International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-6 Issue-5, June 2017

Efficient XML Interchange as Encoding Scheme in DDS

Sheela S, Ravi V.

Abstract: ***Data Distribution Services has a world wide application in distributed embedded and real time applications. These systems communicate data between computing nodes over a network. DDS when used in time-critical applications like military systems, there is always a need for data being communicated to be delivered in real time. In this article we propose a novel scheme where efficient XML interchange can be used for compression of the data before being communicated between the publisher and the subscriber. This scheme helps in increasing the efficiency of the data transfer by reducing the file size along with encryption of plain text so that unintended person can be avoided reading the data.***

Keywords: middleware, QoS parameters, participant, pre-compression, publisher, subscriber, data-centric

I. INTRODUCTION

There's a universe of prospects for distributed embedded and real-time applications. The rundown of uses continues developing: military frameworks, broadcast communications, industrial facility robotization, movement control, budgetary exchanging, therapeutic imaging, building mechanization, customer hardware, and the sky is the limit from there. Each of these ventures battles with how to make a solitary framework from many dispersed parts. In real time it is most important for the application is finding the right data, knowing where to send it and delivering it to the right place at the right time. On the other hand in embedded systems, it is important to quickly disseminate the data to multiple nodes. Topical floats in information driven frameworks are driving the development of mission-basic data administration proficiencies that guarantee that the correct data is conveyed to the ideal place at the perfect time to fulfill Quality of Service (QoS) prerequisites in complex grouped handling conditions. These information management frameworks, for example, radar processors, flight data processors and combat management systems increasingly run in net-centric environments pigeonholed by thousands of platforms, sensors, decision nodes, and computers coupled together to exchange information, support sense-making, enable collaborative decision making, and upshot changes in the physical environment. To bolster these mission-basic information management capacities in net-driven frameworks, the Object Management Group (OMG) molded the DDS specification, which is a standard for QoS-empowered information driven publish and subscribe correspondence, That helps net-driven mission and business-basic information management frameworks to share data continuously by exploiting QoS-driven publish/subscribe patterns. The advances in the publish/subscribe middleware are well suited for applications requirements of delivering the right information at the right place and the right time in data centric operations. In exact standard based distribute subscribe middleware, for example, Object Management Group's OpenSplice Data Distribution Service is a key engaging innovation to shape and propel vast scale and reliable distributed ongoing and embedded systems in a long run. On the other hand the exponential growth of data has led to the marvels known as big data that is generally huge in size such as peta bytes. Big data processing and storing requires innovative computational archetypes and skills to course these data in real time. Cloud computing has emerged as a promising hypothesis in providing a suitable infrastructure for large scale and complex computing. A data distribution scheme with differential levels of QoS enforcement policies can likewise be of assistance in cloud based big data distribution. In the immediate imminent, distributed application contexts are expected to support mobile code, multimedia data streams, user and device mobility, and impulsive networking. Scalability, quality of service, and sturdiness with respect to fractional component letdowns would become the key issues. In such a scenario, policy based Data Distribution Services would act as an apt solution to it. Framework utilitarian and QoS necessities can put diverse impediments and command on how a particular DDS framework can be fabricated. The arrangement of the execution environs and QoS necessities of the frameworks can force prerequisites on the essential DDS platform. Additionally increased security threats in WAN environments make information assurance, such as sheltered communication, user authentication, access control, and information veracity, a key ability a DDS foundation ought to bolster. It is essential for DDS implementations to provide the required QoS and performance trade-offs scalably and adaptively based on these restraints and requirements.

II. OGANISATION OF THIS PAPER

The remaining section of the paper is organized as follows. Section III provides the background details on DDS and EXI. Section IV introduces the proposed system.

Manuscript published on 30 June 2017.
***** Correspondence Author (s)
Sheela S, M.Tech Scholar, Department of Computer Science and Engineering, Siddaganga Institute of Technology, Tumkur - 572103, (Karnataka), India. E-mail: Sheelaslingaiah@gmail.com
Ravi V, Assistant Professor, Department of Computer Science and Engineering, Siddaganga Institute of Technology, Tumkur - 572103, (Karnataka), India. E-mail: ravi@sit.ac.in

Section V reviews the related work and finally, Section VI draws some conclusions.

