a point where visualization components can be animated using true data values, rather than interpolated values. This allows short-lived behaviour in the data to be more accurately presented to the user. Short-lived data behaviour is commonly important in understanding the true dynamics of the real-time system represented by that data. For example, a person watching a physical gauge can discern important system properties by watching vibration or overshoot in the gauge needle behaviour. In one embodiment of the invention, a digital representation of the physical gauge can capture the needle dynamics and provide the same high-quality information as the physical gauge.

[0054] The invention vastly improves the speed of data transmission from the user to the data server, reducing CPU and network costs and reducing latency. This allows the user to participate in more sophisticated control scenarios where system responsiveness is important to correct behaviour. For example, the system may require a hold-and-release interaction while filling a vessel with water. The user would press a button and hold it until the vessel is full, then release the button. Clearly, the system must respond rapidly in order to avoid over-filling the vessel. This type of control is not possible in typical web-based applications due to the unpredictability of the data delivery latency. Surprisingly, the invention makes possible classes of control and real-time data applications that were previously too slow, unreliable or primitive to be contemplated through a web browser.

[0055] Typical web applications deal with data provided in a specific format by the application designer. This may be an intentional method for limiting the end-user choice, or simply a limitation on the design. Even where the data format follows an industry standard (such as XML or JSON), the data source is specific to the application. The invention also provides a general purpose mechanism for delivering a wide variety of real-time data originating from both industry-standard and proprietary sources. Advantageously, the invention can further provide that data in a variety of data formats.

[0056] Many sources of data, both real-time and non-real-time, are not intended for network use (i.e., transmission over a network). The present invention allows data from these data sources, such as Microsoft Excel™ (Microsoft Corp.), to be reliably and rapidly delivered to any RIA or web-based application over a network. Some data sources, such as those based on OPC, were intended for network use but are not designed for communication with a web browser. The invention allows these sources to also be delivered reliably and rapidly to a web-based application. Other data sources, such as database systems, provide no interface at all for real-time information. The invention allows non-real-time data from sources such as database applications to be delivered as if it were real-time, thereby eliminating the need for a RIA or web-based application to perform very inefficient polling of the database.

[0057] Data sources and data sinks may connect to the server via persistent connections or short-lived connections. It is understood that the connection method to the server will reflect the requirements of the particular data source or sink.

[0058] The invention provides a method by which real-time data from one or more data sources is efficiently made available to a Rich Internet Application. The invention further provides a method for the RIA to efficiently modify the real-time data or generate new real-time data that can be transmitted back to the data source or sources. The data source or sources can then retransmit that data to other RIAs on the network. The invention thus effectively allows any number of RIA applications to communicate with one another in real time, and to jointly communicate with one or many real-time data sources. The invention allows for the abstraction of real-time data such that any data that can be represented using the abstraction can be made available to the RIA, regardless of its original source, representation or transfer protocol.

[0059] The present invention provides a system and method for an Internet or cloud-based communication framework and service that does not require any open incoming firewall ports for connected data sources and clients (e.g. industrial facilities, end-user client devices), thereby eliminating exposure to potential attacks. The invention provides this novel improvement by reversing the client/server relationship between the plant and the cloud server. Instead of the plant data source acting as a server, with the present invention, the plant data source acts as a client, and the cloud service acts as the server.

This reverses the direction of how a connection is made with the Internet. The plant data source server sends an outbound connection request to a server in the cloud, and therefore there is no need to open any inbound ports in the plant firewall. This novel approach keeps the plant firewall closed, and shrinks the potential attack surface to zero.

[0060] In an example aspect of the present invention, there is a method of providing real-time data to a spreadsheet application (SSAPP) from a plurality of authoritative data sources. The method can include: obtaining a first data set propagated from a first authoritative client; obtaining a second data set propagated from a second authoritative client; aggregating the first data set and the second data set into an aggregated data set; and sending at least a portion of the aggregated data set to a subscribed SSAPP over a persistent TCP connection. The SSAPP can be a first SSAPP and the second authoritative client can be a second SSAPP. Aggregating the first data set and the second data set can include converting the first data set and the second data set into a first data format and aggregating the converted first data set and the converted second data set into the aggregated data set, wherein the aggregated data set has the first data format. Sending the portion of the aggregated data set can include converting the portion of the aggregated data set into a second data format. The portion of the aggregated data set can include data that has changed since a previous communication with the subscribed SSAPP. The method can further include obtaining SSAP-originated data from the SSAP through the persistent TCP connection, the SSAPP-originated data comprising an instruction to alter the first authoritative client; and causing a modification to the fist authoritative client based on the instruction. The first authoritative client, the second authoritative client, and the SSAPP can each run on separate devices that are remote from each other.

[0061] Prior to the present invention, reversal of the client/server relationship was not done before because there was no perceived need, and it did not make intuitive or architectural sense. Existing SCADA and control systems, as well as standard industrial protocols, such as OPC, expect the server to be an authoritative holder of a data set. Since the data is being generated at a process, and then used elsewhere, it is logical that consumers of the data (e.g. outside users) are the clients, and that the clients request data from the process, the server. A client is naturally expected to connect to the server, query the data set, and subscribe to the data that the client requires. This prior art method

works well enough in a closed system that existing protocols were designed for. However, a cloud-based system requires a fundamentally new approach.

[0062] By changing the role of client and server, the present invention provides the unusual and novel case where the client becomes the authoritative holder of the data set. The process, acting as a client, connects to the cloud server and configures that server with its current data set. Updates to the data set are subsequently passed from the process to the cloud server. On the other side, users (clients) of the data connect to the cloud server by a similar method. Clients also make outbound client connections to the cloud server, and can interact with the data set in real time. On the client side as well, no incoming firewall ports need be opened. Functioning in this manner, the present invention allows a cloud server to provide access to process data without opening a single incoming port in the plant firewall or in the client's firewall.

[0063] The current invention inverts the client/server relationship. That is, the client application can optionally act as the authoritative data source, and the server can act as a non-authoritative consumer of that data. In fact, the current invention provides for a single application to act as an authoritative server, an authoritative client, a non-authoritative server and a non-authoritative client simultaneously. This makes it possible to situate a server application on a publicly accessible cloud computer that is acting as a non-authoritative server, while configuring a SCADA system within a secure network as an authoritative client. The SCADA system makes an outbound connection from within the secure SCADA network to the cloud server, and populates the cloud server with its data set. The cloud server requires no a-priori knowledge of its data set, but instead learns its data set from the authoritative client in the SCADA system. Other clients on the public network that need access to the SCADA system's data will connect to the cloud server as non-authoritative clients, thus treating the cloud server as an authoritative server for the data that in fact originates at the SCADA system. Thus, a client application is able to connect to the cloud system and interact with the SCADA system's data as if it were connecting to the SCADA system, yet the SCADA system is never exposed to the public network. An unexpected result of the present invention is to provide remote access to the SCADA system's data without compromising the network security of the SCADA system itself.

[0064] For added security, the current invention allows for a second instance of the application, operating as a non-authoritative server to the SCADA system and as an authoritative client to the cloud system, to be installed in a separate network within the industrial plant such that this second instance has no access to the secure SCADA network. The SCADA system then emits data to this second instance, and this second instance emits the data to the cloud server. In this configuration, the SCADA system does not even have a direct outbound connection to the Internet, but instead is further isolated by the network containing the second instance.

[0065] SCADA systems generally provide a mechanism to allow a client application (like an operator panel) to emit value changes to certain data items. For example, an operator may want to start or stop a machine in the plant. Plant owners may be reluctant to allow modifications to the SCADA data from remote locations. The current invention allows the SCADA system (acting as an authoritative client to the cloud) to refuse all attempts to modify the values of data items, even where user permissions would normally allow it. In addition, the cloud server can be configured to allow certain users to modify the values of certain data items based on their user credentials and the IP address of the computer from which they are connecting. Thus, the current invention provides security both from attacks via the public network and from unauthorized attempts to modify the data, even in the event that the cloud server is compromised.

[0066] Although this description refers to its application to SCADA systems, it should be understood that this same mechanism is broadly applicable for any data that may be made available via a public network. That data could originate from any program or process, such as financial trading systems, home electricity meters, remote machinery, cell phones, embedded devices, or any other program or device that generates data, and still fall within the scope of the present invention. In one aspect of the invention, a common requirement is that the data source is not be required to accept inbound network connections in order to make its data available to users of that data.

[0067] Existing spreadsheet sharing technology generally assumes that spreadsheet information will be shared among collaborating humans, and that the collaboration will happen through user actions. It also assumes that there is a directionality to that collaboration, with one user being a producer of information and other users being

consumers. This is driven in part by asymmetry in the communication model, where it may be more efficient to retrieve data into a spreadsheet than it is to publish it from a spreadsheet. In the case where more than one user is a producer, this is generally assumed to be "serial", in the sense that only one user will produce information at a time to avoid implementation problems like file sharing conflicts. An unexpected result of the current invention is that this assumption need not apply. Any user in the system can be both a consumer and producer of data at any time. Any other user can consume the data produced by any aggregation of the data from the other users. This means, for example, that a company with many stock traders can make available the content and analysis from any trader's spreadsheet to all of the other traders to share expert information widely and instantly throughout the organization.

[0068] Surprisingly, the combination of a bidirectional networked connection to control systems along with low-latency communication makes it possible to use a spreadsheet application as a user interface front-end to a control system. Previous efforts at networking SSAPPs have relied on DDE, which is not a networked technology and which does not provide a suitable bidirectional communication channel. The use of spreadsheets in control systems in the past has been limited to the acquisition of a few items from a control system for the purpose of reporting. With the current invention it is not only possible, but also practical to produce user interfaces for supervisory control *within a spreadsheet application*.

[0069] The current invention provides a high-speed, low-latency bidirectional data communication mechanism from a spreadsheet application (e.g., Excel) and any of a) an industrial control system, b) a cloud server, c) a private server, d) other spreadsheet applications and e) other instances of the same spreadsheet application. The current invention further provides this capability over a network, including LAN, WAN and Internet-accessible networks.

[0070] The current invention provides a real-time bidirectional connection to a cloud system, where the cloud system is also connectible to IoT (Internet of Things), IIoT (Industrial Internet of Things) and Industrie 4.0 devices and protocols. This effectively makes the spreadsheet application into an IoT device, with all of the capability of other IoT devices both for data production and consumption.

[0071] Closed-loop control consists of acquiring data from a data source, performing a computation or decision based on that data, and then submitting a control signal back to that data source without human intervention. Closed-loop control systems are normally implemented as close as possible to the system being controlled to reduce the chance of latency or connectivity problems producing late control or no control at all. Being close to the system being controlled does not necessarily mean physical proximity, but rather being separated from the physical system by few transducers, signal converters, data collection devices, processors and network segments. This could be thought of as transmission proximity. Application-critical closed-loop control, like emergency stop logic, is normally implemented using PLCs that are near in terms of transmission proximity to the system being controlled. For less critical control, like optimal routing decisions, closed-loop control can be implemented in SCADA systems, expert systems or other application-specific logic. This type of control requires flexibility in how control logic is expressed, and sufficient reliability, speed and low latency to be practical in the application. Spreadsheets provide the flexibility to express the control logic, but until the current invention have not provided the communication reliability and low latency required by many control systems. The current invention makes close-loop control using a spreadsheet practical in many more systems than was previously thought possible, which is an unexpected improvement over conventional solutions.

[0072] The concept of closed-loop control extends beyond control system. For example, a stock trader would use Excel to implement a decision-making algorithm to determine when to trade a stock. The current invention makes it possible to use the results from that algorithm to automatically initiate a trade once the algorithm has signaled its decision.

[0073] Similarly, a spreadsheet could monitor home sensors like such as proximity, motion, window and door sensors to and then to implement sophisticated decision algorithms that then actuate lights, heating systems, coffee makers and other electronic and physical equipment.

[0074] Because the present invention connects a spreadsheet to the IoT, any closed-loop scenario could be addressed from a spreadsheet operated near the system being controlled, or anywhere in the world.

[0075] The present invention allows spreadsheet applications, such as Excel, to communicate real-time data to and from a program running locally on the same computer, and/or over a network. In so doing, the present invention enables Excel to act as a participant in the Internet of Things (IoT) and the Industrial Internet of Things (IIoT). In addition, the present invention provides a mechanism for multiple Excel applications to share data in real time through a broker situated anywhere on a LAN, WAN or Cloud / Internet-accessible networks. Both of the communication mechanisms provided in Excel (DDE and RTD) suffer from limitations that the present invention overcomes; advantageously, the present invention:

1. Uses TCP/IP for network communication, including support for WebSockets, network proxies, direct TCP connections, SSL and authentication.

2. Allows Excel to exchange information with a data broker that provides a publish/subscribe data model.

3. Is fully bi-directional. It uses the same connection for both inbound (read from the broker) and outbound (write to the broker) data.

4. Only transmits data values that have changed since the last update, minimizing network bandwidth utilization.

5. Binds to cells and ranges in Excel without requiring an Excel function in the bound cell(s). This makes it possible to implement cell bindings that are bi-directional. Modifying the cell value will not result in a loss of binding.

6. Supports cell ranges for efficient transmission.

7. Automatically reconnects to the broker if the connection is lost

8. Employs an asynchronous data transmission model to avoid freezing Excel while waiting for data to be transmitted or received.

9. Provides tag, type, value, quality and timestamp information for every data item.

10. Allows all configuration to be performed from within Excel, for both inbound and outbound data.

11. Allows a cell to participate in multiple bindings simultaneously. A cell can be updated by multiple sources, and can be a member of a larger range that is transmitted or received via a different binding.

12. Allows the spreadsheet application to act as the authoritative source of a data set.

13. Mulitple spreadsheet applications with different spreadsheets can simultaneously share and consume identical or different data sets in an easy and robust manner.

[0076] In addition to acting as a means of communicating data to and from the SSAPP, the current invention may optionally act as a pre-processor for the information being received by the SSAPP. Large-scale spreadsheet applications can suffer from performance limitations due to the nature of the SSAPP's formula evaluator. The current invention may optionally perform computations, including aggregation, transformation, filtering and any other computation expressible as computer software prior to presenting that information as cell contents in the SSAPP. Similarly, the current invention makes it possible to process the data originating in the SSAPP's computation engine prior to transmitting it via the network. Common cases may include transformation, aggregation and filtering of information prior to transmission. As an example, a powerful filtering step may be to choose not to transmit a value at all based on its similarity to a previously transmitted value. This type of filtering is commonly referred to as a deadband filter. That would act to reduce network bandwidth utilization and to reduce the processing load on any other application working on that data.

Brief Description of the Figures

[0077] FIG. 1 is an exemplary block diagram illustrating a direct connection between a SSAPP and a data server, in accordance with one embodiment of the present invention.

[0078] FIG. 2 is an exemplary block diagram illustrating a connection between a SSAPP, a server, and a separate data source, in accordance with one embodiment of the invention.

[0079] FIG. 3 is an exemplary block diagram illustrating connections between multiple SSAPPs, a server, and multiple, separate data sources, in accordance with one embodiment of the invention.

[0080] FIG. 4 is an exemplary flowchart illustrating one method of SSAPP control flow, in accordance with one embodiment of the invention.

