uniDDS: A unified Data Distribution Approach for the International Monitoring System

T4.1-P25
SnT2017

S. Laban*, A . Sudakov, T. Edwald and G.Kuzmenko

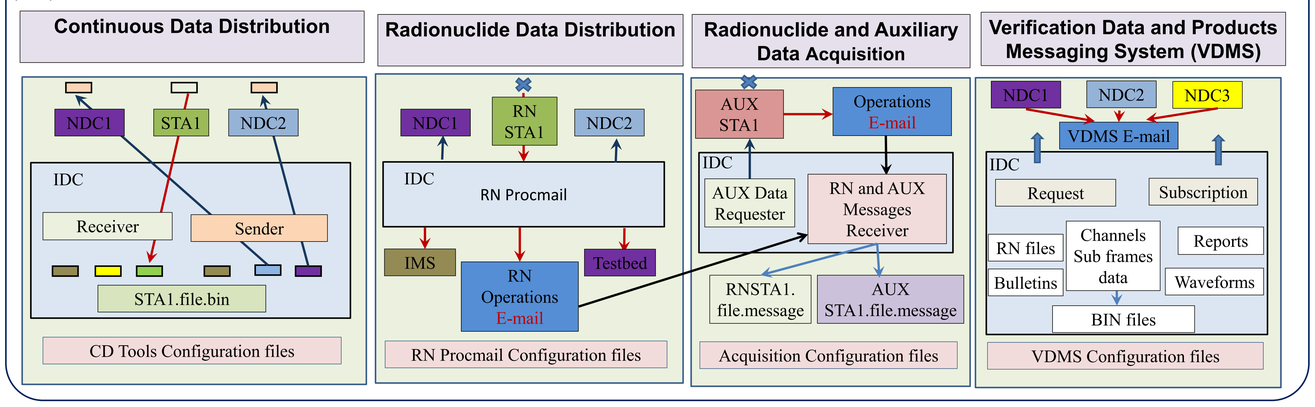
Preparatory Commission for the Comprehensive Nuclear-Test-Ban Treaty (CTBT) Organization, Vienna International Centre, P.O. Box 1200, A-1400 Vienna, Austria.

Abstract

The International Monitoring System (IMS) consists of 337 facilities, 321 monitoring stations and 16 laboratories, built and distributed worldwide. These facilities send different types of raw data in near real time to the International Data Center (IDC) using a dedicated Global Communications Infrastructure (GCI) that provides timely and reliable transmission of IMS data. IDC distributes IMS data, IDC bulletins and reports to member states. Currently, different types of processes and tools are used for information distribution based on type of data or reports generated. Maintaining and monitoring such processes has flexibility, scalability, efficiency and performance issues, as both data volume and number of recipient states increases. The aim of this poster is to present a unified, flexible, reliable, efficient and distributed framework for data distribution based on Data Distribution Service (DDS) and Complex Event Processing (CEP), to overcome these challenging issues. The details of the proposed methodology, implementation, experimental results, advantages, and limitations of this approach are presented. Finally, future directions and recommendations are discussed.