III. BACKGROUND

This section of the paper throws some light on the background of the OMG's data distribution services and the efficient XML interchange specification.

A. Data Distribution Services (DDS)

Originally Data Distribution Services was designed by RTI as real time embedded technology grounded on two major product lines: RTI Data Distribution Services from Real Time Solutions and Splice from Thales. Initially it was developed as a solution for Robotic System and later has been adapted over to several applications. Many standards have been evolved. Object Management Group (OMG) has lately recognized the need for Publish Subscribe architecture and has adopted its first open international standard addressing the publish/subscribe middleware for embedded systems, named as OpenSplice. Middleware technologies can be broadly classified into several classes: Client Server, Message Passing and Publish Subscribe. Publish Subscribe technology augments a data model to messaging where the nodes just subscribe to the data they need and publish the data they produce. Publish Subscribe data model basically uses the multicasting mechanisms to enable effective and dependable data distribution. The messages are realized to be logically passing between the communicating nodes straight from the source to the sink without any intermediate servers making the whole process efficient. The major challenge lies in determining both discovering what data needs to be sent and delivering to the right place at the right time. Publish Subscribe systems have ended up being great at disseminating expansive amounts of time critical information even when the conveyance systems are undependable. In summary DDS aims at providing a scalable, real time, dependable, high performance and interoperable information exchange.

DDS at the core is comprised of Data-Centric Publish and Subscribe (DCPS) model, and a DDS Interoperability Wire Protocol (DDSI). The anterior describes the DDS architecture, the entities that are involved in the communication process and a standard API. When the communication is across DDS implementations from dissimilar vendors the later ensures the interoperability. The entities involved in creation and usage of a DCPS based applications are as follows:

- **Domain** – Domain provides an essential communication environment where the DDS applications communicate data. All the participants within the same domain will have the same domain id. Domain helps in isolation of communiqué within the community that shares similar interests by constraining only the participants with the common domain id to communicate.
- **Domain Participant-** A DDS application's contribution in a domain is represented by the entity Domain Participant. It works as manager, container and factory for the DDS entities briefed below.
- **DataWriter and Publisher-** DataWriter are the primary access point and used by the DDS applications to multicast the data to the domain's global data space. The Publisher entity is just a container to cluster together distinct Data Writers.
- **DataReader and Subscriber-** Data readers are used by the DDS applications to receive the data by different approaches agreed upon during while entering the contract. Subscribers are the factories used to create and manage the DataReaders.

Topic- *Publishers and Subscribers treat the Topics as the basic connection point. The Topic of a given publisher on one node must match the Topic of a related subscriber on some other node. Communication cannot be established if the Topic doesn't match. A Topic is consists of a Topic Name and a Topic Type. While the Topic Name is a string that particularly perceives the Topic inside an area, the Topic Type is the demarcation of the information contained inside the Topic. Topics must be unmistakably characterized inside any one precise domain.*

Data Distribution Service for actual time Systems provides the developers with a wide range of attributes to enable configuration of middleware to be able to control the end- to -end Quality of Service (QoS). QoS policies specify the different features associated with data delivery, data availability, data timeliness, resources, configuration, and entity lifecycle. DDS model and the associated communication mechanism are virtually controlled by the QoS parameters. QoS parameters are realized as "contracts" between the publishers and subscriber. On one end the publisher offers the available level of service and on the other end the subscriber request the required level of service. The middleware is accountable for determining if the offer could be satisfied and hence establishing the connection. QoS policies are applicable to particular entity while it is created and activated. It demands the expertise of the developers to be able to choose and apply the apt QoS policies depending on the system functionalities and the required performance. These QoS attributes remain unchanged during the whole execution of the system, unless the user has chosen to change it during the run time. The developer has the choice of changing the attributes at the run time by either having the standard communication templates, predefined to cover the recurring problems in the publish/subscribe applications or the adaptive QoS policies that can adjusted at the run time based on the context. The Data Distribution Services is fundamentally meant for data-centric communications while it is not well suited for message passing, file transfer and transaction processing services. The motivating factors that indicate when application succeeds with DDS include the Complex Data Flows, Need for Speed, Fault Tolerance, Dynamic Configuration.

