International Journal of Networked and Distributed Computing, Vol. 1, No. 2 (April 2013), 72-78

A Middleware-Based Implementation for Data Integration of Remote Devices

Xianyong Liu*
School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University City
E-mail: ayongwust_sjtu@sjtu.edu.cn†

Yanping Liu
School of Ocean Science, Zhejiang Ocean University
Email: liuyp80@gmail.com

Lizhuang Ma
School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University
Email: ma-lz@sjtu.edu.cn

Qing Gao
Inner Mongolia Autonomous Region Disabled Rehabilitation Service Center

Received 4 April 2012
Accepted 13 November 2012

Different from existing techniques[3-5] that often focus on high level data integration, this paper contributes a novel solution at device level, implementing a fully scalable and configurable middleware, aiming at releasing the power of parallel processing with distributed data. It is designed as a soft-gateway to link upper-level PCs with lower-level data transfer units, and implemented by extracting the key part of data communication out of its original holder such that the UI only cares about how to display data.

Keywords: middleware, distributed systems, parallel processing, remote sensing, modules and interfaces

1. Introduction

According to the "Evaluation Criteria for Modern Urban Waterworks in Zhejiang", water supply companies are required to establish a pre-detection mechanism for auto-electrical equipment and machinery, using online instruments for status monitoring and fault determination.

Lower-level controllers interact with upper-level PCs by executing commands and reporting device status. The interaction, in essence, is conducted by exchanging data packets that are transmitted through GPRS-based network. Upper-PC functions as either a callee or a caller. As a callee, it monitors a certain computer port, receives data packets, unpacks them and displays the values on User Interface (UI). As a caller, it packs instructions and data, writes to the output buffer, then to the device via a serial port.

It is advisable for an enterprise to cut manpower cost by utilizing the network and auto-control. Unfortunately, an undesirable fact is that companies are greatly constrained in buying and upgrading devices. Because each vendor has its own Information System (IS) to support device management. What's worse, data, in different IS, is organized with discretionary protocols which are not public.

Although a large screen could be setup to show all providers' software with a window for each, this would still be inconvenient for operators to check and manage all targets because the operational style of each window

* 504 Room, SEIEE building 3#, 800 Dongchuan Rd,Min Hang,Shanghai,200240,China.
† ayongwust_sjtu@sjtu.edu.cn.

would be quite different. The consequence is that the information construction costs paid by the company increase:

(i) Duplicate investment on a workstation with software
The company has to buy the accompanying software though needs the lower device only;
(ii) Additional resource payment
Additional network, hardware, system and computer room space are allocated for a new controlling PC as a workstation;
(iii) More maintenance
System administrators have more servers and databases to maintain; the staffs in service center frequently switch between different software to monitor terminal devices;
(iv) Harder for data integration
It gets harder to integrate and share remote data internally/externally, because they are organized with different format and stored in dispersed storage.

With the number of device vendors, the company either finds a way to integrate heterogeneous remote data, or has to give up trading with other suppliers even though their equipment had a good price/performance ratio. The companies' evolving application landscape was becoming more and more complex as long as no standardization took place. [1-2]

In this paper, we focus on remote terminal data integration, which is the fundamental for inter-organizational integration that helps to exchange data, to unify software components, and to streamline business processes.

Different from those existing solutions and techniques, which focus on high-level data integration, our key contribution is a data integration solution for device-level. From the view of practical applications within companies, we believe lower level data integration plays fundamental part. Proposing a feasible and robust solution, aiming at solving this problem, is of vital interest. In this paper, a fully scalable middleware implementation is described by using a range of advanced technologies, such as component oriented programming, multiple thread, message queuing, XML, topic-based publish/subscribe design pattern.

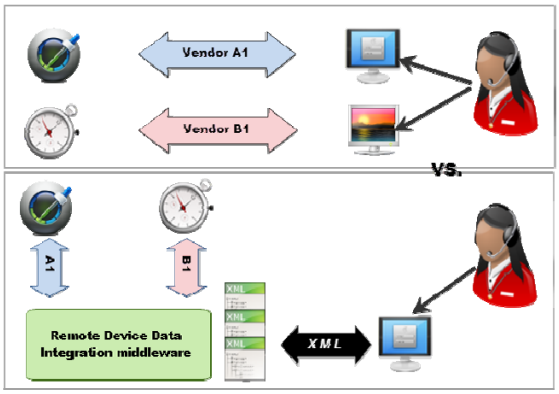

Fig.1. Existing problematic architecture vs. Improved architecture.