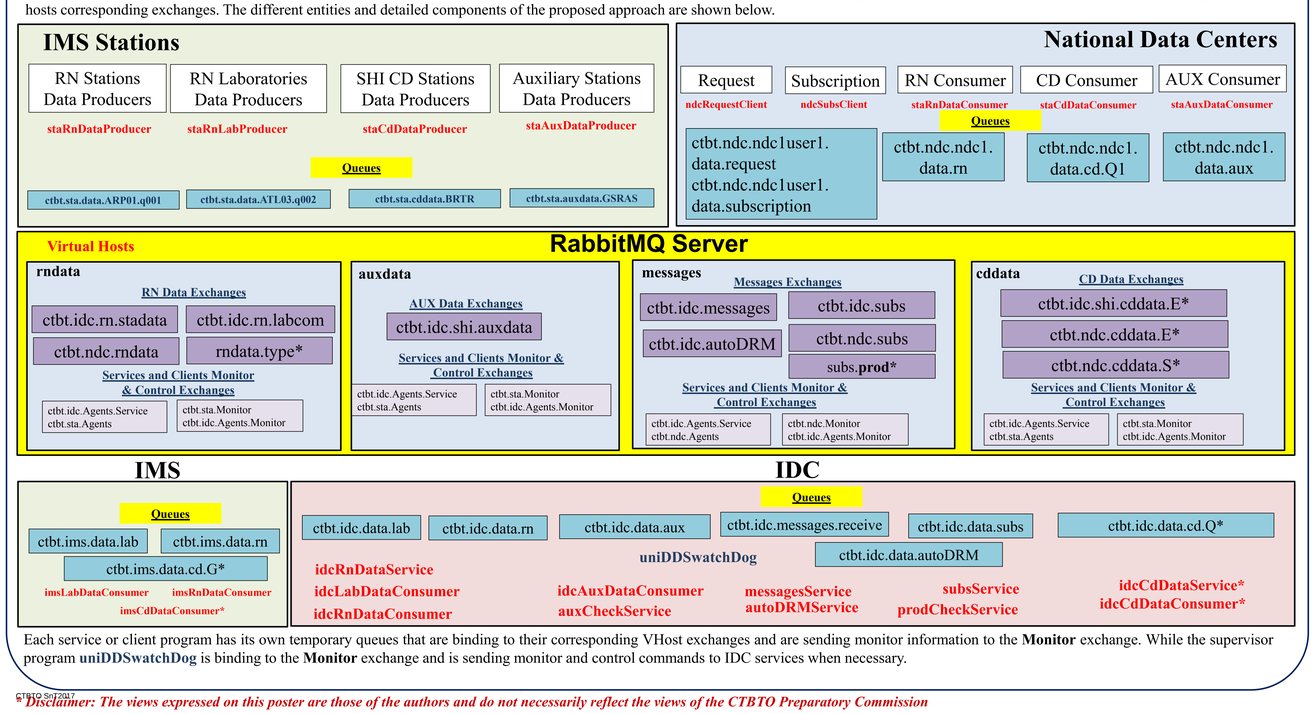
Background

Different types of data are used during the distribution process including Continuous Data (CD), Radionuclide data (RN), segmented or Auxiliary data (AUX), e-mail messages and command & control messages. As show in the following diagrams, different protocols and approaches are used , during data distribution, to handle the different data types among station (STA), IDC and any National Data Center (NDC).



Proposed Approach

The proposed approach uses rabbitMQ server , an open source message broker software, that implements the Advanced Message Queuing Protocol (AMQP) . Four virtual hosts (VHost) are proposed for handling the different types of data. The rabbitMQ clients are shown in red. The station data producers and IDC data services are sending data to the IDC data consumers or NDC clients through their virtual hosts corresponding exchanges. The different entities and detailed components of the proposed approach are shown below.



Each service or client program has its own temporary queues that are binding to their corresponding VHost exchanges and are sending monitor information to the **Monitor** exchange. While the supervisor program **uniDDSwatchDog** is binding to the **Monitor** exchange and is sending monitor and control commands to IDC services when necessary.

Experimental Implementation

The rabbitMQ server allows virtual hosts implementation and uses different types of exchanges and queues as a flexible data distribution technique. Four virtual hosts are proposed: **messages**, **rndata**, **auxdata** and **cddata**. The experimental implementation for each virtual host is summarized below and depicted in its corresponding right graph.

The **messages** virtual host has the exchanges: **ctbt.idc.messages**, **ctbt.idc.autoDRM, ctbt.idc.subs,** **ctbt.ndc.subs** and **subs.prod*** (exchange per product type, e.g. **subs.REB**). Four main IDC services are proposed. The **messagesService** is binding to the **ctbt.idc.messages** exchange and is responsible for handling all NDC incoming messages. If the message is of type 'request', the service forwards it to the **autoDRMService**, through **ctbt.idc.autoDRM** exchange, that handles the request and sends the reply directly to the requestor's queue. While the **prodCheckService** is monitoring IDC database for new products and if it finds new product it sends a message to the **subsService**, through the **ctbt.idc.subs** exchange, that generates the product and sends it to the queues of the NDCs through the **headers** exchanges **ctbt.ndc.subs** (filtered by product type) and **subs.prod*** (filtered by station).