[0081] FIG. 5a, b is an exemplary flowchart illustrating one method of operation of a server, in accordance with one embodiment of the invention.

[0082] FIG. 6 is an exemplary block diagram illustrating a data server managing simultaneous connections to multiple SSAPPs, in accordance with one embodiment of the invention.

[0083] FIG. 7 is an exemplary block diagram illustrating real-time transmission of data via a local or wide area network between a spreadsheet application and a SSAPP, in accordance with one embodiment of the invention.

[0084] FIG. 8 is an exemplary block diagram illustrating a system implementation, in accordance with one embodiment of the invention.

[0085] FIG. 9a, b, c is an exemplary flowchart illustrating one method of operation of a client and a server, in accordance with one embodiment of the invention.

[0086] FIG. 10 is an exemplary block diagram illustrating a prior art system implementation.

[0087] FIG. 11 is an exemplary block diagram illustrating a system implementation, in accordance with one embodiment of the invention.

[0088] FIG. 12 is an exemplary block diagram illustrating a system implementation, in accordance with one embodiment of the invention.

Detailed Description of the Invention

[0089] The following description is presented to enable any person skilled in the art to make and use the invention, and is provided in the context of particular applications of the invention. Various modifications to the disclosed embodiments will be readily apparent to those skilled in the art, and the general principles defined herein may be applied to other embodiments and applications without departing from the scope of the present invention. Reference to various embodiments and examples does not limit the scope of the invention, which is limited only by the scope of the claims attached hereto. Additionally, any examples set forth in this specification are not intended to be limiting and merely set forth some of the many possible embodiments for the claimed invention.

[0090] The program environment in which a present embodiment of the invention is executed illustratively incorporates a general-purpose computer or a special purpose device such as a hand-held computer, telephone or PLC. Details of such devices (e.g., processor, memory, data storage, display) may be omitted for the sake of clarity.

[0091] It is also understood that the techniques of the present invention may be implemented using a variety of technologies. For example, the methods described herein may be implemented in software executing on a computer system, or implemented in hardware utilizing either a combination of microprocessors or other specially designed application specific integrated circuits, programmable logic devices, or various combinations thereof. In particular, the methods described herein may be implemented by a series of computer-executable instructions residing on a suitable computer-readable medium. Suitable computer-readable media may include volatile (e.g., RAM) and/or non-volatile (e.g., ROM, disk) memory, carrier waves, non-transitory computer-readable mediums, and transmission media (e.g., copper wire, coaxial cable, fiber optic media). Exemplary carrier waves may take the form of electrical, electromagnetic or optical signals conveying digital data streams along a local network, a publicly accessible network such as the Internet or some other communication link.

[0092] In reference to the example embodiments shown in the figures, it is understood that simplified examples were chosen for clarity. Single instances of an element (e.g. a SSAPP, a server, a client, a data source, a data sink, etc.) appearing in the

figures may be substituted for a plurality of the same element, and still fall within the scope of the present invention.

[0093] Accordingly, in one aspect, the present invention provides a method of providing real-time data to a SSAPP, the method comprising: producing data at a data source; propagating the data to a server; collecting the data at the server; creating a persistent connection from the SSAPP to the server; and subscribing the SSAPP to subscribed data, wherein the subscribed data comprises at least some of the data collected at the server, wherein the server propagates the subscribed data to the SSAPP through the persistent connection as the data is collected at the server. The method may further comprise sending SSAPP-originated data to the server. The SSAPP-originated data may contain at least one change request to the data or at least one command to the server through the persistent connection. Further, the data may be propagated through at least one intermediate component. The server may receive the at least one change request and transmit the at least one change request to the data source. The at least one change request may be transmitted through the intermediate component. The intermediate component may be an intermediate hardware component or an intermediate software component. Optionally, the SSAPP may subscribe to the subscribed data. Producing data at the data source and propagating the data to the server may be concurrent with collecting the data at the server. The SSAPP may perform an action based upon the data, such as a calculation or a modification of a graphical representation. The SSAPP may provide a visual representation of the data on a user display, and a user may interact with the visual representation to generate SSAPP-originated data. The visual representation may be a program running within a SSAPP framework. The SSAPP-originated data may instruct the server to perform an action, such as to shut down the server, or to alter its behaviour, such as to alter which data arrives from the server.

[0094] For example, SSAPP-originated data may be as a result of user interaction, a timer event, a response to a data change coming from the server, a script, or another non-user generated event.

[0095] In another aspect, the present invention provides a method of providing real-time data to a SSAPP, the method comprising: providing data from a data source; propagating data from the data source to a server; collecting data at the server; producing

data at the SSAPP; creating a first persistent connection from the server to the SSAPP; creating a second persistent connection from the SSAPP to the server; propagating data from the SSAPP to the server through the second persistent connection; and subscribing the SSAPP to subscribed data, wherein the subscribed data comprises at least some of the data collected at the server, and wherein the server propagates the subscribed data to the SSAPP through the first persistent connection. The method may further comprise propagating data from the server to a data sink. The first persistent connection and the second persistent connection may consist of a single connection. The data source, data sink and server may consist of a single component, or any combination of two or more components. The data may be propagated though at least one intermediate selected from the group comprising: a software component, a hardware component, and a network.

[0096] A data item may be propagated between the SSAPP and the server on a subscription basis, wherein the data item is propagated immediately in response to a change in the data item. The propagated data may be selected from the group comprising: numeric data, non-numeric data, configuration settings and executable commands. The SSAPP may perform an action based upon the data, where the action is selected from the group comprising: a modification of a visual representation of a user display, a calculation, production of new data, modification of existing data, storage of data, an audible indication, execution of a script, propagation of data to the server, a user-visible programmatic response, and a non-user-visible programmatic response. Data produced at the SSAPP may instruct the server to perform an action selected from the group comprising: modification of data within the server, propagation of the data to data sinks connected to the server, execution of a script, storage of the data to a file system, creation of new data, propagation of new data to data sinks connected to the server, modification of a server configuration, modification of a server behaviour, a user-visible programmatic response, and a non-user-visible programmatic response.

[0097] In yet another aspect, the present invention provides a computer readable storage medium storing instructions that, when executed on one or more computers, causes the computers to perform methods of providing real-time data to a SSAPP as described above.

[0098] In another aspect, the present invention provides a system for providing real-time data to a SSAPP, the system comprising: at least one data source; at least one server comprising: a data collection component for collecting data from the at least one data source; and a data emission component for emitting data to at least one data client; at least one SSAPP; and optionally at least one data sink. The server may further comprise a data modification component for modifying the form of the data collected by the data collection component for emission by the data emission component. It is understood that the at least one data source and at least one server may be implemented in at least one computer program (i.e. a single computer program, or two or more separate computer programs).

[0099] The server may further comprise one or more components selected from: a data modification component; a data creation component; a user interface component; a computer file system interaction component; a program interaction component for interacting with other programs running on a computer running the server; a scripting language component to perform programmable actions; a HTTP component for accepting HTTP requests from client programs and respond with documents as specified by those requests, in a manner analogous to a "web server", including the ability to dynamically construct the document in response to the request, and to include within the document the current values of the data resident in the server and the results of executing statements in the server's built-in scripting language; a synchronization component to exchange and synchronize data with another running instance of the server on any local or network-accessible computer, such that both servers maintain essentially identical copies of that data, thereby enabling client applications connected to either instance of the server to interact with the same data set; a first throttling component to limit the rate at which data is collected; a second throttling component to limit the rate at which data is emitted; a connectivity component to detect a loss of connectivity to other servers, and to reconnect to the other servers when connectivity is regained; a redundancy component to redundantly connect to multiple other servers of identical or similar information such that data from any of the other servers may be collected in the event that one or more of the other servers is inaccessible; and a bridging component to "bridge" data among sources of data such that some or all of the data within those sources will maintain similar values with one another, or bridge data among data sources including a mathematical

transformation such that the data in one source is maintained as the mathematical transformation of the data in the other source, including the ability to apply the mathematical transformation in both the forward and inverse directions through a bi-direction bridging operation. It is understood that this set of server components could be extended by adding additional functionality to the server to support other data collection and transmission mechanisms, other processing mechanisms and other storage mechanisms.

[0100] The data collection component may collect data in one or more of the following manners: on demand, wherein the server sends a request for some or all of the data resident in another server, and that other sever responds with the current value or values of the requested data only once in response to the request; by subscription, wherein the server sends a request for a subscription to some or all of the data resident in another server, and the other server responds by sending the current value or values of its data, and then continues to send any subsequent changes to the value or values of the data until the server either terminates its connection to the other server, or requests that the other server cease sending updates; on a trigger, wherein a client, script or human (a "user") configures the server to collect the data only if a certain trigger condition is met, be that a timer, a time of day, a data change, a change in the system status, a user action or some other detectable event; and passively by waiting for a "client" application to send data to the server.

[0101] The data emission component may emit data in one or more of the following manners: on demand, wherein a "client" application sends a request for some or all of the data, and the sever responds with the current value or values of the requested data only once in response to the request; by subscription, wherein a client application sends a request for a subscription to some or all of the data, and the server responds by sending the current value or values of the data, and then continues to send any subsequent changes to the value or values of the data until the client either terminates its connection to the server, or requests that the server cease sending updates; and on a trigger, wherein a client, script or human (a "user") configures the server to emit the data only if a certain trigger condition is met, be that a timer, a time of day, a data change, a change in the system status, a user action or some other detectable event.

[0102] The data collected at the data collection component may be received using one or more transmission protocols selected from: Dynamic Data Exchange (DDE), OLE for Process Control (OPC), OPC Alarm and Event specification (OPC A&E), OPC Unified Architecture (OPC-UA), OPC Express Interface (OPC-Xi), TCP/IP, SSL (Secure Socket Layer) over TCP/IP through a custom interface, Hypertext Transfer Protocol (HTTP), Secure HTTP (HTTPS), Open Database Connectivity (ODBC), Microsoft Real-Time Data specification (RTD), Message queues, Windows Communication Foundation (WCF), industrial bus protocols such as Profibus and Modbus, Windows System Performance Counters, TCP/IP communication from embedded systems, TCP/IP communication from non-MS-Windows systems, TCP/IP communication from Linux, TCP/IP communication from QNX, TCP/IP communication from TRON, TCP/IP communication from any system offering a C compiler and TCP implementation, Scripts written using a built-in scripting language, data entered by humans through a user interface, data read from a local disk file, data read from a remotely accessible disk file, proprietary formats, user-defined formats, and formats added through extensions to the server. An example of a proprietary format is Wonderware SuiteLink™.

[0103] The data emitted from the data emission component may be transmitted using one or more transmission protocols selected from: Dynamic Data Exchange (DDE), OLE for Process Control (OPC), OPC Alarm and Event specification (OPC A&E), OPC Unified Architecture (OPC-UA), OPC Express Interface (OPC-Xi), TCP/IP, SSL (Secure Socket Layer) over TCP/IP through a custom interface, Hypertext Transfer Protocol (HTTP), Secure HTTP (HTTPS), Open Database Connectivity (ODBC), Microsoft Real-Time Data specification (RTD), Message queues, Windows Communication Foundation (WCF), industrial bus protocols such as Profibus and Modbus, TCP/IP communication to embedded systems, TCP/IP communication to non-MS-Windows systems, data presented to humans through a user interface, data written to a local disk file, data written to a remotely accessible disk file, proprietary formats, user-defined formats, formats added through extensions to the server, electronic mail (E-Mail), and Short Message Service (SMS) message format.

[0104] Further, the data collected at the data collection component may be in a format appropriate to the transmission protocol. The data emitted from the data emission

component may be in a format appropriate to the transmission protocol. The data collected at the data collection component and the data emitted from the data emission component may also be in a format selected from: parenthetical expression (LISP-like) format, Hypertext Markup Language (HTML), eXtensible Markup Language (XML), JavaScript Object Notation (JSON), proprietary binary format, user-definable text format, and a format added through extension of the server.

[0105] The system may further comprise an Application Programming Interface (API) that implements a TCP/IP connection and one or more of the data formats supported by the server, which may assist a programmer in establishing a connection as described above. The API may be implemented for one or more of the following platforms: "C" programming language, "C++" programming language, Microsoft .Net programming environment, Microsoft Silverlight framework, Adobe Flash framework, Adobe Air framework, a programming language supporting TCP/IP communication (including any scripting language), and a SSAPP framework supporting TCP/IP communication.

[0106] The SSAPP may comprise support for: making a first long-lived TCP/IP data connection to the server to receive data; receiving data from the server; and transmitting data to the server over a second TCP/IP data connection. The data may be received from the server on demand or by subscription. The first TCP/IP data connection and the second TCP/IP data connection may be the same connection. The second TCP/IP data connection may be a long-lived connection. The second TCP/IP data connection may be a short-lived connection. The TCP/IP data connection to the server may be in a protocol selected from: an API, as described above, a direct TCP/IP connection, HTTP and HTTPS.

[0107] The client may be implemented using a RIA framework, a web browser, a compiled computer language, an interpreted computer language, a hardware device, or another implementation mechanism that supports the HTTP and/or HTTPS protocols. The client may comprise support for: making a first long-lived TCP/IP data connection to the server to receive data; receiving data from the server; and transmitting data to the server over a second long-lived TCP/IP data connection. The data may be received from

the server on demand or by subscription. The TCP/IP data connections to the server may be in a protocol selected from: HTTP and HTTPS.

[0108] Data from the server may be received, or data to the server may be transmitted, in one or more form selected from: a parenthetical expression (LISP-like) format, Hypertext Markup Language (HTML), eXtensible Markup Language (XML), JavaScript Object Notation (JSON), a proprietary binary format, a user definable format, and a format added by extension to the server.

[0109] The SSAPP may further comprise support for presenting a graphical display representing the data to a user. The graphical display may comprise one or more graphical elements selected from: a textual display, a slider, a chart, a trend graph, a circular gauge, a linear gauge, a button, a check box, a radio button, a progress bar, a primitive graphical object, controls supported by the SSAPP, custom controls created to extend the SSAPP, third-party controls implemented using the SSAPP, and a customized graphical element.

[0110] Configuration information of the graphical display may be saved on the server, as well as loaded from the server. A graphical element may be created and modified within the graphical display. The graphical element may be a customized graphical element, customizable by a user, wherein the customization may be saved on the server. Customization may be performed by a programmer, without requiring modification to an application implemented in the RIA framework. The customized graphical element may be available for use to a user in other graphical displays. These customizations may be for creating new displays, modifying existing displays, all in addition to the graphical elements originally supported by the user interface application. The graphical element may comprise one or more property that is user-modifiable, and which may be selectable by a programmer. User interaction with the graphical element may cause a user interface application to emit modifications to the data to the server. A user-only mode may be provided to disallow creation or modification of the graphical display by a user, and a read-only mode may also be provided to disallow interaction with the graphical element by the user. A system administrator may select which user and for which graphical display a user interface application will operate in one of the user-only mode and read-only mode. The user may be required to identify himself, and

where such identification is required, the user interface application may operate in at least one of the user-only mode and the read-only mode. Advantageously, the features of the invention allow modification of the graphical displays through any user RIA terminal and the resulting changes, upon saving, are immediately available to all other RIA terminals connected to the server.