2. Solution

The root of the trouble is that each equipment provider uses its own protocol for packing data. In the water industry, as in many other fields, there is no unified standard available for the makers.

The communication between an upper-level PC and lower-level Data Transfer Units (DTU) is the key part in IS considered in this project. It is generally designed as a component that is tightly coupled with the upper-level software (We use workbench to refer to upper-level software as well).

Our work in this project is to separate the data communication functionality from its container, construct a middleware proxy, and have it run in the background as a common service. Figure 1 shows a comparison between the existing problematic architecture and the improved architecture after involving remote device data integration middleware.

The middleware takes charge of communication and bridges lower-level DTUs and upper-level software. It converts multi-source raw packets into parsed values in a general view of XML, which can be shared via Message Bus and consumed by all applications, including the device management workbench and other information systems.

3. Implementation

The toughest issues to resolve in design and implementation are that remote data is heterogeneous and there is a large amount of it. Unlike the high-level software of a particular vendor, the middleware needs to talk to all devices from any vendor.

A component-based technique is useful; each individual component is a software package (or module) that encapsulates a set of related functions. Following this principle, the middleware is composed of four child modules. They are packet transceiver, message reader/writer, packet parser/re-packer, and data forwarder. Furthermore, each module is comprised of independent components. Figure 2 illustrates the data processing flow in the data integration middleware. Such implementation guarantees the middleware has full flexibility and robustness.

It is fairly evident that the original higher-level software, used as a workbench, is much thinned out in functionalities; it only cares about how to display data in a nice way.

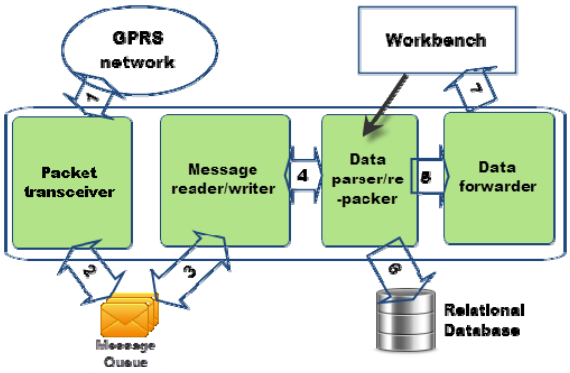

Fig.2. The data processing flow in the data integration middleware.

3.1. *Data layer design*

Yin and Zhang[6], analyzing information construction in the water industry, gave a result that the difficulties of data integration mainly reside in the difference of data structure, specification and technology architecture.

Before development, therefore, making a reasonable logic design in the data layer is quite important. It must make sure all parsed values of diverse data sources finally stream into a centralized storage. Such design is device independent, and offers a general data view for later global data integration. Having a thorough analysis of the logic design of existing legacy databases is also necessary.

Considering that the amount of data grows fast, the normalization rule is indispensable. For instance, the relation schema of tables in the database should be at least 3NF.

Given A_i is the set of parameters of device type T_j, A is defined as the union set of tuples in *IndicatorParam* table.

$$A \leftarrow A_1 \cup A_2 \cup \cdots \cup A_n. \qquad (1)$$

In database logic design, both "*horizontal pattern*" and "*vertical pattern*" are adoptable approaches. We adopt the latter because not all RTUs have the same number of properties, especially when equipment types are different. For example, serial *SM510-PF* has *29* parameters, while *NEW_JK* has *24*. Devices located in different places have different business uses. Data from a factory is comprised of residual chlorine, turbidity, PH, water level etc. A pressure checking point in a pipeline reports only the current pressure value. In "*horizontal pattern*", the number of columns in table RTUIndicator equals $|A_1 \cup A_2 \cup \cdots \cup A_n|$. This pattern doesn't accord with the rule of normalization.