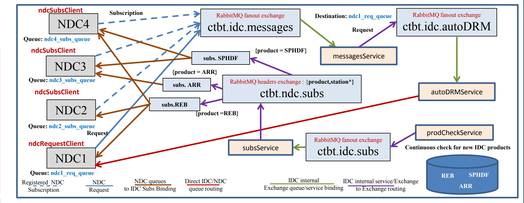
The **auxdata** virtual host needs one exchange , **ctbt.idc.shi.aux**, for receiving data from auxiliary stations. While every station has private queue **ctbt.sta.auxdata.STATION** (e.g. **ctbt.sta.auxdata.GSRAS**). The **auxCheckService** regularly checks IDC SHI processing for any requests that need to be forwarded to any station. In case a new request is needed, the program sends the required request to the queues of the identified stations. Once the station is ready, its program **staAuxDataProducer** sends the generated data to **ctbt.idc.shi.aux** exchange. The exchange forwards the request to the **idcAuxDataConsumer** that receives the data, stores and process es it accordingly.

The **rndata** virtual host has two main IDC exchanges: **ctbt.idc.rn.stadata** and **ctbt.idc.rn.labcom.** Also, it has NDC exchanges: **ctbt.ndc.rndata** and **rndata.type*** (exchange per RN data type, e.g. **rndata.SAMPLEPHD**). Once the RN data is ready at the station, the **staRnDataProducer** program sends it to the **ctbt.idc.rn.stadata** exchange. The exchange forwards the message to queues bound to it: **ctbt.idc.data.rn** and **ctbt.ims.data.rn**. The **idcRnDataConsumer** is listening to queue **ctbt.idc.data.rn** that receives RN data and stores it accordingly. Also, the **idcRnDataService** is listening to same queue **ctbt.idc.data.rn** that receives copy of RN data and prepares/updates message header accordingly. Then it sends the message to the queues of the NDCs through the **headers** exchanges **ctbt.ndc.rndata** (filtered by RN data type) and **rndata.type*** (filtered by station). In case of **labdata**, the **staLabProducer** sends it to **ctbt.idc.rn.labcom** exchange that forwards message to all queues bound to it including **ctbt.idc.data.lab** where **idcLabDataConsumer** is listening and handles the laboratory data accordingly.

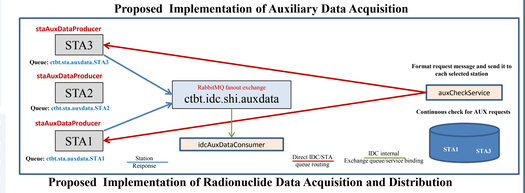
The **cddata** virtual host has several topic exchanges, one exchange per group of CD stations, for handling CD data (e.g. **ctbt.idc.shi.cddata.E[1-N]**). For NDCs, CD data groups of exchanges may be created (e.g. **ctbt.ndc.cddata.E***). Also, IDC has different queues configured to receive the CD data from all SHI stations (e.g. **ctbt.idc.data.cd.Q[1-N]**). Additionally, a queue or more per sending CD array will be created (e.g. **ctbt.sta.cddata.STA**). For NDC, several queues may be created (e.g. **ctbt.ndc.ndc1.data.cd.Q1**). Once the CD frame is ready at the station, the station sends it to its dedicated exchange that forwards it to its bound queues where instances of **idc/ndc/imsCdDataConsumer** handles the received frame and store it accordingly. The program **idcCdDataService** will read CD frames, filters it and finally sends CD data frames to special exchanges (e.g. **ctbt.ndc.cddata.S[1-M]**) and queues that will be created for handling special requests from NDCs regarding specific arrays. Such requests include partial forwarding of CD data for specific sites and/or channels.