[0111] In another aspect, the present invention provides a method of providing bi-directional streaming communication over the HTTP or HTTPS protocol between a client and a server, the method comprising: generating a session ID; opening a first socket via a first HTTP transaction from the client to the server; associating the session ID with the first socket at the server and client; opening a second socket via a second HTTP transaction from the client to the server; associating the session ID with the second socket at the server and at the client; maintaining a long-lived connection on the first socket; and maintaining a long-lived connection on the second socket, wherein a correspondence is created among the session ID, the first socket and the second socket, and wherein bi-directional communication is established between the client and the server.

[0112] The method may further comprise the client transmitting at least one data message selected from the group comprising: configuration information, commands, real-time information, pending data from a previous transaction, and other data. The method may further comprise waiting for an event from the first socket; verifying whether the event from the first socket is an error; reading available data from the first socket when the event is not an error; processing the data to produce a result; and optionally sending the result to the server via the second socket. The method may further comprise the client: closing the first socket; and closing the second socket, wherein the event from the first socket is an error. The method may further comprise the client: waiting for a client-generated event; processing the client-generated event to produce a result; and optionally sending the result to the server via the second socket. The client-generated event may be selected from the group comprising: an internally-generated stimulus, a result of user activity, a timer, and an external stimulus. The method may further comprise the client: marking data for transmission to the server as pending; closing the second socket; opening a new second socket; and associating the new second socket with the session ID.

[0113] The method may further comprise the server: waiting for an event from the second socket; verifying whether the event from the second socket is an error; reading available data from the second socket when the event is not an error; processing the data to produce a result; and optionally sending the result to the client via the first socket. The method may further comprise the server closing the second socket, wherein the event from the second socket is an error. The method may further comprise the server: waiting for a server-generated event; processing the server-generated event to produce a result; and optionally sending the result to the client via the first socket. The server-generated event may be selected from the group comprising: an internally-generated stimulus, a result of user activity, a timer, a result from another connected client, data from a data source, and an external stimulus. The method may further comprise the server: closing the first socket; and closing the second socket.

[0114] In the above method, the first HTTP transaction may be selected from the group comprising: a HTTP GET transaction and a HTTP HEAD transaction; and the second HTTP transaction may be selected from the group comprising: a HTTP POST transaction, a HTTP PUT transaction, a HTTP PATCH transaction, and a HTTP TRACE transaction. Preferably, the first HTTP transaction is a HTTP GET transaction, and the second HTTP transaction is a HTTP POST transaction.

[0115] In yet another aspect, the present invention provides a system for providing bi-directional streaming communication over the HTTP or HTTPS protocol, the system comprising: at least one client; and at least one server, wherein the at least one client is adapted to implement the above-described method, and wherein the at least one server is adapted to implement the above-described method. The at least one client may comprise a RIA. The at least one server may comprises: a data collection component for collecting data from the at least one data source; and a data emission component for emitting data to at least one data client.

[0116] In yet a further aspect, the present invention provides a computer readable memory storing instructions that, when executed on one or more computers, causes the computers to perform a method of providing bi-directional streaming communication over the HTTP or HTTPS protocol between a client and a server, the method comprising the steps of the above-described method.

[0117] In yet a further aspect, the present invention provides a computer readable memory storing instructions that, when executed on one or more computers, causes the computers engage in a bidirectional networked real-time data exchange over the HTTP or HTTPS protocol between a SSAPP and a server, the method comprising the steps of the above-described method.

[0118] As described above, the HTTP protocol implements a transaction model where each transaction is generally short-lived. Each transaction is initiated by the client, and is specified to either transmit data to the server, or to request data from the server, but not both.

[0119] A web client may need to transmit or receive a large volume of data. In this case, it may implement an API that allows the client to send and receive the data in incomplete chunks. That is, it may require multiple send and receive actions before the entire data set has been transmitted. For example, a client that receives an image from a server may receive the image in chunks of 1KB so that it can begin to render the image before the entire image has arrived to produce a progressive rendering effect. This behaviour can be leveraged within the client to produce a continuous stream of data. The client may make an HTTP GET request to a URL on a specially designed server (or a standard server with a specially designed handler for that URL). The server may respond with an HTTP header, and then hold the socket open. At any time in the future, the server may transmit data on the socket, which will arrive at the client as an incomplete transmission. The client can process this data and then wait for more. So long as the server holds the socket open, the client will simply act on the expectation that there is more data to be received, and will process it as it arrives. The server can transmit more information asynchronously to the client at any time without the need for the client to repeatedly open and close HTTP connections. This mechanism is the underlying methodology of Streaming AJAX. As disclosed above, it is uni-directional. This mechanism does not provide high-speed communication from the client to the server.

[0120] One of the important innovations of the present invention is to solve the problem of creating a high-speed connection from the client to the server. The solution provides that the client opens an HTTP POST transaction with the server, and transmits the necessary HTTP header information. The server will then wait for the data payload

of the POST to arrive. At any time in the future, the client may transmit data on the open socket, effectively acting like the Streaming AJAX mechanism in the reverse direction. The client may hold the socket open indefinitely, transmitting data as necessary without having to repeatedly open and close HTTP connections for each new transmission.

[0121] The server must be aware that the data will arrive as a stream, and to process the information as it arrives. This may require custom behaviour in the server.

[0122] The HTTP protocol specifies that a client must inform the server of the size of an HTTP POST message in the HTTP headers (the content-length). It is a violation of the HTTP protocol for the client to transmit more or less data than specified in the content-length header. The present invention recognizes this by tracking the number of bytes transmitted from the client to the server. The HTTP POST content length is specified by the client to be an arbitrary number of bytes. When the client has transmitted content-length bytes, it closes its existing connection and opens a new connection and continues transmitting. The number of bytes in a POST message can be large (e.g. up to 2^31 bytes), so this open and close will happen very infrequently. The result will be a slight latency in the transmission of some data, but no loss of information.

[0123] In a preferred embodiment, the present invention requires two sockets, one handling the server-to-client communication via HTTP GET, and the other handling client-to-server communication via HTTP POST. In order for these two sockets to act in concert to provide bi-directional streaming communication, the web server must be aware that they are related halves of a single conversation. This relationship may be established by the client. The client opens the HTTP GET connection first, and includes in its URL a unique session handle (e.g., a randomly generated GUID). When the client subsequently opens the HTTP POST request, it includes the same session handle in the URL. The server is then able to associate the two connections. When the HTTP POST connection must be closed and re-opened due to reaching the content-length limit, the client transmits the same GUID again. The server is then able to associate this new POST socket with the existing GET socket.

[0124] The web server needs to understand that this methodology is being employed. It must keep track of calls to a specially designated URL for the original GET

connection, associate the session handle with that connection, and then subsequently associate POST connections with the same session handle with that GET connection. It may be desirable, but not necessary, for the web server to spawn a separate thread to handle each connection pair.

[0125] Having established the GET and POST connections, the client can receive asynchronous data transmissions from the server via the GET connection and transmit asynchronous data to the server via the POST connection. The server does the reverse, transmitting data via the GET connection and receiving data via the POST connection. The behaviour of both client and server are otherwise the same as if they were communicating via a single bi-directional socket.

[0126] As will be understood by a person skilled in the art, other HTTP verbs such as HEAD, PUT, PATCH and TRACE may also be used. It will also be appreciated, for example, that it is possible to further modify a server to recognize other verbs or relax protocol restrictions on the HEAD transaction to behave like a GET. So, other verbs may be used if the server is modified to recognize the added/different behaviour. Such modifications depart from a strict implementation of the HTTP specification, yet still fall with the present invention.

[0127] The unexpected advantages of the present invention in regard to the system and method for secure real-time cloud services are several. To address security concerns, one prior art method for sharing process data on the cloud has been to use a Virtual Private Network ("VPN"). However, from a security perspective, use of a VPN is problematic because every device on the VPN is open to every other machine. Each device (and each user of said device) must be fully trusted on the VPN. Security is complex and not very good, making it virtually impossible to use this approach for open communication between companies. Accordingly, the present invention allows sharing of data between third party companies without requiring that the third parties access an existing VPN, and therefore never exposing computers and devices on the VPN to those third parties. Furthermore, VPNs also incur a performance penalty, either compromising real-time performance or significant additional cost to compensate (e.g., by requiring additional hardware, computational resources and complexity to a system).

[0128] Further advantageously, the present invention allows users to connect plant floor equipment to management as well as partner and third-party companies, using software at the plant site that is configured by the client company to allow specific data streams to be uploaded or downloaded.

[0129] The present invention may be completely software-based, and can be implemented on existing hardware, therefore not introducing significant complexity to an established network.

[0130] Advantageously, using methods disclosed herein, once the client/server connection is established, the data can flow in either direction. Client users can monitor a system in real time, effect changes, and see the effect of their actions immediately, as if they were working on a local system. Or, if required, the system can be configured from the plant to be one-way, read-only.

[0131] The present invention provides the ability to connect to any industrial system, using open, standard protocols like OPC, TCP, and ODBC. Such flexibility allows further cost reduction by fully utilizing investments in existing equipment, or enhance new installations with cloud connectivity. Examples uses of the present invention are the addition to existing SCADA systems, enhanced function as an HMI for an individual machine, or access RTUs or even individual embedded devices.

[0132] In combination with methods disclosed herein, the present invention supports publish/subscribe data delivery, an event-driven model in which a client registers for data changes one time and then receives subsequent updates immediately after they occur. This low-latency, cloud-based system adds extremely low overhead to the overall data transmission time, effectively keeping throughput speeds to just a few milliseconds (or less) more than the network propagation time.

[0133] In one embodiment, the present invention may achieve very high-speed performance is by handling data in the simplest possible format. Providing a data-centric design, the present system can function with various kinds of data sources and users, such as control systems, OPC servers, databases, spreadsheets, web pages, and embedded devices. Preferably, when a connection is made to the cloud server, incoming data is stripped of unnecessary formatting (XML, HTML, OPC, SQL, etc.) and passed as

quickly as possible to any registered clients. At the receiving end the data is delivered in whatever format the client requires.

[0134] With the methods disclosed herein, a RIA or web-based user interface for secure cloud services provide anywhere-access to register for the service, configure data connection options, and monitor usage and costs. Additionally, all data display screens may be provided via the web-based interface. This web-based HMI allows users to create pages from anywhere, and deploy them immediately.

[0135] Further advantageously, one of the benefits of cloud computing is its ability to scale up or down to meet the needs of its users. The present invention can not only handle bursts of high-speed activity in the data flow, it can also be quickly configured to meet the needs of a growing system. Users can add data points to a particular device, or bring on new devices, new SCADA systems, even new locations and installations through an easy-to-use, web-based configuration interface.

[0136] The present invention is operable as a real-time industrial system, and can maintain a suitable level of performance and security in a cloud environment. Its sophisticated connectivity options allow the primary control system in a plant to continue functioning without disruption. The result is a robust and secure feed of live process data into an enterprise to provide opportunities for real-time monitoring, collaboration, and predictive maintenance.

[0137] Referring to FIG. 1, in one embodiment, SSAPP 101 makes a data connection directly to a program that is acting as both data source and data server 100. This could occur where the data source is both a collector of raw data and a transmitter via a TCP/IP protocol. An example of this would be an OPC-UA server embedded within a PLC. Another example would be an embedded device that acts as a data source and provides a TCP/IP server capability offering a custom TCP/IP interface. Yet another example would be a stock market data feed that offers a TCP/IP interface.

[0138] Referring to FIG 2, in one embodiment, another configuration comprises a separate data source 202 and server 203. This configuration extends the communication model by converting the data protocol of data source 202 into a TCP/IP protocol that can be processed by SSAPP 201. This greatly broadens the number and type of data sources

202 by allowing the server 203 to interact with data sources 202 that do not provide a TCP/IP interface directly.

[0139] Referring to FIG. 3, in one embodiment, server 303 may manage connections to more than one data source 302 and to more than one SSAPP 301 simultaneously. This more complex configuration performs aggregation of data from data sources 302 and SSAPPs 301 into a single data set that is accessible from anywhere on the TCP/IP network.

[0140] In another embodiment, a system may include multiple servers, interconnected with one or more data sources and/or one or more SSAPPs.

[0141] Referring to FIG. 4, in one embodiment, a method of SSAPP behaviour and control flow is shown. The SSAPP does not require an explicit stopping criterion, though one or more may be incorporated. The SSAPP is stopped implicitly when a user closes the application. The SSAPP simultaneously follows two flows of control, which can be either interleaved in a single program thread or implemented in separate program threads. The method may comprise additional processing specific to the SSAPP.

[0142] In the first flow of control, the SSAPP attempts to establish and maintain a connection to a server, and to respond to changes in the data available from the server. The SSAPP first attempts to establish a connection (Step 401). If the connection is not successful, it simply re-tries that connection indefinitely. If the connection succeeds (Step 402) then the SSAPP may subscribe to all or part of the data set (Step 403). Alternatively, it is possible for the server to implicitly subscribe the SSAPP to the data set based on the presence of a connection, in which case Step 403 may be skipped. In addition to a subscription, the SSAPP may also transmit other information to the server to configure the behaviour of the data transmission, such as a minimum time between updates or timeout parameters on the connection.

[0143] Having once established a connection, the SSAPP waits for notifications of a change in data from the server (Step 404). If a data change has occurred (Step 405) then the SSAPP processes that data in some fashion (Step 407). This processing may be to modify an internal state of the SSAPP, modify cell contents, modify a graphical representation, play a sound or any other programmatic response that the SSAPP

designer determines. If no data change occurs, the SSAPP checks to determine if the connection to the server has been lost for any reason. If the connection has not been lost, the SSAPP returns to wait for a data change to occur (Step 404). If the connection has been lost then the SSAPP re-tries the connection to the server (Step 401).

[0144] Simultaneously with Steps 401 through 407, the SSAPP may also accept user input, allowing the user to generate changes in the data that can be propagated back to the server. The SSAPP waits for user input (Step 420) either in a separate program thread or multiplexed with Steps 401 through 407.

[0145] FIG. 4 exemplifies a separately threaded method. If user input has occurred (Step 421) then the SSAPP can attempt to transmit the resulting data to the server. It does this by first checking to see if the server is connected (Step 422). If so, the SSAPP transmits the new data to the server (Step 423). If not, the SSAPP waits for more input (Step 420). The check for the server connection (Step 422) may be implicit in the attempt to transmit the data, in which case Steps 422 and 423 are combined in practice.

[0146] The SSAPP may also be non-interactive such that user input is not accepted, in which case Steps 420 to 423 can be omitted.

[0147] Referring to FIG. 5 in one embodiment, the method of operation of a data server is shown. The server may be simultaneously collecting data from zero or more data sources while serving data to zero or more SSAPP connections. The two main flows of control can be implemented in separate threads, or by interleaving the two flow control paths within a single thread.