3.2. *Packet transceiver*

The packet transceiver component plays a role as a Data Service Center (DSC). A DSC does not interact with lower-level controllers, such as PLCs, directly. Instead they are linked by a data transfer unit. DTU is designed as a GPRS-support solution. It transfers data to the controlling PC as data gathering device, and connects to lower-level devices via serial port RS-232/442/485 or TTL.

Normally, the packet transceiver is designed to comprise several sub-modules. Each module is used to communicate with a particular type of DTU. Two cases need to be taken into account during development:

(a) Is the DTU a third-party product?
Most equipment providers use a DTU from a third party, as the access point to the GPRS mobile network.

(b) Is DTU a proprietary product?
A few makers provide their own DTU with devices. In this case, free library available for interacting with lower-level devices as in the above case cannot be expected. As an alternative, developers could use auxiliary tools, like WinPcap, to grab raw TCP/UDP packets as a data source.

Figure 3 shows the configuration of the packet transceiver in our project. "Hongdian" and "PCapNet" are two sub modules in this component. The former

calls a third party API to fetch packets on port 5002, while the latter grabs raw packets on port 5003.

```
<DscConfig>
  <DSCs enabled="true">
    <DSC id="HongDian" enabled="true"
      type="Com.Maya.Fcrm.Services.Package.GPRSHon
          gDian"
      assembly="PackageManager.dll">
       <GPRS intervalunit="ms" packageToFile="true"
        packageFileRollsize="1024">
      <ServerParam ip="*.*.*.*" port="5002"
          workmode="1" protocol="0" />
      <DataParam displaywithhex="true" />
      <NonBlockParam interval="100" />
     </GPRS>
    </DSC>
    <DSC id="PCapNet" enabled="true"
      type="Com.Maya.Fcrm.Services.Package.PCapNet"
      assembly="PackageManager.dll">
     <GPRS intervalunit="s" packageToFile="true"
          packageFileRollsize="1024">
      <ServerParam ip="*.*.*.*" port="5003"
          workmode="1" protocol="0"   />
      <DataParam displaywithhex="true" />
      <NonBlockParam interval="1" />
     </GPRS>
    </DSC>
  </DSCs>
</DscConfig>
```

Fig.3. The configuration of packet transceiver

IP address and the port number, set in Fig. 3., are not the same as those set on DTU. In order to channel data packets into the physical gateway through the GPRS network, an IP address and port are configured for the DTU, and then a security rule could be defined by a firewall like Microsoft ISA server, which forwards raw packets to an intranet computer where the packet transceiver component resides.

The configurability is a typical characteristic of the packet transceivers, so that a new child component can be added or an old one removed on demand. A property switch named "*enabled*" (enabled=true/false) is provided for <*DSC*> node such that it is easy to turn on or shut down a sub module in this component.

3.3. *Message reader/writer*

Previously, packet processing in the vendor's software has been often synchronized. Namely a new packet will get no response until the packet on-hand is dealt with. It works well when the number of devices is small and the data load not heavy, however, in our scenario, an asynchronous mechanism is more appropriate. The middleware needs to interact with DTUs provided by all vendors. In addition, we need to leave place for future device expansion while designing it.

Microsoft Message Queuing (MQ) technique is introduced to resolve the data load problem mentioned above. It enables applications running at different times to communicate across heterogeneous networks and interact with systems that may be temporarily offline. From the perspective of development, there is no much difference from using other messaging systems, like IBM MQSeries. Also, many custom-built MQ models and implementations are available for reference. [7-9]

```
<!--MQ Processors-->
 <WorkQueueConfig>
  <service enabled="true" workerthreadcount="10"
                receivetimeout="90" >
   <logging enabled="true" debug="true" info="true"
      warning="true" error="true" logdir="d:\..\logs" />
  </service>
  <queues>
   <queue priority="1" path=".\PRIVATE$\zswaterone"
         messagetypes="HongDian,PCapNet" />
   <queue priority="1"
path=".\PRIVATE$\zswateronepackagetodb"
         messagetypes="PackageToDB" />
  </queues>
  <messagetypes>
   <messagetype id="HongDian"
                useconstructorparm="true"
       type="Com.Maya.Fcrm.BusinessRules.HongDianPr
                ocessor"
       assembly="BusinessRules.dll" />
   <messagetype id="PCapNet"
                useconstructorparm="true"
       type="Com.Maya.Fcrm.BusinessRules.PCapNetPro
                cessor"
       assembly="BusinessRules.dll" />
   <messagetype id="PackageToDB"
                useconstructorparm="true"
       type="Com.Maya.Fcrm.BusinessRules.PackageToD
                BProcessor"
       assembly="BusinessRules.dll" />
  </messagetypes>
 </WorkQueueConfig>
```