B. Efficient XML Interchange

Web developers make use of XML in many aspects, but often to simplify data storage and data sharing. In the real world, data is contained in the incompatible formats in computer systems and databases. While the plain text format is used to store the XML data, it provides a software- and hardware-independent way of storing data.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

The major challenge faced by the web developers was to exchange the data over the internet between incompatible systems. XML was developed as a solution to this, as the data exchange as XML was known to greatly reduce the complexity and data could be read by different incompatible applications.

Also Upgrading to new systems, always tend to be time consuming, as it involves converting large amounts of data which in turn may return result in loss of incompatible data. Data stored in the XML format is usually in text format, and hence simplifies the process of upgrading or expanding to new applications, new browsers or new operating system, without the loss of any data. It may be unreasonable for some to utilize XML for the reason that it comes in sheer size and verbosity clogs networks, overpowers littler gadgets, and may lead ease back information transmission to a creep. Handling XML can moreover ingest a lot of CPU time which contracts throughput and versatility, expanding the cost/execution proportion of the computing resources.

To assist the developers of the applications where the faster and efficient data exchange between the cross platforms over the internet was needed the Efficient XML Interchange (EXI) format was developed **Efficient Extensible Interchange Working Group at W3C**. EXI is a binary XML format which is a very condensed and highly recital representation of the XML data. It instantaneously progresses the performance and considerably diminishes bandwidth requirements without conceding effective use of other resources such as memory, code size, battery life, and processing power.

The customary utilities like gzip and bzip compressed just the data, but Efficient XML grosses the knowledge that is already present within XML data to be able to make the data smaller, faster and, comparatively, more proficient. It is also known to work well even for a wider range of XML messages. It essentially makes the XML documents hundreds of times smaller and quicker by expending the prior knowledge about the XML, resulting in the production of more rationalized, capable version of that document.

The Efficient XML was identified by the World Wide Web Consortium (W3C) best-of-kind XML optimization technology and proposed the adoption of Efficient XML Interchange (EXI) format as the global web standard for efficient, interoperable XML exchange across cross platform devices. Efficient XML is completely built on open standards, and hence allows the developers to use Efficient XML without limiting them to a closed, proprietary approach.

Benefits of using Efficient XML are as follows:

- It makes XML data smaller by hundreds of times
- It works for XML messages of all schemas and all sizes
- It is 100% compatible with XML
- It works well with fully unstructured, semi-structured, and structured XML data
- It adapts to changes in XML data over time, including new versions, updates, and deviations

Efficient XML provisions an extensive range of conversant, customary XML APIs that lets the application developer to integrate Efficient XML to their existing XML applications at ease. Java API for XML Binding (JAXB), Java API for XML Processing (JAXP), .NET XML Reader/XML Writer, Simple API for XML (SAX), Streaming API for XML (StAX) and W3C Document Object Model (DOM) APIs forms the set of API's that are provisioned with Efficient XML . Efficient XML is sustained on the platforms listed in the below table:

Table 1: Platforms that support efficient XML

Platform	Version
Java Micro Edition	Java Wireless Toolkit 1.0.4 or greater
.NET	. NET Framework version 2.0 and later
.NET Compact Framework	.NET Framework version 2.0 and later
C/C++	Windows, Linux, Mac, Mobile & Embedded
Android	2.2 or later
iPhone, iPad	iOS 4 or later
Integrity Real-time OS	x.x or later
Java Standard Edition or Enterprise Edition	1.4 or greater

Open EXI converts any XML file to EXI format by the use of Transmogrifier class. The Transmogrifier class uses the schema information that is available; if no scheme information is available the Transmogrifier constructs and records the structure of the file as it is converted. Each of the elements and attributes is handled as a separate event as the file is read by the SAX parser. It excludes all the non-essential information from the transformed file.

The initial processing instruction is replaced with a Start Document, or SD, marker, but the SD marker is not included in the final file. When decoded, OpenEXI assumes that a new file stream starts with an SD event, and ends with an ED event. The SE event replaces every new element encountered. When the element is encountered for the first time; the name string is stored with it. At the later happenstances, the name is inferred. AT and CH events store the content data. Each of these markers is only 2-6 bits in size. Removing the labels from the start and end elements in the file is just the starting point for reducing the file size, but it represents a significant savings. Another reason that content can be stored so economically is EXI's use of a String Table.