[0148] In order to interact with a data source, the server must first establish a connection to that data source (Step 501). Normally, the server initiates this connection to the data source. In some cases, the data source may initiate the connection to the server. If the connection succeeds (Step 502), the server begins collecting data from the data source (Step 503). If the connection fails, the server re-tries the connection to the data source (Step 501). If the data source is the initiator of the connection to the server, then Steps 501 and 502 collapse to a single wait state and the server passively waits for the data source to connect. The data collection (Step 503) will follow a method appropriate to the data source, and may differ from one data source to another. The

server can be made to accommodate any data source whose data can be represented in the server. If new data becomes available from the data source (Step 504), the server converts that data to the server's internal data representation. This allows the server to aggregate data from a variety of data sources using different data representations. Step 506 can be omitted in the simple case where the data source, server and SSAPP all use the same data representation. The server then attempts to transmit the data to each SSAPP. The server may first establish that a SSAPP is connected (Step 507). If one or more SSAPP is connected, the server converts the data to a representation suitable for the SSAPP (Step 508) and transmits that data to each connected SSAPP (Step 509). If no SSAPP is connected, the server continues collecting data from the data source (Step 503). The server repeats this sequence (Steps 501 - 509) indefinitely. The server may choose not to collect data from a data source when no data sink is connected to the server that requires data from that data source.

[0149] Simultaneous with, or interleaved with, collecting data from the data source, the server also manages connections from SSAPPs. The server waits for a connection from an SSAPP (Step 520). When an SSAPP attempts to connect to the server (Step 521) the server accepts the connection (Step 522) and continues to wait for connections from other SSAPPs. While waiting for an SSAPP to connect, the server must also determine whether an existing SSAPP connection has disconnected (Step 523). If an SSAPP has disconnected, the SSAPP connection is removed from any tracking in the server (Step 524) so that no attempt is made in future to transmit data (Step 509) to the disconnected SSAPP. The server repeats this sequence (Steps 520 - 524) indefinitely. The server may apply acceptance criteria when the SSAPP attempts to connect (Step 522) such that the server may refuse the connection for any reason, such as an authentication failure or a server-applied limit on the maximum number of concurrent connections from SSAPP instances.

[0150] Simultaneously with, or interleaved with, collecting data from the data source and managing new connections from SSAPPs, the server may also receive data from SSAPPs already connected. The server waits for data to arrive from the SSAPP (Step 530). When new data arrives (Step 531), the server converts this data into the server's internal data format (Step 532). The server then determines if any SSAPP is currently connected (Step 533). The server then converts the data to a format suitable for receipt

by the SSAPP (Step 534) and transmits the data to each currently connected SSAPP (Step 535). The server then determines if any data source that requires this change of information is currently connected (Step 536). For each data source requiring the information that is currently connected to the server, the server converts the data to a format suitable for that data source (Step 537) and transmits the data (Step 538). The server repeats this sequence (Steps 530 - 538) indefinitely.

[0151] Steps 501 through 509 can be replicated repeatedly for each data source to which the server may connect.

[0152] Steps 520 through 524 can be replicated repeatedly for each SSAPP from which the server may receive a connection.

[0153] Steps 530 through 538 may be replicated for each connected SSAPP, or may be multiplexed such that Step 530 waits simultaneously for all connected SSAPPs at once, or any combination of these options.

[0154] It is understood that the methods exemplified in FIG. 4 and FIG. 5 may be modified to include additional capabilities, including: explicit stopping conditions for both the SSAPP and the data server; the ability of the server to wait passively for a data source to connect to the server; the ability of the server to actively connect to the SSAPP; the ability of the server to simultaneously manage connections to multiple data sources; the ability of the server to simultaneously manage connections to multiple SSAPPs; and the ability of the server to simultaneously receive data from multiple SSAPPs.

[0155] Referring to FIG. 6, in one embodiment, the data server's 603 ability to simultaneously manage connections to multiple SSAPPs 601 advantageously allows for SSAPPs 601 to communicate among one another through the server. Any information transmitted from SSAPP 601 to server 603 will be treated by the server as if the SSAPP 601 is a data source, and will propagate that data to any other SSAPPs 601 that are connected to the server and have subscribed to that data. Surprisingly, this effectively creates a network of SSAPPs intercommunicating in real time. In fact, server 603 may be used to enable communication among any number of client applications, using any combination of protocols that the server supports.

[0156] Referring to FIG. 7, in one embodiment, a substantial benefit of this invention is the ability to present data in SSAPP 701 that originates from sources that cannot otherwise be accessed via a network. In this embodiment, data originating via a non-networked protocol like DDE in DDE application 705, such as a Microsoft Word document, may be transmitted via a local or wide area network, which was not possible prior to the present invention. The invention allows any application to communicate in real time with the DDE data over any TCP/IP network, vastly broadening the scope of applications for spreadsheet data. This same functionality extends to any protocol that server 703 supports.

[0157] In one embodiment of the invention, a system implementing the methods of the invention comprises the following software applications:

[0158] 1. Cogent DataHub® (Cogent Real-Time Systems Inc.) acting as the data server

[0159] 2. Cogent DataHub (Cogent Real-Time Systems Inc.) acting as the web server

[0160] 3. Microsoft Excel™ (Microsoft Corp.) acting as the spreadsheet application

[0161] 4. DataHub API for .Net (Cogent Real-Time Systems Inc.) acting as a protocol implementation layer for Microsoft Excel

[0162] In addition, Cogent DataHub may send and receive data from a variety of data sources, including:

[0163] 1. Microsoft Excel™ (Microsoft Corp.) acting as a spreadsheet application

[0164] 2. OPC-DA server (various manufacturers) acting as a data communication interface

[0165] 3. OPC-UA server (various manufacturers) acting as a data communication interface

[0166] 4. OPC Xi server (various manufacturers) acting as a data communication interface

[0167] 5. ODBC server (various manufacturers) acting as a database interface

[0168] 6. DDE server (various manufacturers) acting as a data communication interface

[0169] Referring to FIG. 8, in one embodiment, depending on the particular implementation, zero or more data sources are attached to the Cogent DataHub.

[0170] The data server may be any application designed to collect data from a data source or act as a data source itself, as long as it also supplies a TCP/IP communication method that can be accessed by a constructed SSAPP.

[0171] A data source may be any application or system capable of producing real-time data that can be converted into a format suitable for representation within the server.

[0172] A data source may also be any application or system capable of producing non-real-time data that can be converted into a format suitable for representation within the server. The server can poll this data repeatedly or collect it by subscription to provide the data to a SSAPP even in the case that the original data is not real-time. For example, a database management system (DBMS) is generally not real-time, but the data can be polled repeatedly to create a periodically updating data set within the server, thus supplying a SSAPP with a pseudo-real-time view of the data within the DBMS.

[0173] The server and the data source may be combined into a single application, as may be the case with an OPC-UA server, or with an embedded device that offers access to its data via a TCP/IP connection.

[0174] A program developed using any compiled or interpreted computer language that can open and interact with a TCP/IP socket may be used in place of a SSAPP, which may or may not run within a web browser.

[0175] Referring to FIG. 9, in one embodiment, a mechanism for bi-directional streaming communication between a client and a server using two HTTP connections is

shown. It is assumed that the server is already running, and is listening for TCP connections on a port agreed upon by the server and the client. The TCP connection may use the WebSocket protocol or any other protocol over TCP. The handling of immaterial error conditions is omitted.

[0176] As shown in FIG. 9a, the client starts, or begins, its attempt to communicate with the server via a TCP connection (Step 900). First, the client opens a TCP socket (Step 902). The server holds this socket open.

[0177] Once the TCP socket has been successfully opened, the client may transmit configuration information and any data pending from a previous connection via the TCP socket (Step 906). The client may choose to send configuration information only on the first connection of the TCP socket for a given session. On subsequent TCP socket connections, there may be data that was previously undeliverable that is delivered at this point. If any commands or data were transmitted in Step 906, then the server processes them (Step 907) and generates zero or more responses that the client will receive in Step 908.

[0178] Once the connection is fully established, the client and server respectively enter wait states where they wait either for data arriving from the other, or for events that would cause them to emit data to the other. That is, the server may wait for data to arrive from the client, or for a locally generated (server-generated) event to occur (Step 919), as illustrated in FIG. 9c. Similarly, the client may wait for data to arrive from the server, or for a locally generated (client-generated) event to occur (Step 908), as illustrated in FIG. 9b.

[0179] Referring to FIG. 9b, the client will subsequently enter a loop where it waits for an event (Step 908) and processes it according to its type (Step 909). If the event is an event originating from the TCP socket, the client will first check whether that event is a socket error (Step 911). If so, the client closes its end of the TCP socket (Step 915), effectively closing the communication session with the server, and tries to create a new session with the server by returning to Step 902. If the event is an event originating from the TCP socket and is not an error, the client reads the available data from the socket (Step 912), and processes it in some manner (Step 913). This processing may generate a

result that can be transmitted back to the server via the TCP socket (Step 914). In the common case the processing (Step 913) includes extracting the contents of a spreadsheet cell or range of cells to produce a result. Advantageously, this processing may also involve further case-specific computation such as aggregation, transformation and filtering. Such processing may act to change the value that is written to server, or may produce a nil result. If the result is a nil result, nothing is transmitted back to the server. Alternatively, the client may optionally choose to transmit nothing back to the server due to a decision made during the processing step.

[0180] The result transmission via the TCP socket in Step 914 could fail. At least one failure mode is an unexpected network failure. Subsequent attempts to send data on the TCP socket will fail, so the client checks for this and other failures (Step 916). If a transmission failure occurs then the client may choose to mark the data for this transmission as pending (Step 917), and will close the TCP socket (Step 918). The client will then attempt to re-open the TCP socket by returning to step 902.

[0181] If the client generates an event internally, or as a result of user activity, a timer, or other external stimulus that requires communication with the server in Step 909, then the client will perform whatever processing is required to compute data to be transmitted to the server (Step 910). This data is effectively the result data of the event, which is then transmitted to the server (Step 914) and follows the same transmission method as for result data from a socket event.

[0182] The client may loop indefinitely, establishing the connection to the server and re-establishing that connection should it fail. The client may choose to signal failures and reconnection states to a user or other program, or may simply reconnect to the server without notification.

[0183] After Step 907, the server will also enter a loop where it waits for an event in Step 919 and processes it according to its type (Step 920), as illustrated in FIG 9c. If the event is an event originating from the TCP socket, the server will first check whether that event is a socket error (Step 922). If the event is a socket error, the server closes its end of the TCP socket (Step 923), effectively requesting that the client re-establish its TCP socket. If the event is an event originating from the TCP socket and is not an error, the

server reads the available data from the socket (Step 924), and processes it in some manner (Step 925). In the common case this processing includes writing the data to the contents of a spreadsheet cell or range of cells. Advantageously, this processing may also involve case-specific computation such as aggregation, transformation and filtering prior to being written to the cell or range. Such processing may act to change the value that is written to a cell or range, or may act to inhibit any change to a cell or range. This processing may generate a result that can be transmitted back to the client via the TCP socket (Step 926). The result may be a nil result, in which case nothing is transmitted back to the client. Alternatively, the server may optionally choose to transmit nothing back to the client.

[0184] The result transmission via the TCP socket in Step 926 could fail. The server checks for transmissions failures (Step 927), and if a transmission failure occurs then the server will close the TCP socket, effectively ending the session (Step 928). The server does not attempt to re-establish a connection with the client, but rather waits for the client to re-establish the connection if necessary. This effectively will return the client/server system to Step 902.

[0185] If the server generates an event internally, or as a result of user activity, a timer, another connected client, data from a data source, or other external stimulus that requires communication with the client in Step 919, then the server will perform whatever processing is required to compute data to be transmitted to the client (Step 921). This data is effectively the result data of the event, which is then transmitted to the client (Step 926) and follows the same transmission method as for result data from a socket event.

[0186] As would be readily appreciated by a person skilled in the art, there can be errors handling in Steps 902 through 907 that would close any open sockets and re-start the connection process at Step 902. Although these errors handling have not been illustrated in FIG 9 for clarity, they would be included in a preferred embodiment. As will be appreciated by a person skilled in the art, the client may choose to terminate the connection (e.g. closing the browser client), and any such termination may be handled by the server in the same manner as a transmission error. That is, the server will close the TCP socket, terminate the session and wait for a client to connect (Step 900).

[0187] The client and server can implement wait states in any number of ways, including creating a new process thread to perform a synchronous wait or performing a multiple-event wait in a single thread. These are implementation details that will depend on choices made during the client and server implementations, but do not depart from the scope of the present invention.

[0188] Advantageously, the present invention is operable on any device that is capable of opening a TCP socket. For example, the client/server implementation may comprise multiple servers propagating data in real-time over a network or the Internet, optionally in a secure manner, without any major and therefore costly changes in existing infrastructure (e.g., security policies, firewalls, software, hardware, etc.). The present invention may be implemented, for example, by way of software code built-in to the SSAPP, a software add-on, a software plug-in, a scripting language, a separate software application, or a combination thereof.

[0189] Referring to FIG. 10, a prior art system for providing direct communication between a server 1002 and a client 1001 separated by a network 1007 is shown, as envisioned by a traditional SCADA system. In this example implementation, the server 1002 and client 1001 are situated behind firewalls 1003, 1004 to protect from unauthorized access from any third parties (not shown) on network 1007. Arrow 1006 symbolically shows client 1001 originating a request for data located on server 1002, and arrow 1005 shows server 1002 waiting for incoming requests from client 1001. In order for the client 1001 to access data on server 1002, the server's firewall 1003 must be configured to allow an incoming connection from outside firewall 1003. In this example, server 1002 is exposed to incoming requests originating from network 1007, and therefore firewall 1003 provides a point of attack or vulnerability that may be exploited.

[0190] When the network 1007 is a private network, the assumption is that a concerted malicious attack on the server 1002 through the open port on the firewall 1003 is unlikely and an acceptable risk. However, when the network 1007 is a public network (e.g. the Internet), the likelihood of a concerted attack on the server 1002 is high, and the risk is unacceptable.

[0191] Referring to FIG. 11, in one embodiment, a system for providing secure, real-time data over a network 1107 is shown. In contrast to the prior art system shown in FIG. 10, the novel system shown in FIG. 11 illustrates the differences between a direct client/server connection, and a cloud-based system as provided by the present invention.

[0192] In the present invention, a cloud server 1100 is situated away from both the server 1102, acting as an authoritative client, and the user client 1101 (non-authoritative). Both the server 1102 and client 1101 initiate outbound connections, as shown by arrows 1105, 1106, to the cloud server 1100, through their respective firewalls 1103, 1104. The firewalls 1103, 1104 are not required to provide any open inbound ports. This configuration is equally secure regardless of whether the network 1107 between server 1102 and client 1101 is private or public.

[0193] The server or authoritative client 1102 decides what data to send to the cloud server 1100. Further, each server 1102 can set each data stream to be one-way or two-way, and can send some or all of its data, depending on its needs. Preferably, this configuration is set by the customer at the authoritative client by way of a connector, provided in the form of a software application described above (the DataHub). Accordingly, configuration may be set entirely in the connector, not at the cloud server, thereby optionally providing an additional layer of security should the cloud server be compromised.