Fig.4. The configuration of message read/writer

Figure 4 shows the configuration of the message reader/writer component.

The transceiver defined in subsection 3.2 is the message sending application. When a packet arrives, the transceiver first checks the identity of the DTU and recognizes the packet's device type, then creates a message entity and tags it with a processor label, and finally pushes the message into a message queue. After that, it turns to the next-new-comer immediately without caring about what will subsequently happen to the

previous one. The message reader/writer component is the receiving application. It continuously fetches entities from the message queue, and triggers the corresponding processor by recognizing message labels.

3.4. *Packet parser/re-packer*

The data parser/re-packer component is the processor called by the message receiving application discussed in the previous section. It can also be regarded as the counterpart of the sub communication module in subsection 3.2.

The message processor first determines the packet's device type which is in the message entity, and then queries the database to get the list of parameters associated with that device type. Usually, the parameter list is queried once and then cached into the memory. Each item in the list contains a set of definitions, such as parameter index, bytes, type, conversion ratio. They are used to parse values from binary data stream. Finally, it calls the re-packer to encapsulate the parsed values.

The re-packer is a key component for data integration in the middleware. The goal of remote data integration in this paper is to finally output data to applications in a unified form by eliminating the heterogeneity of data that are from a multitude of sources. They can be shared by every IS no matter what technology it uses and what business it concentrates on.

XML is undoubtedly the most recommended form to re-encapsulate parsed values. As Alon[10] points out, one cannot ignore the role of XML in the development of data integration over the past decade. In a nutshell, XML fueled the desire for data integration, because it offered a common syntactic format for sharing data sources. The success of other technologies, like XPath, XQuery, XSLT, XML Data to UML Diagrams, and Web DB, shows the XML's popularity. [11-13]

Figure 5 demonstrates how the parser decodes values and the re-packer encapsulates data in XML format.

Fig.5. The parser decodes values and the re-packer encapsulates data in XML format.

3.5. *Data forwarder and workbench*

Applications, such as the high-level device monitoring workbench, can get values from underlying data inventory, while, from the viewpoint of management, this way is less ideal, and potentially risky, because it leads to certain delay in obtaining data on-site. In practice, the greatest value of an information system is its sensitivity to business change.

In order to meet the rule, a socket server is put in place as a data forwarder to overcome that problem, such that all applications are able to consume real-time values. A topic-based Publish/Subscribe design pattern is applyed in the implementation of the data forwarder component. This pattern is a very loosely coupled architecture, in which a data sender does not even know who the subscribers are. Sender applications tag each message with the name of a topic, instead of referencing specific receivers. The messaging system then sends the message to all applications that have asked to receive messages on that topic. Message senders need only concern themselves with creating the original message, and can leave the task of servicing the receivers to the messaging infrastructure. [14]

Workbench runs as an independent data subscriber like other enterprise applications, which only care about how to display the values in the User Interface layer. Since it becomes device-compatible directly, companies are no longer forced to invest in upper-level software. As times goes on, replacing or upgrading workbench won't cause any trouble.

Fig.6. An upper-level device management workbench.

Figure 6 is a screen-shot of a device management workbench in Chinese version.

Having the previously distributed data stream into a centered work station, not only does the company avoid duplicate investment on both software and hardware, but also achieve higher efficiency in management. In

addition, lower level data integration obviously paves the way for future data analysis, which becomes feasible only when the data is in a uniform view. Figure 7 shows daily archive data statistics from 11/17/2010 to 05/10/2011 in our project. The first one is daily total records. The second reflects fluctuation of data size. The peak value of rows appears on 03/22/2011 with 1,393,132 rows while 77,832KB is the peak data size value that emerges on 04/28/2011.