The rabbitMQ server has its own flexible web interface to manage and configure users, virtual hosts, exchanges and queues. From the web interface it is possible to monitor the created queues and exchanges. In addition, a special program **uniDDSwatchDog** is proposed for monitoring ,per virtual host, the current running IDC services as well as NDC or station programs. The program is capable of starting or stopping only IDC services if necessary. Experimental implementation examples for running the **watchdog** program for **messages** and **auxdata** virtual hosts are shown below.

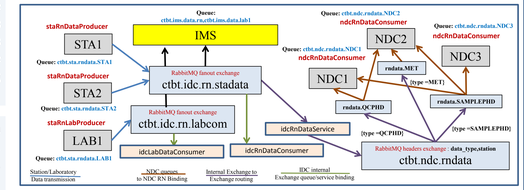
Proposed Implementation of Request and Subscription



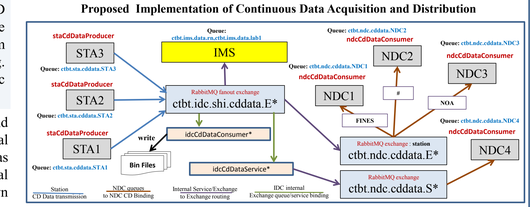
Proposed Implementation of Auxiliary Data Acquisition

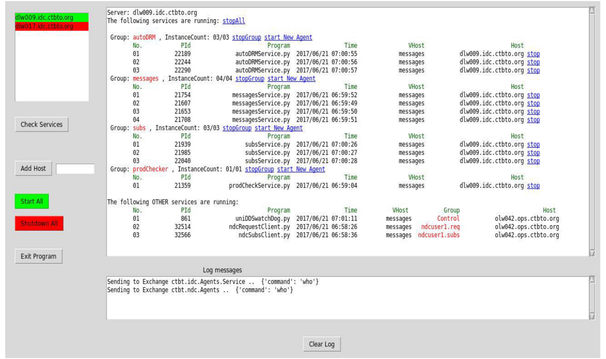


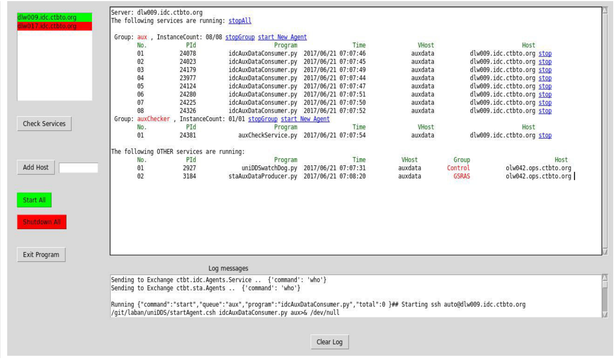
Proposed Implementation of Radionuclide Data Acquisition and Distribution



Proposed Implementation of Continuous Data Acquisition and Distribution







Summary

Experimental implementation of the proposed approach is feasible. Building data distribution applications using message queues can significantly simplify coding of applications as well as improving performance, scalability and reliability of such systems. By using rabbitMQ server, it is possible to build a flexible, unified and efficient data distribution framework. Also, it was possible to send big files by splitting them by the data producers and combine them later by the data consumers. Additionally, it was possible to transmit CD data frames, big auxiliary files as well as RN messages. The proposed approach is using the IMS2.0 format for messages/products distribution as well as using CD protocol for sending CD frames. The additional information, commands and responses used among the framework processes are sent within the header information of the rabbitMQ message. Due to volume, criticality and nature of real time data, more time and efforts are needed for intensively testing those various aspects. Furthermore, data authentication need to be implemented with the proposed approach.

Future Work

Future work will be dedicated to add data authentication to the proposed approach. Also, more time and efforts are needed to fully implement and test the CD data distribution process. For CD data forwarding using rabbitMQ, we may apply changes to the way of storing temporary CD frames to other searchable formats, file based database (e.g. SQLite), to allow fast and simple search of requested frames. Also, more stress testing of the proposed approach is needed. Additionally, a real time testing of the proposed approach by some NDCs need to be facilitated. Finally, web–based interface for monitoring and controlling IDC services may be developed .