[0194] Also shown in FIG. 11 are multiple servers 1102 acting as authoritative clients, which is an unexpected result made possible by the present invention. Specifically, the present invention allows multiple servers 1102 to act as authoritative clients for their own data sets (not illustrated), and aggregated at the cloud server 1102 for efficient consumption by one or more clients 1101. Surprisingly, this allows for servers 1102 to be located in physically separate locations from each other (e.g. in different plant facilities located around the world), while producing a unified data set at the cloud server such that the client(s) 1101 see the unified data set as if it were produced from a single system. That is, distributed system appears to the client as a single system. Such functionality is not possible in traditional SCADA systems. This is advantageous at least to provide convenience and the ability to simultaneously monitor an entire network of systems, and to share data amongst the servers 1102. Examples of such

applications are broad, for instance coordinated operation of vehicle fleets, networks of devices, global financial trading systems and redundant parallel systems.

[0195] When an authoritative client (server 1102) connects to the cloud server 1100, it does not know whether the server 1100 contains the data items that the authoritative client 1102 intends to publish, and where those data items do exist in the server 1100, the client 1101 does not know their current values. Typically a client will rely on a server to provide the items and their values, but in the case of an authoritative client 1102 it is the client 1102 which must provide the items and their values. Thus, upon an initial connection, the authoritative client 1102 must emit its entire data set and current values, ignoring and overwriting any values already present in the server 1100. The current invention optionally provides this behaviour, allowing any client to select whether it will be authoritative for a particular data set.

[0196] Similarly, when an authoritative client 1102 disconnects from the cloud server 1100, it must be able to inform other connected clients that the authoritative source of data is no longer providing data. The authoritative client 1102 informs the server that it is authoritative, and additionally instructs the server 1100 to alter the properties of the data items in the server when the client disconnects to indicate that those data items are "not connected". The cloud server 1100 must co-operate in this process, since the authoritative client has already disconnected before the data items are marked as "not connected", and the server must propagate this change of status (sometimes referred to as "quality") to other connected clients.

[0197] The combination of these important features ensures that the data on the server 1100 is either consistent with the data on the authoritative client 1102, or it is in a known error state to indicate that the authoritative client 1102 is not connected to the cloud server 1100.

[0198] Referring to FIG. 12, in another embodiment, a system is shown that is similar to that shown in FIG. 11, but in a more visually descriptive manner. In particular, exemplar types of servers may comprise embedded devices, SCADA systems or various connected consumer products, all producing, propagating, sending and/or receiving data (some in real time, some not). As illustrated in FIG. 12, these devices are behind

firewalls without open incoming firewall ports, thereby eliminating direct attacks from would-be hackers on the public network (not shown). On the same public network, a secure cloud server may receive outbound connections initiated by the device behind the firewalls, as illustrated symbolically by the large arrow across the firewall to the cloud server. Within the large arrow, data may be sent safely back to the devices.

[0199] Also shown in FIG. 12 is another aspect of the present invention, whereby the cloud server may employ methods described above to send data to SSAPPs for predictive maintenance or HMI displays, for data analysis (e.g. generating key performance indices), or to databases, or to provide alerts.

[0200] In another embodiment (not shown), a firewall is not provided in front of the servers, clients or devices on the network, where the servers/clients/devices are configured to reject inbound connection requests. This is to be contrasted with employing firewalls that ignore inbound connection requests to the server/client/device behind the firewalls. In this embodiment, the operating system of the server/client/device responds to inbound connection requests, after a fashion, with a 'nobody is listening' response. While this configuration may be less secure than employing firewalls, there are situations where the server/client/device is resource constrained to the point where it is not possible to include a firewall. Surprisingly, the present invention thus provides a method to provide a more secure method of communication with resource-constrained devices.

[0201] It should be understood that when referring to a network residing between, for instance, a server and a client, the network itself may comprise a series of network connections; that is, there is no implication of a direct connection. Similarly, any server, client or device 'on' the Internet is understood to mean that the server, client or device is connected to a network connection that is accessible to the Internet.

[0202] It is also understood that an authority on a data set, or an authoritative holder of a data set, refers to the originator of the data set, and all other recipients of the data set hold non-authoritative copies. In the present invention, a server, client or device can inherit authority from another server, client or device; for example, the cloud server may act as an authority on a data set for another client/end-user device; the client/end-user

device sees the cloud server as the authority on the data set, but unknown to the client/end-user device, the cloud server may be propagating the data from a "true" authoritative client/end-user device connected to the cloud server. It is appreciated that the present invention allows for a myriad of combinations of servers, clients, and devices interconnected and inheriting authority over multiple data sets shared amongst them.

CLAIMS

1. A method for real-time interaction with a spreadsheet application (SSAPP), the method comprising:

obtaining data propagated from a data source;

creating a persistent connection with a SSAPP;

subscribing the SSAPP to subscribed data, wherein the subscribed data includes at least some of the data obtained from the data source;

propagating the subscribed data to the SSAPP through the persistent connection as the subscribed data is obtained;

operating in a non-authoritative configuration, wherein operating in the non-authoritative configuration further includes:

storing a data set

receiving a connection from an authoritative client device;

responsive to receiving the connection, obtaining an authoritative data set from the authoritative client device;

storing the authoritative data set

determining that the authoritative client device disconnected; and

responsive to the authoritative client device disconnecting, notifying the SSAPP that the authoritative client device is no longer providing data.

2. The method of claim 1, further comprising:

sending the subscribed data to the SSAPP via a bi-directional Transmission Control Protocol (TCP) connection.

3. The method of claim 1 or 2, further comprising:

obtaining, from the SSAPP, a command; and

executing the command.

4. The method of claim 3, wherein executing the command includes: (i) altering a behavior based on the command, (ii) altering how the data propagated from the data source is obtained based on the command, or (iii) any combination of (i) and (ii).

5. The method of any one of claims 1 to 4, wherein obtaining the data propagated from the data source further includes:

obtaining the data through at least one intermediate component.

6. The method of any one of claims 1 to 5, further comprising:

obtaining a change request from the SSAPP;

transmitting the change request to the data source;

obtaining updated data propagated from the data source based on the change request; and

propagating the updated data to the SSAPP through the persistent connection as the updated data is obtained.

7. The method of any one of claims 1 to 6,

wherein storing the authoritative data set includes one or both of:

(i) overwriting a portion of the stored data set that conflicts with the authoritative data set; and

(ii) ignoring a portion of the stored data set that conflicts with the authoritative data set.

8. The method of any one of claims 1 to 7, wherein notifying the SSAPP that the authoritative client device is no longer providing data includes:

marking data items as not connected.

9. The method of any one of claims 1 to 8, further comprising:

treating the SSAPP as both an authoritative data source and a non-authoritative data source.

10. The method of claim 9,

wherein the SSAPP is a first SSAPP;

wherein treating the first SSAPP as an authoritative data source includes:

obtaining SSAPP-originated data from the first SSAPP through the persistent connection; and

propagating the SSAPP-originated data to a second SSAPP; and

wherein treating the first SSAPP as a non-authoritative data source includes:

obtaining second SSAPP-originated data from the second SSAPP; and

propagating the second SSAPP-originated data to the first SSAPP.

11. A non-transitory computer-readable medium comprising instructions that, when executed by a processor, cause performance of a method comprising:

obtaining, at a spreadsheet application (SSAPP), subscribed data from a server through a persistent connection between the SSAPP and the server via a TCP socket, the subscribed data originating from a data source;

performing an action on the subscribed data, wherein the action comprises presenting a representation based on the subscribed data to a user;

receiving a notification that the subscribed data has been updated;

obtaining updated subscribed data, the updated subscribed data being based on an authoritative data set from an authoritative client device;

performing the action on the updated subscribed data;

extracting contents from a set of cells comprising one or more cells;

processing the contents from the set of cells to produce a result;

transmitting the result to the server via the persistent connection; and

receiving a change of status indication propagated from the server indicating that the authoritative client device is not connected to the server.

12. The non-transitory computer-readable medium of claim 11, wherein the method further comprises:

creating the persistent connection between the SSAPP and the server; and

subscribing the SSAPP to subscribed data from the server.

13. The non-transitory computer-readable medium of claim 11 or 12, wherein the action further comprises:

(i) visually representing at least some of the subscribed data; (ii) performing a calculation on the subscribed data; (iii) producing new data based on the subscribed data; (iv) modifying the subscribed data; (v) storing the subscribed data; (vi) generating an audible indication; (vii) executing a script; (viii) propagating data to the server; (ix) performing a user-visible programmatic response; (x) performing a non-user-visible programmatic response; or (xi) any combination of (i)-(x).

14. The non-transitory computer-readable medium of any one of claims 11 to 13, wherein the method further comprises:

sending an instruction from the SSAPP to the server through the persistent connection, the instruction comprising (i) an instruction to alter a behavior of the server, (ii) an instruction to alter how the subscribed data is propagated from the data source, or (iii) any combination of (i) and (ii); and

wherein the updated subscribed data comprises an update based on the instruction.

15. A system for real-time interaction with a spreadsheet application (SSAPP), the system comprising:

a processor;

a memory comprising instructions that, when executed by the processor, cause performance a method comprising:

obtaining data propagated from a data source;

creating a persistent connection with a SSAPP;

subscribing the SSAPP to subscribed data, wherein the subscribed data includes at least some of the data obtained from the data source; and

propagating the subscribed data to the SSAPP through the persistent connection as the subscribed data is obtained;

operating in a non-authoritative configuration, wherein operating in the non-authoritative configuration further includes:

storing a data set

receiving a connection from an authoritative client device;

responsive to receiving the connection, obtaining an authoritative data set from the authoritative client device;

storing the authoritative data set;

determining that the authoritative client device disconnected; and

responsive to the authoritative client device disconnecting, propagating to the SSAPP a change of status indicating that the authoritative client device is no longer providing data.

16. The system of claim 15, wherein the method further includes:

obtaining, from the SSAPP, a command; and

executing the command, wherein executing the command includes: (i) altering a behavior based on the command, (ii) altering how the data propagated from the data source is obtained based on the command, or (iii) any combination of (i) and (ii).

17. The system of claim 15 or 16, wherein the method further includes:

obtaining a change request from the SSAPP;

transmitting the change request to the data source;

obtaining updated data propagated from the data source based on the change request; and

propagating the updated data to the SSAPP through the persistent connection as the updated data is obtained.

18. The system of any one of claims 15 to 17, wherein storing the authoritative data set includes one or both of:

(i) overwriting a portion of the stored data set that conflicts with the authoritative data set; and

(ii) ignoring a portion of the stored data set that conflicts with the authoritative data set.

19. The system of any one of claims 15 to 18, wherein operating in the non-authoritative configuration further includes:

receiving an instruction from the authoritative client device to alter properties of the authoritative data set when the authoritative client device disconnects to indicate that the authoritative client device is not connected.

20. The system of any one of claims 15 to 19, wherein the SSAPP is a first SSAPP; and

wherein the method further includes:

obtaining SSAPP-originated data from the first SSAPP through the persistent connection; and

propagating the SSAPP-originated data to a second SSAPP.

21. A method for real-time interaction with a spreadsheet application (SSAPP), the method comprising:

obtaining data propagated from a data source;

creating a persistent connection with a SSAPP;

subscribing the SSAPP to subscribed data, wherein the subscribed data includes at least some of the data obtained from the data source;

propagating the subscribed data to the SSAPP through the persistent connection as the subscribed data is obtained;

obtaining a change request from the SSAPP through the persistent connection;

transmitting the change request to the data source;

obtaining updated data propagated from the data source based on the change request;

propagating the updated data to the SSAPP through the persistent connection as the updated data is obtained;

operating in a non-authoritative configuration, wherein operating in the non-authoritative configuration further includes:

storing a data set;

receiving a connection from an authoritative client device;

responsive to receiving the connection, obtaining an authoritative data set from the authoritative client device over the connection; and

storing the authoritative data set,

wherein storing the authoritative data set includes at least one of: (i) overwriting a portion of the stored data set that conflicts with the authoritative data set and (ii) ignoring a portion of the stored data set that conflicts with the authoritative data set, and

wherein operating in the non-authoritative configuration further includes:

determining whether the authoritative client device has disconnected; and

responsive to the authoritative client device disconnecting, notifying one or more other connected client devices that the authoritative client device is no longer providing data.

22. The method of claim 21, further comprising:

sending the subscribed data to the SSAPP via a bi-directional Transmission Control Protocol (TCP) connection.

23. The method of claim 21 or 22, further comprising:

obtaining, from the SSAPP, a command; and

executing the command.

24. The method of claim 23, wherein executing the command includes: altering a behavior based on the command.

25. The method of claim 23 or 24, wherein executing the command includes:

altering how the data propagated from the data source is obtained based on the command.

26. The method of any one of claims 21 to 25, wherein obtaining the data propagated from the data source further includes:

obtaining the data through at least one intermediate component.

27. The method of any one of claims 21 to 26, further comprising:

treating the SSAPP as both an authoritative data source and a non-authoritative data source.

28. The method of claim 27,

wherein the SSAPP is a first SSAPP;

wherein treating the first SSAPP as an authoritative data source includes:

obtaining SSAPP-originated data from the first SSAPP through the persistent connection; and

propagating the SSAPP-originated data to a second SSAPP; and

wherein treating the first SSAPP as a non-authoritative data source includes:

obtaining second SSAPP-originated data from the second SSAPP; and

propagating the second SSAPP-originated data to the first SSAPP.

29. A system for real-time interaction with a spreadsheet application (SSAPP), the system comprising:

a processor;

a memory comprising instructions that, when executed by the processor, cause the processor to:

obtain data propagated from a data source;

create a persistent connection with a SSAPP;

subscribe the SSAPP to subscribed data, wherein the subscribed data includes at least some of the data obtained from the data source;

propagate the subscribed data to the SSAPP through the persistent connection as the subscribed data is obtained;

obtain a change request from the SSAPP through the persistent connection;

transmit the change request to the data source;

obtain updated data propagated from the data source based on the change request;

propagate the updated data to the SSAPP through the persistent connection as the updated data is obtained;

operate in a non-authoritative configuration by:

storing a data set;

receiving a connection from an authoritative client device;

responsive to receiving the connection, obtaining an authoritative data set from the authoritative client device over the connection; and

storing the authoritative data set,

wherein storing the authoritative data set includes at least one of: (i) overwriting a portion of the stored data set that conflicts with the authoritative data set and (ii) ignoring a portion of the stored data set that conflicts with the authoritative data set, and

wherein operating in the non-authoritative configuration further includes:

determining whether the authoritative client device has disconnected; and

responsive to the authoritative client device disconnecting, notifying one or more other connected clients that the authoritative client device is no longer providing data.

30. The system of claim 29, wherein the instructions further cause the processor to:

obtain, from the SSAPP, a command; and

execute the command, wherein executing the command includes: altering a behavior based on the command.

31. The system of claim 29 or 30, wherein the SSAPP is a first SSAPP; and

wherein the instructions further cause the processor to:

obtain SSAPP-originated data from the first SSAPP through the persistent connection; and

propagate the SSAPP-originated data to a second SSAPP.

32. The system of any one of claims 29 to 31, wherein the instructions further cause the processor to:

obtain, from the SSAPP, a command; and

execute the command, wherein executing the command includes altering how the data propagated from the data source is obtained based on the command.