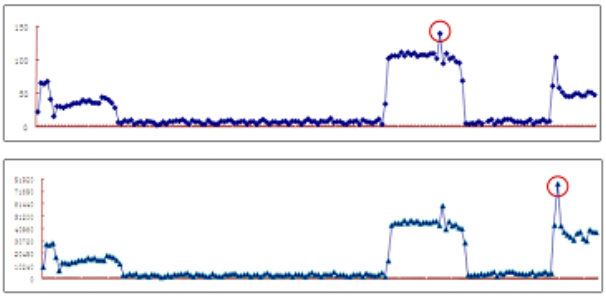

Fig.7. Daily archive data statistics from 11/17/2010 to 05/10/2011..

3.6. *Middleware deployment*

For many reasons, like data load, performance, security, and robustness, the rules, defined in a firewall, forward data packets to multiple servers by group. In this case, it is better to deploy the middleware on multiple servers, dealing with data of distributed sources in parallelization. By controlling “enabled” switch of component, the administrator is able to decide which modules to turn on and which to shut down. In our project, the middleware was deployed on two separate servers; one is for HongDian DSC and the other for PcapNet DSC.

It is also allowed to setup a message queue server and a socket server separate from the one where the middleware runs. Because the remote data integration middleware is designed to comply with the COP principle and each component is completely configurable.

4. Data Integration Extension

In order to respond to fast changing business, more and more information systems are involved in companies' daily operations. Many “How & What” questions have been emerging as this trend grows. How to construct information efficiently? How to avoid duplicate investment? What might existing data tell? What can we do before undesired things happen? Scholars and engineers seek aids and solutions from intra-organizational data integration.

Enterprise Application Integration (EAI) and Service Oriented Architecture (SOA) are feasible approaches people are trying. Nevertheless, they have difference because EAI emphasizes "Seamlessness" and SOA stresses “Capability”. Thus, the architectures of their implementation are different. EAI is commonly realized by establishing a Bus where messages flow through seamlessly to carry real-time data. Web Service is the most widely used method to implement SOA; data can be shared offshore by exposing reachable web methods.

Recently, the role of Radio Frequency Identification (RFID) applications in water supply companies has been gradually expanding. RFID readers can upload data via a GPRS-based DTU automatically. Our middleware can be extended when needed to operate in the framework of Wireless Sensors and RFID for ubiquitous Smart Environments (WISSE) raised by Sanchez Lopez et al. [15]

5. Conclusion and Future work

Remote data integration middleware described in this project resolved a vexing problem that Zhoushan Water Supply Company faced. When every vendor tightly couples upper-level software with lower-level equipment as an integrated solution, this greatly narrows the space for companies, choosing or upgrading devices on demand. Meanwhile, it also becomes an obstacle for intra-organizational data integration.

This paper describes the procedure for implementing a middleware in detail. The middleware is designed as a soft-gateway to link upper-level PCs with lower-level data transfer units, and implemented by extracting the key part of data communication out of its original holder so that the UI only cares about how to display data in a friendly way.

In order to endow middleware with enough robustness and flexibility, a component-oriented technique is used for program development. Here are brief descriptions of four child modules contained in the middleware. 1). The data transceiver plays the role of entrance for raw packets in and out, recognizes the device type of packets, and pushes messages into a queue with the packet as the entity body. 2). The

message reader/writer fetches messages from the queue continuously and triggers the corresponding message processor. 3). The data parser/re-packer parses a packet contained as the body of message entities according to the device type that determines which protocol is used for data organization, and re-packs parsed values with XML format. 4). The data forwarder communicates with a socket server that publishes the data as topics. All applications including the instrument monitoring workbench itself, reacting as subscribers, will get a copy of data that can be consumed without any difficulty since the data is re-defined and encapsulated in plain XML format. The data is pushed into another separate message queue and eventually streams out of a centralized database.