IV. PROPOSED SCHEME

DDS can be deployed as a single process or standalone system or a shared memory system. Standalone mode of deployment allows the DDS applications and administration to be contained together within one single operating system process. This standalone single process deployment option is most useful in environments where shared memory is unavailable or undesirable. As dynamic heap memory is utilized in the single process deployment environment,

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

there is no need to pre-configure a shared memory segment which in some use cases is also seen as an advantage of this deployment option. As already discussed in the previous section, transferring the XML data would take a considerable time. When the DDS is used as part the real time-critical system, there is need for transfer of data instantly in the real time. When we use XML format and when the quantity of data under transmission is large or your XML devices have low memory or bandwidth the time taken for transmission is considerably large.

In this paper we suggest a scheme where we use the Efficient XML Interchange format to represent the data being transmitted. We use the Transmogrifier class to convert the XML format into EXI format. The Transmogrifier class takes an XML stream as an input and converts is to EXI format, which is the binary representation of the XML format. By default the Transmogrifier class is designed to intake FileStream as input, when we want to pass the string as input to the class we convert it into ByteArrayInputStream format before being passed to the Transmogrifier class. Another benefit of using the exi formatted message for data transfer is that when the unintended person receives the message, he would be unable to interpret it as the message is in EXI encrypted form.

XML is loved by people because it is easy to read, flexible and portable. But the word "portability" is a relative term. The computer end up do lot of human baggage work. When EXI is used we can reduce the burden on the computer by sending only the information needed. File size is substantially reduced by substituting the human decipherable elements with the succinct encoding. Exi groups the information into channels that pack most possible values into the smallest number of bits. The deflate algorithm is then used to compress these efficient files into as small as possible. Exi does the pre-compression alignment to optimize the storage.

The converted ByteArrayInputStream is then passed onto the middleware to be picked up by the subscriber. On the subscriber side the SAX transformer and SAX parser is used to decode the EXI string back into the XML format. There is no risk of data loss when the EXI files are compressed and decompressed back, but when the encoded string is passed to the middleware and decompressed back there might be few disagreements with the original data. This is because of the spaces and other special characters in the original file. This problem can be overcome by converting the encoded data into base64 format before it is handed over to the middleware. The extra effort is payed by the increased efficiency in the data transfer.

V. CONCLUSION

The Data exchange between resource constrained devices is often realized using efficient hand-crafted binary encodings of the transmitted data. On the other hand, information exchange in distributed systems on the Internet is increasingly done using the widespread text-based XML format. XML files are simple text files; using them as a means to exchanging data is therefore relatively inefficient in terms of data size and not suitable for devices with limited resources. Use of the efficient XML significantly increases the efficiency of the data transmission which the core of the time-critical systems. Efficient XML can be easily integrated easily with any applications that communicate data in XML formats, as the API's are readily available in many platforms. XML information is convenient for people to work with, while EXI information is far more compact and transfers quickly.

REFERENCES

1. G. Pardo-Castellote, "OMG data distribution service: architectural overview," IEEE Military Communications Conference, 2003. MILCOM 2003.
2. Hadeel T. El Kassabi; Ikbal Taleb; Mohamed Adel Serhani; Rachida Dssouli, "Policy-based QoS enforcement for adaptive Big Data Distribution on the Cloud", IEEE Second International Conference on Big Data Computing Service and Applications (BigDataService), Year 2016
3. Nanbor Wang; Douglas C. Schmidt; Hans van't Hag; Angelo Corsaro, "Toward an adaptive data distribution service for dynamic large-scale network-centric operation and warfare (NCOW) systems," 2008 IEEE Military Communications Conference, Year 2008
4. Paolo Bellavista; Antonio Corradi; Luca Foschini; Alessandro Pernafini, "Data Distribution Service (DDS):A Performance Comparison of Open Splice and RTI Implementations," I IEEE Symposium on Computers and Communications (ISCC), Year 2013
5. Gerardo Pardo-Castellote, Ph.D., "Data distribution service advanced tutorial", Real-Time Innovations, Inc. http://www.rti.com"
6. Juan Ingles-Romero; Adrian Romero-Garces; Cristina Vicente-Chicote; Jesus Martinez, "A Model-Driven Approach to Enable Adaptive QoS in DDS-Based Middleware", IEEE Transactions on Emerging Topics in Computational Intelligence.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)