33. A non-transitory computer-readable medium having stored thereon one or more sequences of instructions for causing one or more processors to perform:

obtaining data propagated from a data source;

creating a persistent connection with a SSAPP;

subscribing the SSAPP to subscribed data, wherein the subscribed data includes at least some of the data obtained from the data source;

propagating the subscribed data to the SSAPP through the persistent connection as the subscribed data is obtained;

obtaining a change request from the SSAPP through the persistent connection;

transmitting the change request to the data source;

obtaining updated data propagated from the data source based on the change request;

propagating the updated data to the SSAPP through the persistent connection as the updated data is obtained;

operating in a non-authoritative configuration, wherein operating in the non-authoritative configuration further includes:

storing a data set;

receiving a connection from an authoritative client device;

responsive to receiving the connection, obtaining an authoritative data set from the authoritative client device over the connection; and

storing the authoritative data set,

wherein storing the authoritative data set includes at least one of: (i) overwriting a portion of the stored data set that conflicts with the authoritative data set and (ii) ignoring a portion of the stored data set that conflicts with the authoritative data set, and

wherein operating in the non-authoritative configuration further includes:

determining whether the authoritative client device has disconnected; and

responsive to the authoritative client device disconnecting, notifying one or more other connected client devices that the authoritative client device is no longer providing data.

34. The non-transitory computer-readable medium of claim 33, further having stored thereon a sequence of instructions for causing the one or more processors to perform:

sending the subscribed data to the SSAPP via a bi-directional Transmission Control Protocol (TCP) connection.

35. The non-transitory computer-readable medium of claim 33 or 34, further having stored thereon a sequence of instructions for causing the one or more processors to perform:

obtaining, from the SSAPP, a command; and

executing the command.

36. The non-transitory computer-readable medium of claim 35, wherein executing the command includes altering a behavior based on the command.

37. The non-transitory computer-readable medium of any one of claims 33 to 36, wherein obtaining the data propagated from the data source further includes obtaining the data through at least one intermediate component.

38. The non-transitory computer-readable medium of any one of claims 33 to 37, further having stored thereon a sequence of instructions for causing the one or more processors to perform:

treating the SSAPP as both an authoritative data source and a non-authoritative data source.

39. The non-transitory computer-readable medium of any one of claims 33 to 38,

wherein the SSAPP is a first SSAPP;

wherein treating the first SSAPP as an authoritative data source includes:

obtaining SSAPP-originated data from the first SSAPP through the persistent connection; and

propagating the SSAPP-originated data to a second SSAPP; and

wherein treating the first SSAPP as a non-authoritative data source includes:

obtaining second SSAPP-originated data from the second SSAPP; and

propagating the second SSAPP-originated data to the first SSAPP.

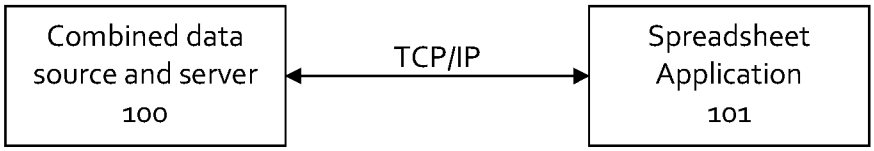


FIG. 1

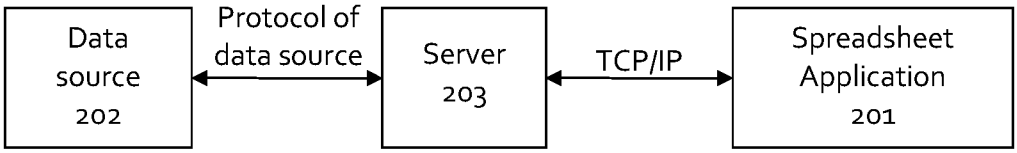


FIG. 2

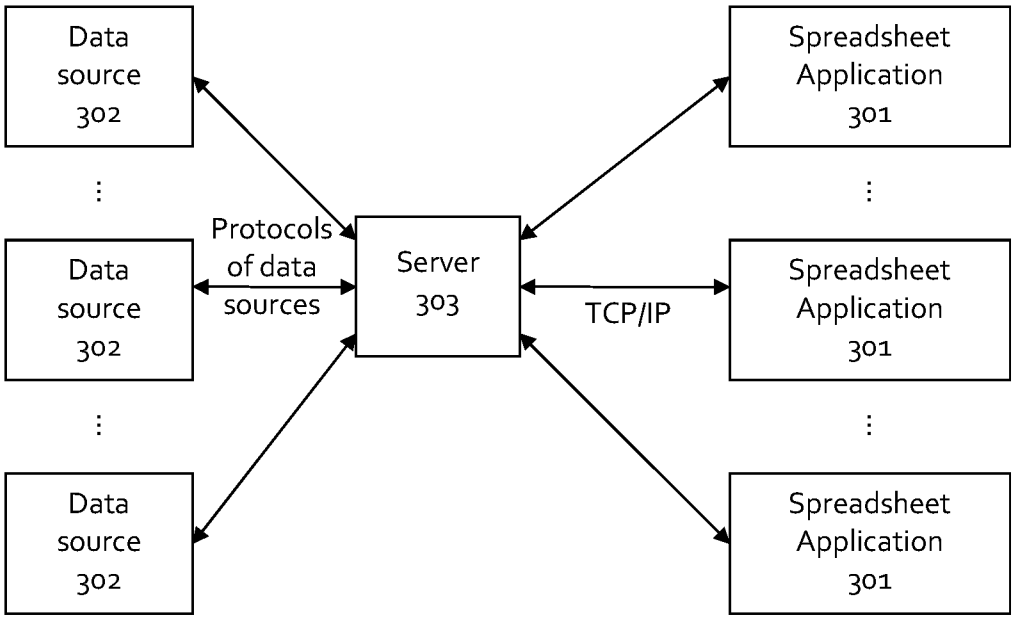


FIG. 3

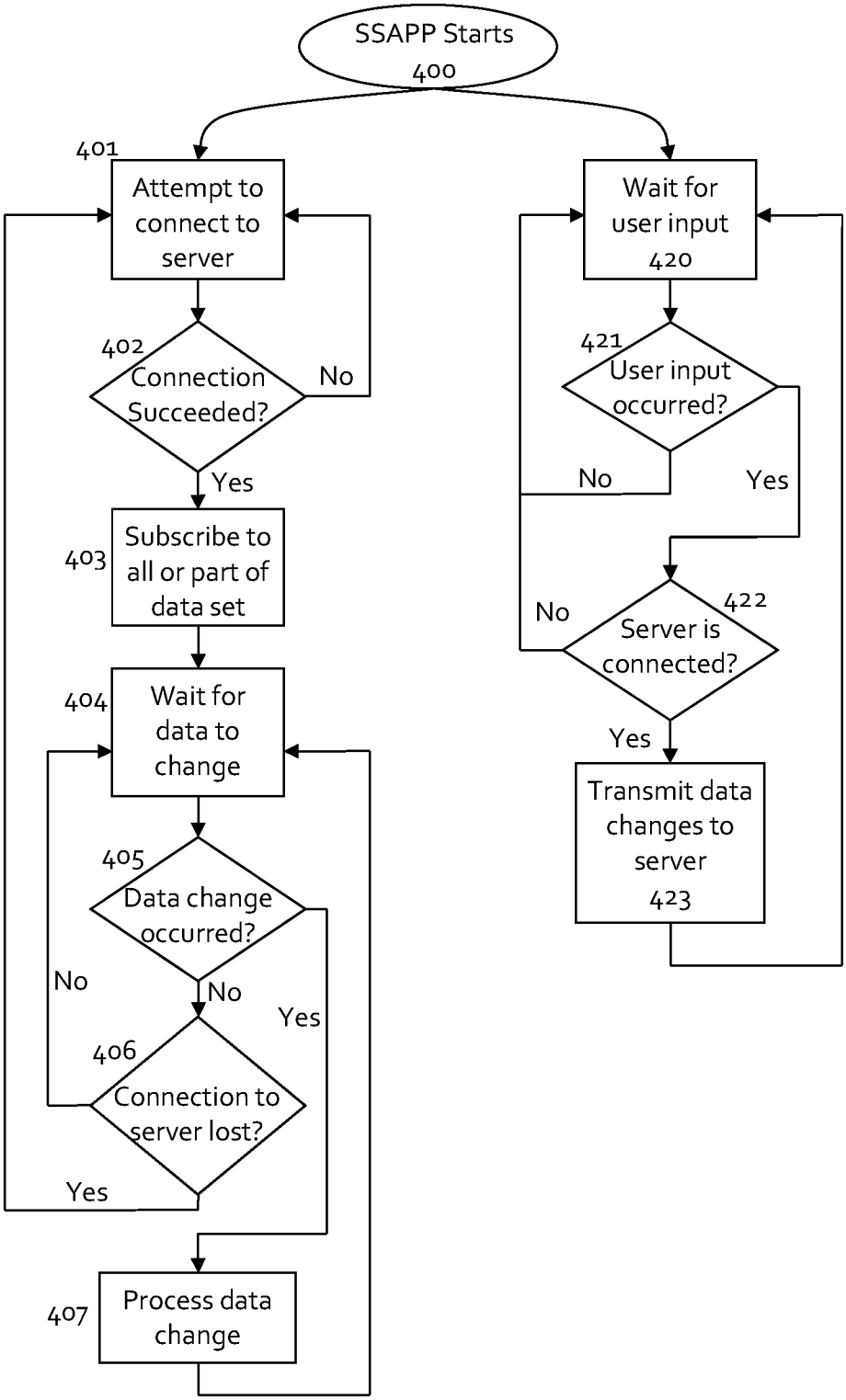

SSAPP Starts
400
401
Attempt to connect to server
402
Connection Succeeded?
No
Yes
403
Subscribe to all or part of data set
404
Wait for data to change
405
Data change occurred?
No
Yes
406
Connection to server lost?
No
Yes
407
Process data change
Wait for user input
420
421
User input occurred?
No
Yes
422
Server is connected?
No
Yes
Transmit data changes to server
423

FIG. 4

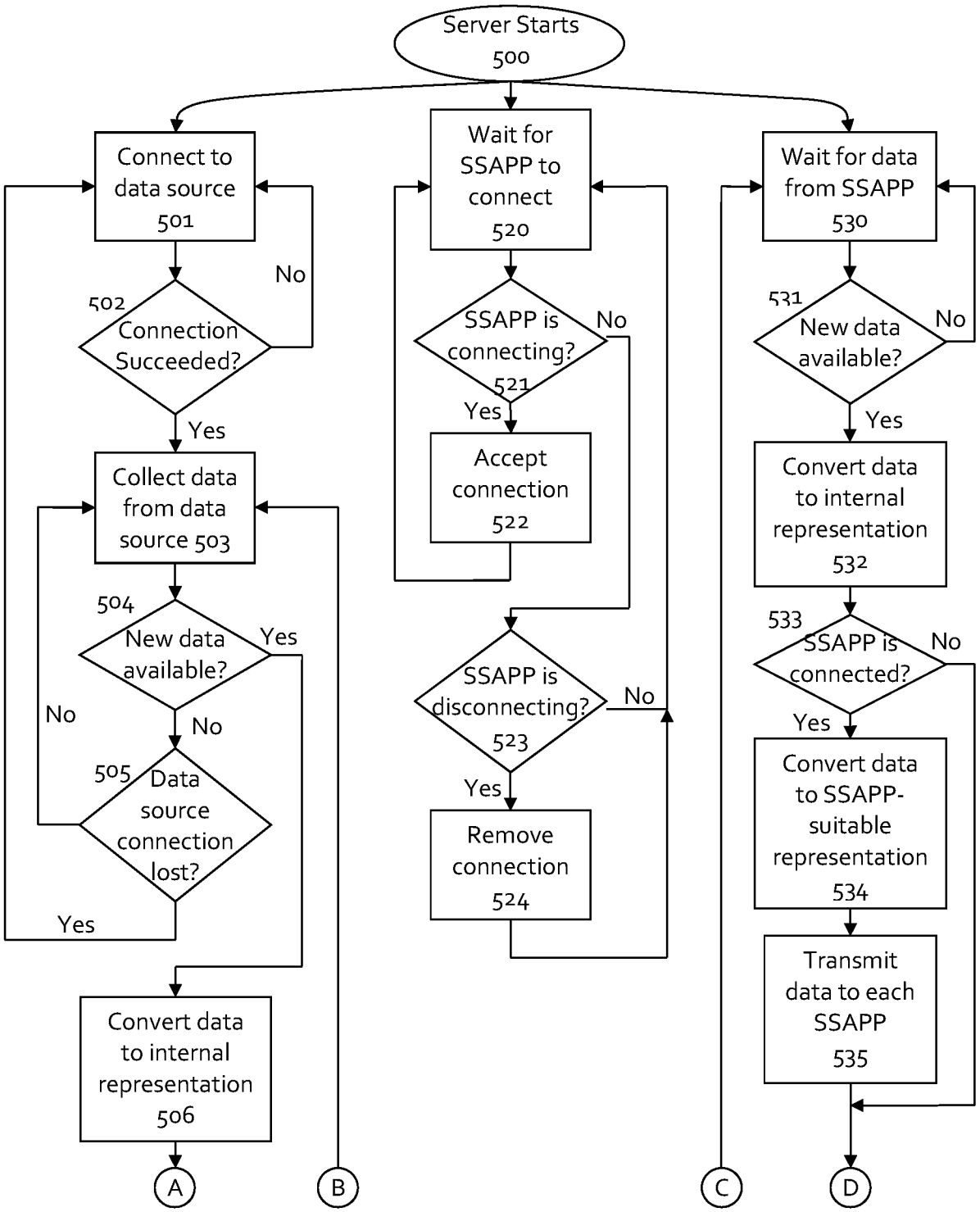

Server Starts
500
Connect to data source
501
502
Connection Succeeded?
No
Yes
Collect data from data source 503
504
New data available?
Yes
No
No
505
Data source connection lost?
Yes
Convert data to internal representation
506
A
B
Wait for SSAPP to connect
520
SSAPP is connecting?
521
No
Yes
Accept connection
522
SSAPP is disconnecting?
523
No
Yes
Remove connection
524
Wait for data from SSAPP
530
531
New data available?
No
Yes
Convert data to internal representation
532
533
SSAPP is connected?
No
Yes
Convert data to SSAPP-suitable representation
534
Transmit data to each SSAPP
535
C
D