Many advanced techniques, like Component Oriented Programming (COP), Message Queuing, XML, Topic-based Publish/Subscribe design pattern, EAI and SOA, are used to get robust middleware with flexibility for future application extensions.

Geographical Information Systems (GIS) are commonly used in water supply companies and play a key role. They provide an e-perspective opened for water supply maintenance, consumer and water quality management, planning pipeline layouts, etc. From the viewpoint of management, it is necessary that devices including all distributed meters and gauges can be displayed in GIS with their real time running states. In the next step, in this project, we will look into cooperating with our GIS vendor to create a plug-in, which works as a topic-based subscriber (see section E), aiming to integrate remote device data with GIS. For other information systems like Equipment Management System (EMS) that have no special requirement for data in real-time, some web methods are to be exposed for retrieving and exchanging device information.

Acknowledgements

The authors would like to thank LvBo Cai, the promoter of this project. We are also grateful to the anonymous reviewers for their helpful comments. This work has been partially supported by the Science and Technology department of Zhejiang province, public welfare technology research project No.2010C31037.

References

1. A. Schwinn and J. Schelp, Data Integration Patterns, *Proceedings of Business Information Systems BIS*, Colorado, Springs, (2003).
2. S. Dirk, F. Daniel and N. Ina, A Framework for Assessing Inter-organizational Integration of Business Information Systems, *International Journal of Interoperability in Business Information Systems*, Issue 2(2), (2006), pp. 9-20.
3. Pervasive, Integration for Industry, (web resource), *http://integration.pervasive.com/Integration-Scenarios/Industry-Integration.aspx.*
4. Progress, Real Time Data Integration, (web resource), *http://www.progress.com/en/real-time-data-integration.html.*
5. A. P. Kalogeras, P. K. Athanasios, V. G. John, E. A. Christos, J. G. Manos and A. K. Stavros, Vertical integration of enterprise industrial systems utilizing web services, *IEEE Trans. on Industrial Informatics*, vol.2, no.2, (May 2006), pp. 120- 128.
6. M. Yin and X.G. Zhang, Water Service Information System Analysis of Data Integration, *China Management Informationization*, vol.11, no.18, (2008), pp. 77-80.
7. G. Min, M. Ould-Khaoua, D.D. Kouvatsos and I.U. Awan, A Queuing Model of Dimension-Ordered Routing under Self-Similar Traffic Loads, *18th International Parallel and Distributed Processing Symposium (IPDPS'04) - Workshop 14*, (2004), vol. 15, pp.251a.
8. A. E. Kostin, I. Aybay and G. Oz, A Randomized Contention-Based Load-Balancing Protocol for a Distributed Multiserver Queuing System, *IEEE Trans. on Parallel and Distributed Systems*, vol. 11, no. 12, (2000), pp. 1252-1273.
9. S. Vinoski, Advanced Message Queuing Protocol, *IEEE Internet Computing*, vol. 10, no. 6, (2006), pp. 87-89.
10. H. Alon, R. Anand and O. Joann, Data integration: the teenage years, *Proceedings of the 32nd international conference on Very large data bases,* (September, 2006), Seoul, Korea., pp. 12-15.
11. V. S. Lakes and F. Sadri, XML interope-rability, *Proceedings of the Sixth WebDB Workshop on the Web and Databases (WebDB),* (June, 2003), pp. 19-24.
12. V. Braganholo, S. Davidson and C. Heuser, On the updatability of XML views over relational databases, *Proceedings of the Sixth WebDB Workshop on the Web and Databases (WebDB),* (June, 2003), pp. 31-36.
13. R. J. Mikael, H. M. Thomas and T. B. Pedersen, Converting XML DTDs to UML diagrams for conceptual data integration, *Data and Knowledge Engineering*, vol.44, no.3, (2003), pp. 323-346.
14. Razan Paul, Topic-based Publish/Subscribe design pattern implementation in C# - Part I (Using socket programming), *www.codeproject.com/KB/IP/SocketBasedPubSub.aspx.*
15. L. T. Sanchez, D. Kim and T. Park, A Service Framework for Mobile Ubiquitous Sensor Networks and RFID, *1st International Symposium on Wireless Pervasive Computing*, (2006), pp. 16-18.