FIG. 5a

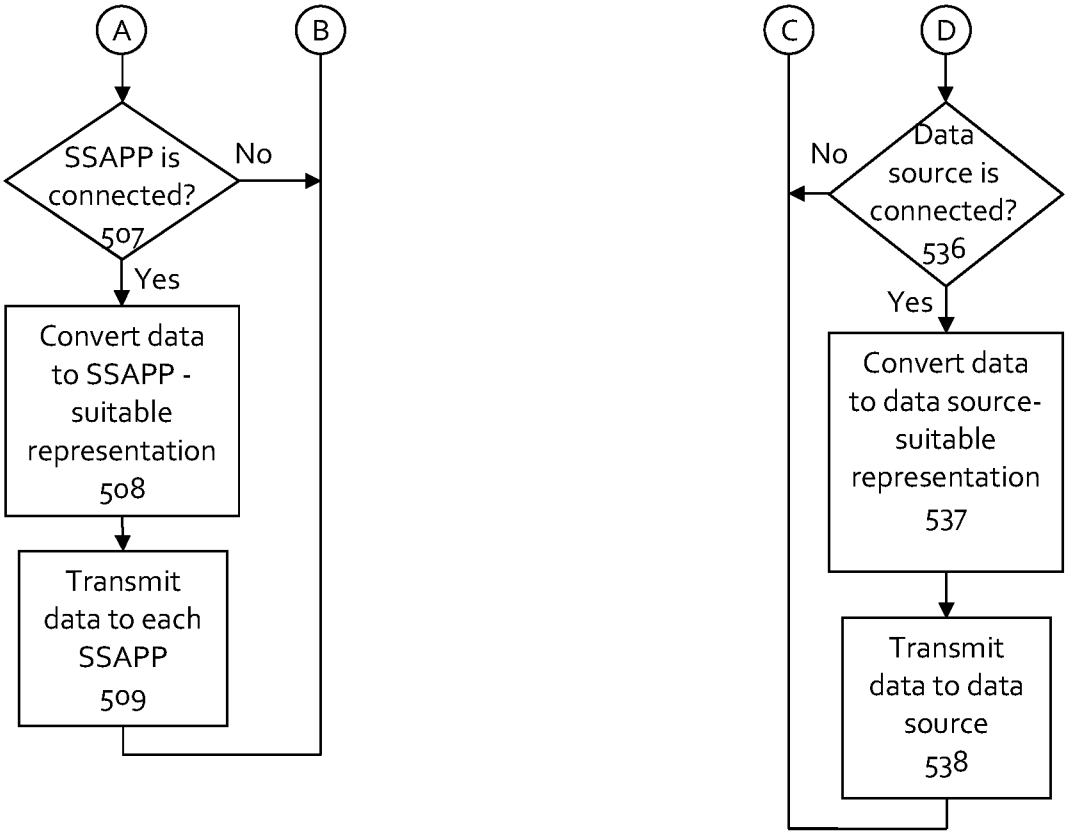

A
B
SSAPP is connected? 507
No
Yes
Convert data to SSAPP - suitable representation 508
Transmit data to each SSAPP 509
C
D
Data source is connected? 536
No
Yes
Convert data to data source-suitable representation 537
Transmit data to data source 538

FIG. 5b

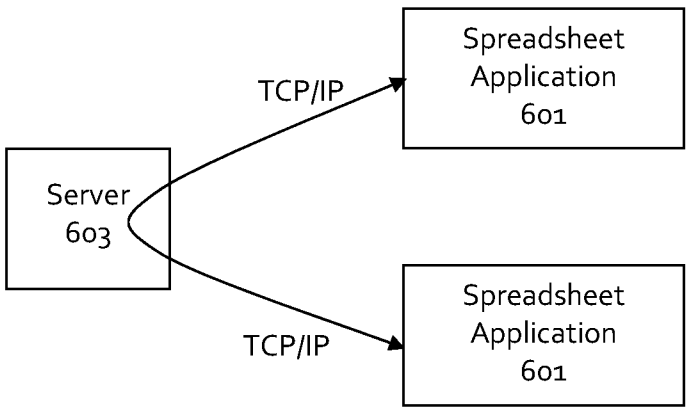

Server 603
TCP/IP
TCP/IP
Spreadsheet Application 601
Spreadsheet Application 601

FIG. 6

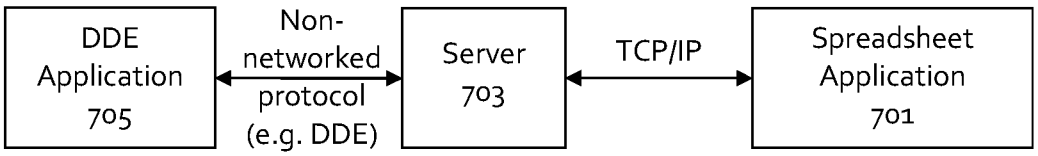

DDE Application 705
Non-networked protocol (e.g. DDE)
Server 703
TCP/IP
Spreadsheet Application 701

FIG. 7

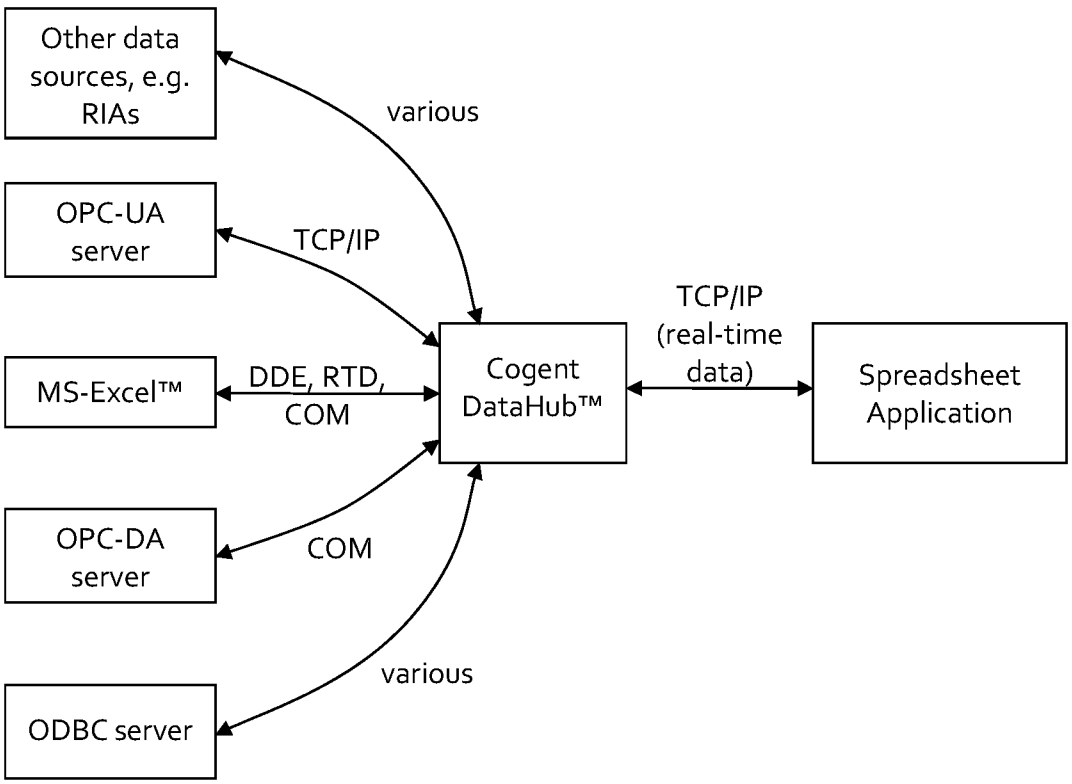

Other data sources, e.g. RIAs
various
OPC-UA server
TCP/IP
MS-Excel™
DDE, RTD, COM
Cogent DataHub™
TCP/IP (real-time data)
Spreadsheet Application
OPC-DA server
COM
ODBC server
various

FIG. 8

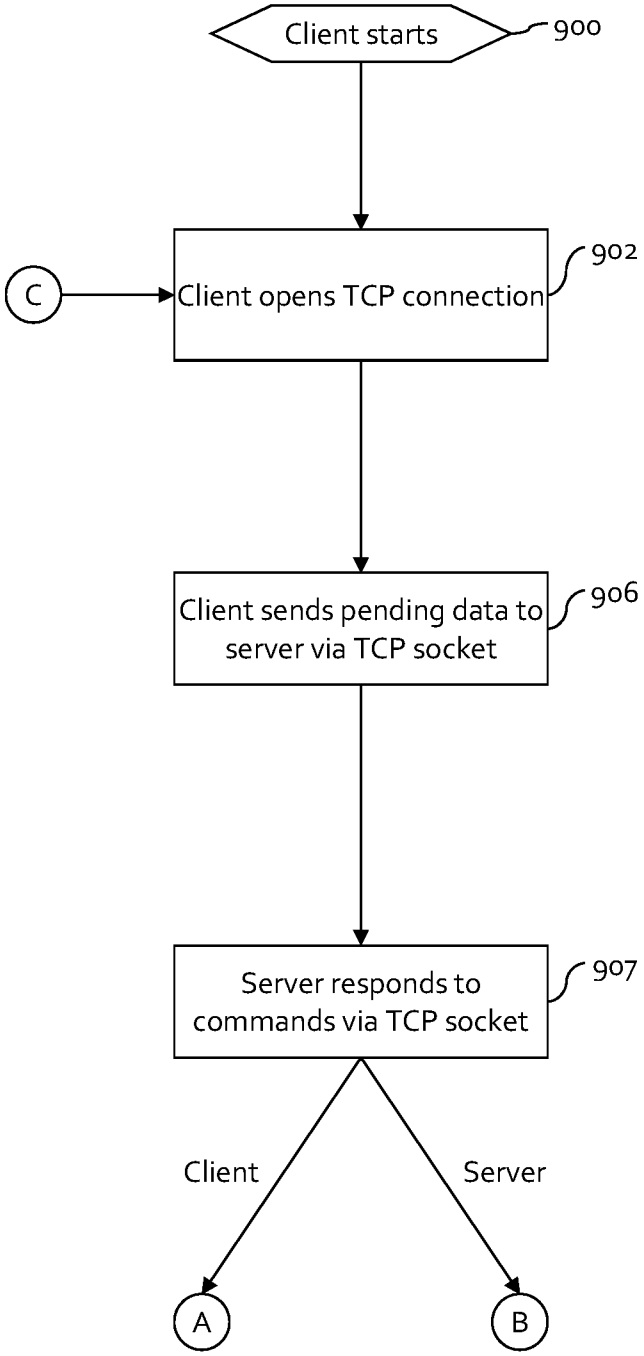

Client starts
900
C
Client opens TCP connection
902
Client sends pending data to server via TCP socket
906
Server responds to commands via TCP socket
907
Client
Server
A
B

FIG. 9a

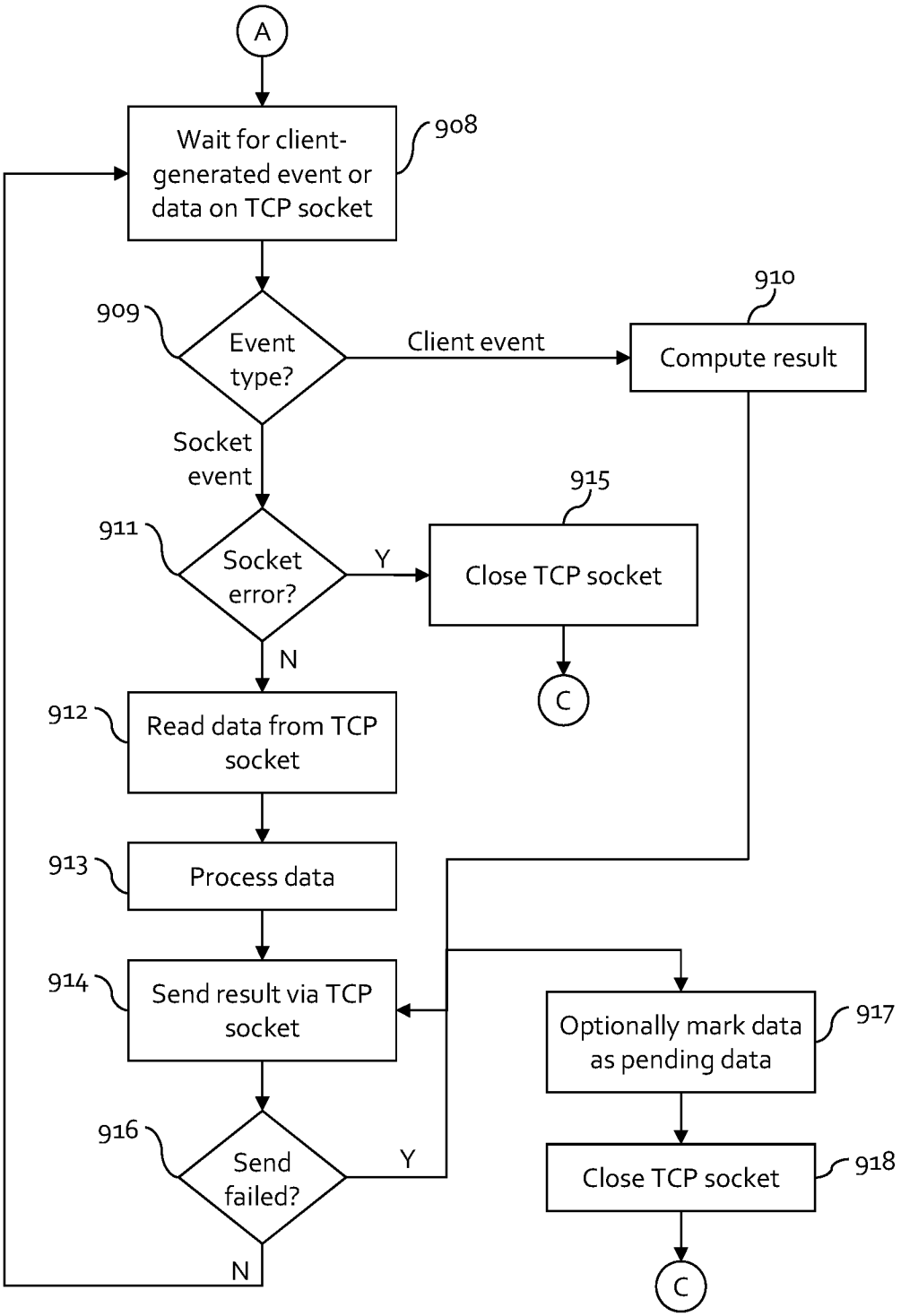

A
Wait for client-generated event or data on TCP socket
908
909
Event type?
Client event
910
Compute result
Socket event
911
Socket error?
Y
915
Close TCP socket
C
N
912
Read data from TCP socket
913
Process data
914
Send result via TCP socket
916
Send failed?
Y
N
Optionally mark data as pending data
917
Close TCP socket
918
C

FIG. 9b

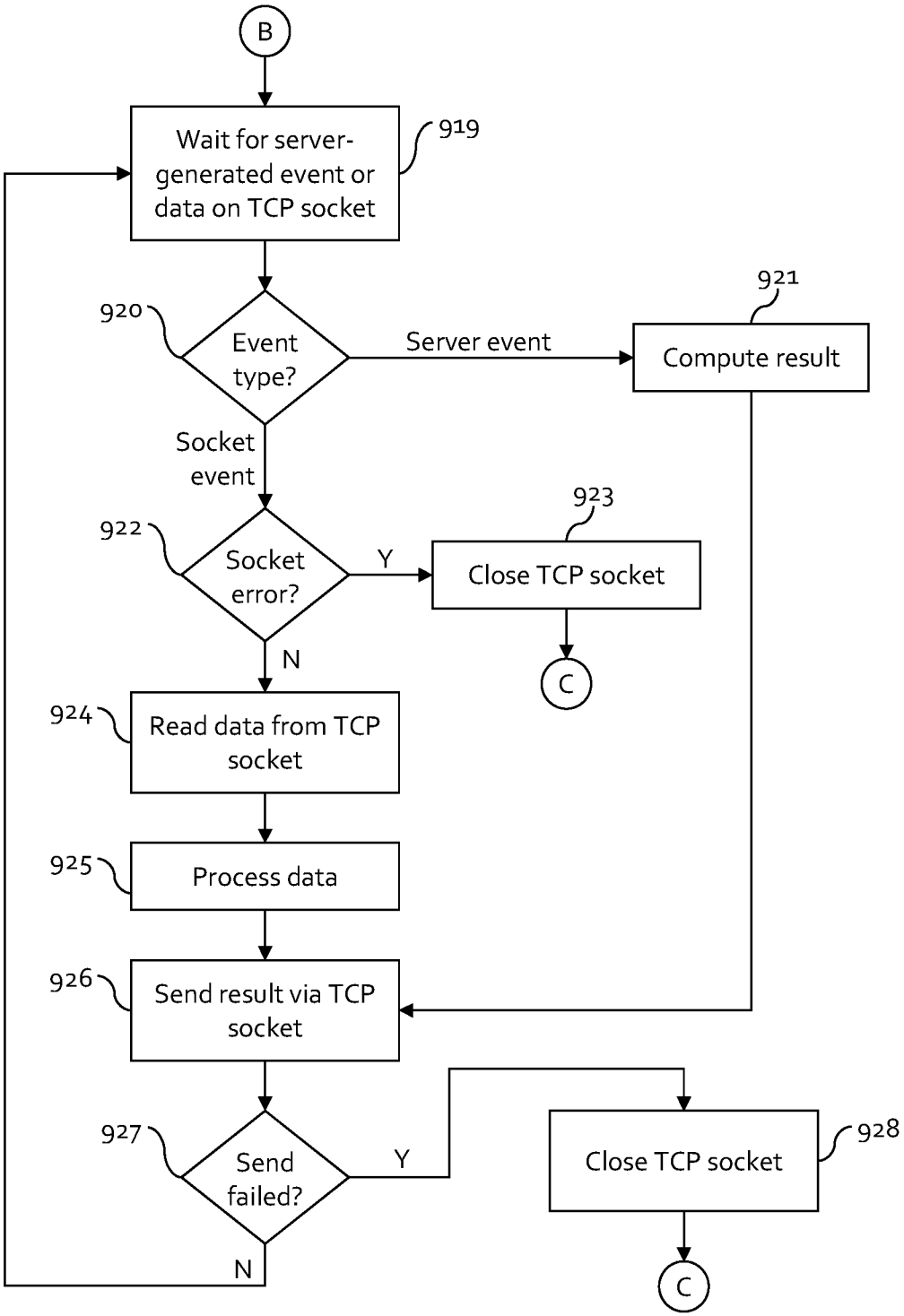

B
Wait for server-generated event or data on TCP socket
919
Event type?
920
Server event
Compute result
921
Socket event
Socket error?
922
Y
Close TCP socket
923
C
N
Read data from TCP socket
924
Process data
925
Send result via TCP socket
926
Send failed?
927
Y
Close TCP socket
928
C
N

FIG. 9c

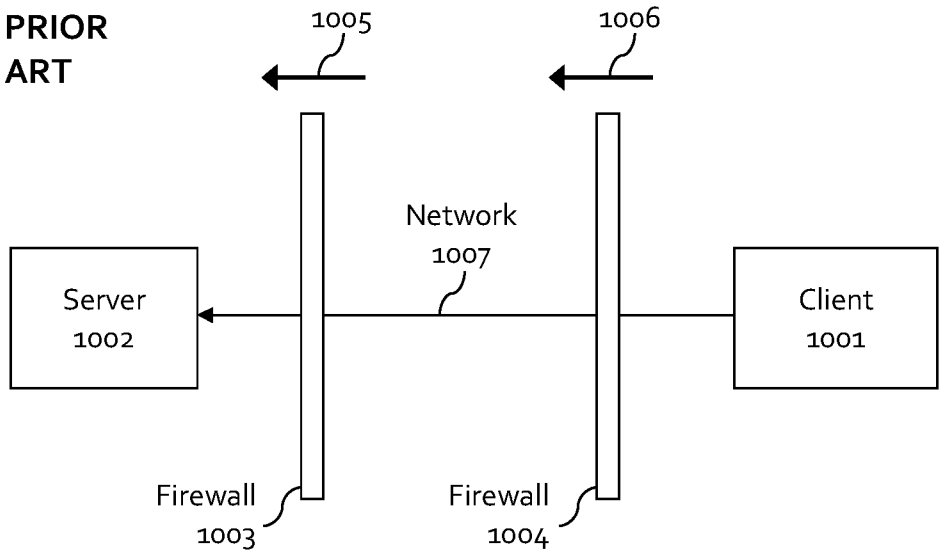

PRIOR ART
1005
1006
Network
1007
Server
1002
Client
1001
Firewall
1003
Firewall
1004

FIG. 10

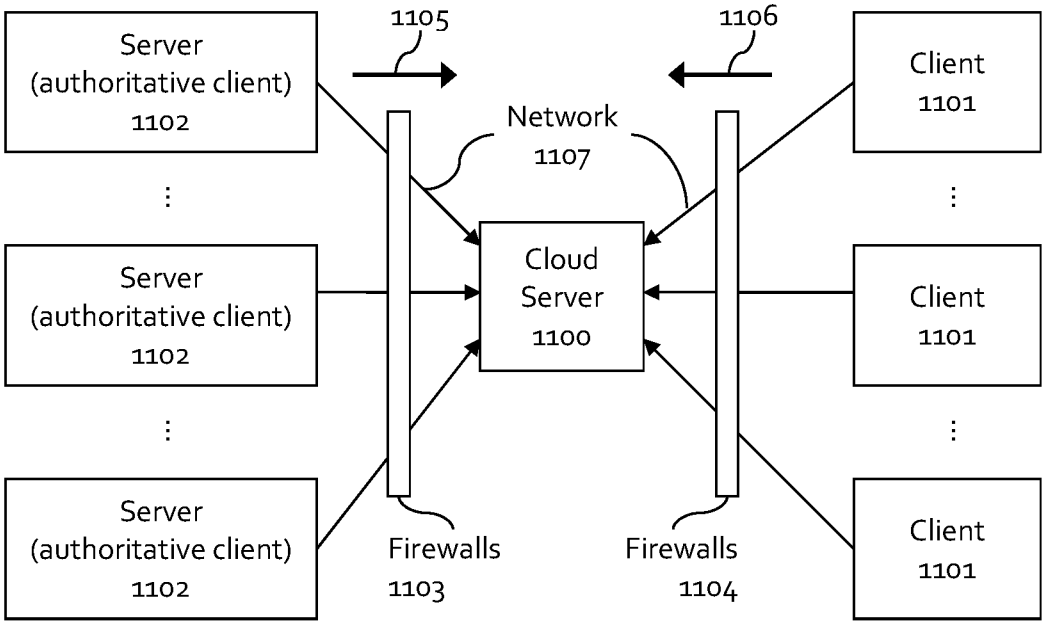

Server
(authoritative client)
1102
Server
(authoritative client)
1102
Server
(authoritative client)
1102
1105
1106
Network
1107
Cloud
Server
1100
Client
1101
Client
1101
Client
1101
Firewalls
1103
Firewalls
1104

FIG. 11

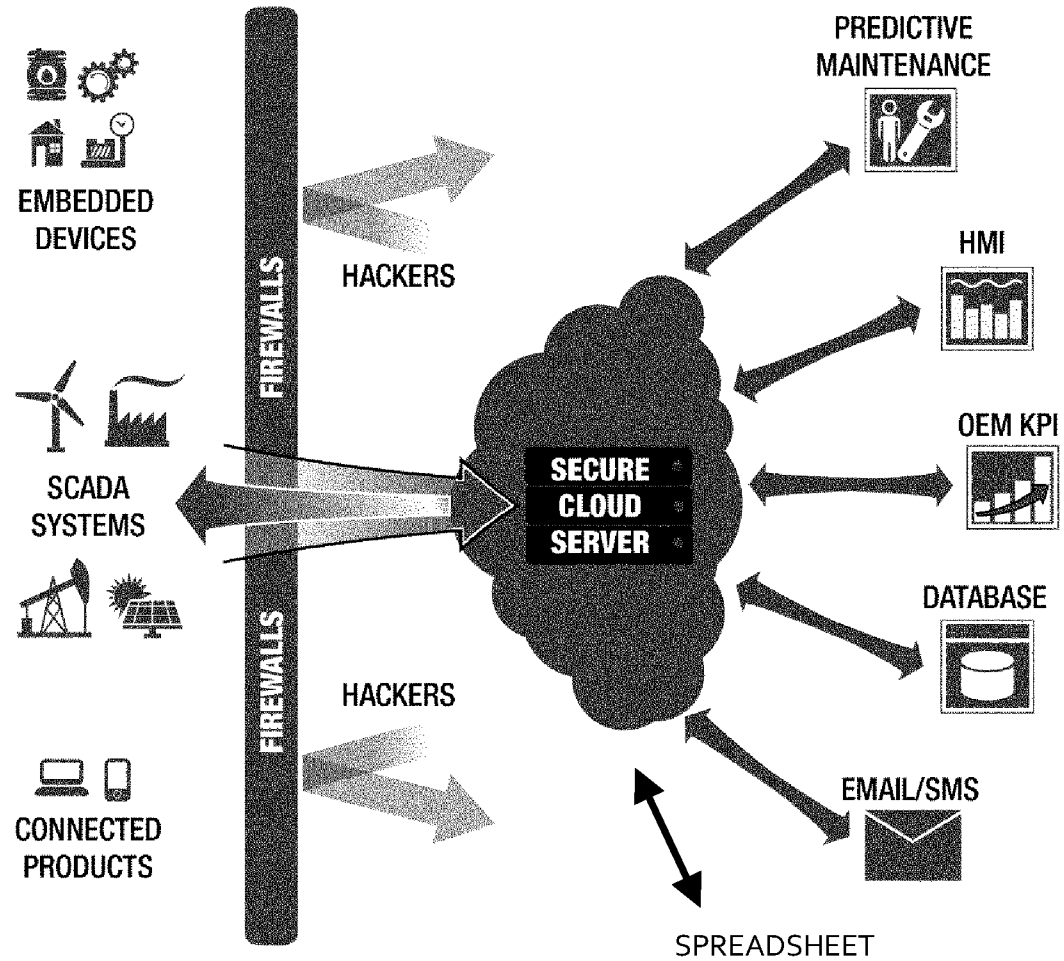
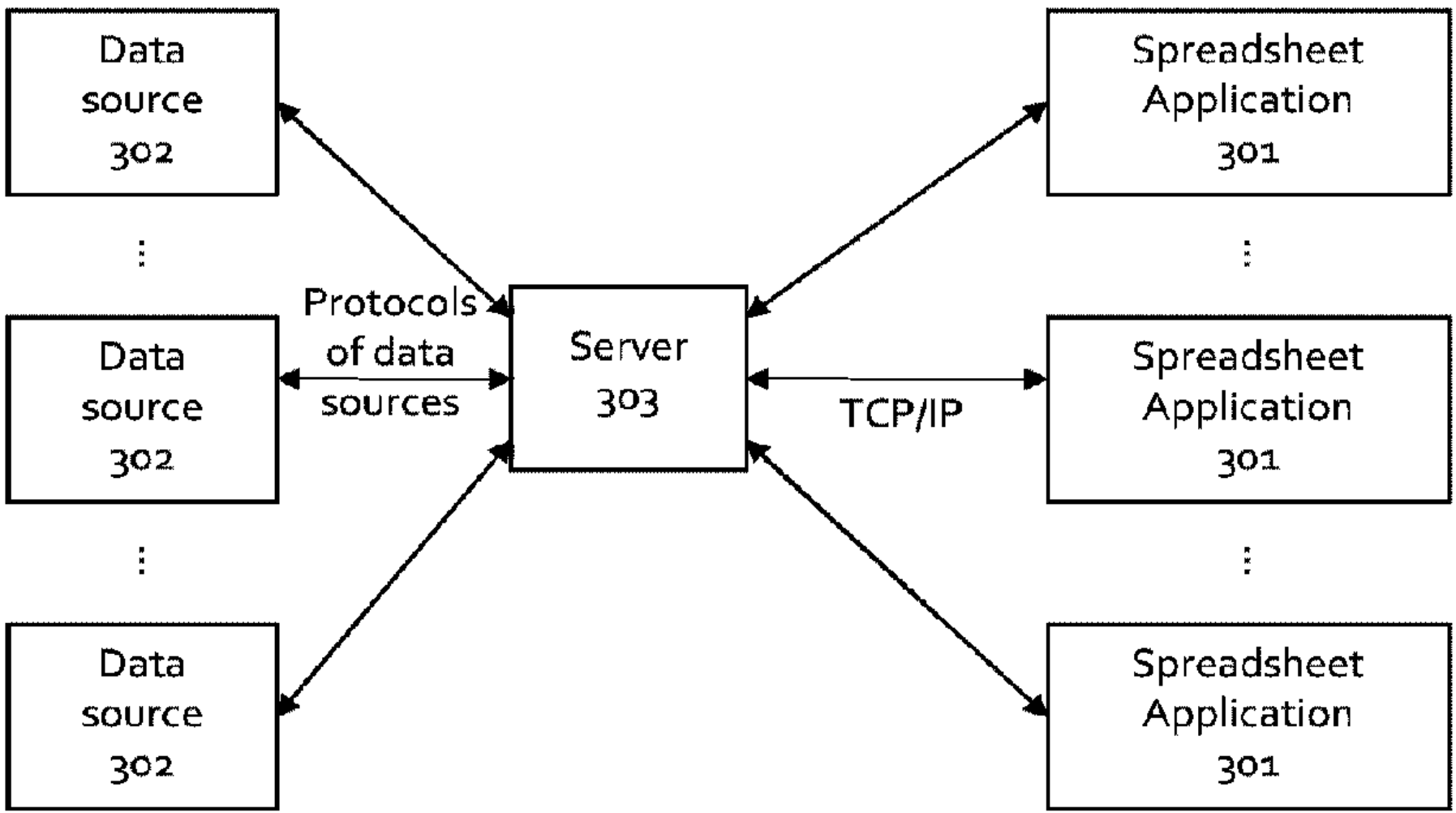

PREDICTIVE MAINTENANCE
HMI
OEM KPI
DATABASE
EMAIL/SMS
EMBEDDED DEVICES
SCADA SYSTEMS
CONNECTED PRODUCTS
HACKERS
HACKERS
FIREWALLS
FIREWALLS
SECURE CLOUD SERVER
SPREADSHEET APPLICATION (SSAPP)
SPREADSHEET APPLICATION (SSAPP)

FIG. 12


Data source 302
Data source 302
Data source 302
Protocols of data sources
Server 303
TCP/IP
Spreadsheet Application 301
Spreadsheet Application 301
Spreadsheet Application 301