Design Document for DTESP (General Integration Data Turbine and Esper)

Heejin Choi

1. System requirements

(1) Receive multi channel streaming data from multiple data turbine servers
(2) Process Esper queries over the data in real time
(3) Save results of the queries into data turbine
(4) Make configurations of connections and queries adaptable to other applications
(5) Ability to process historical (previous stored) data fast

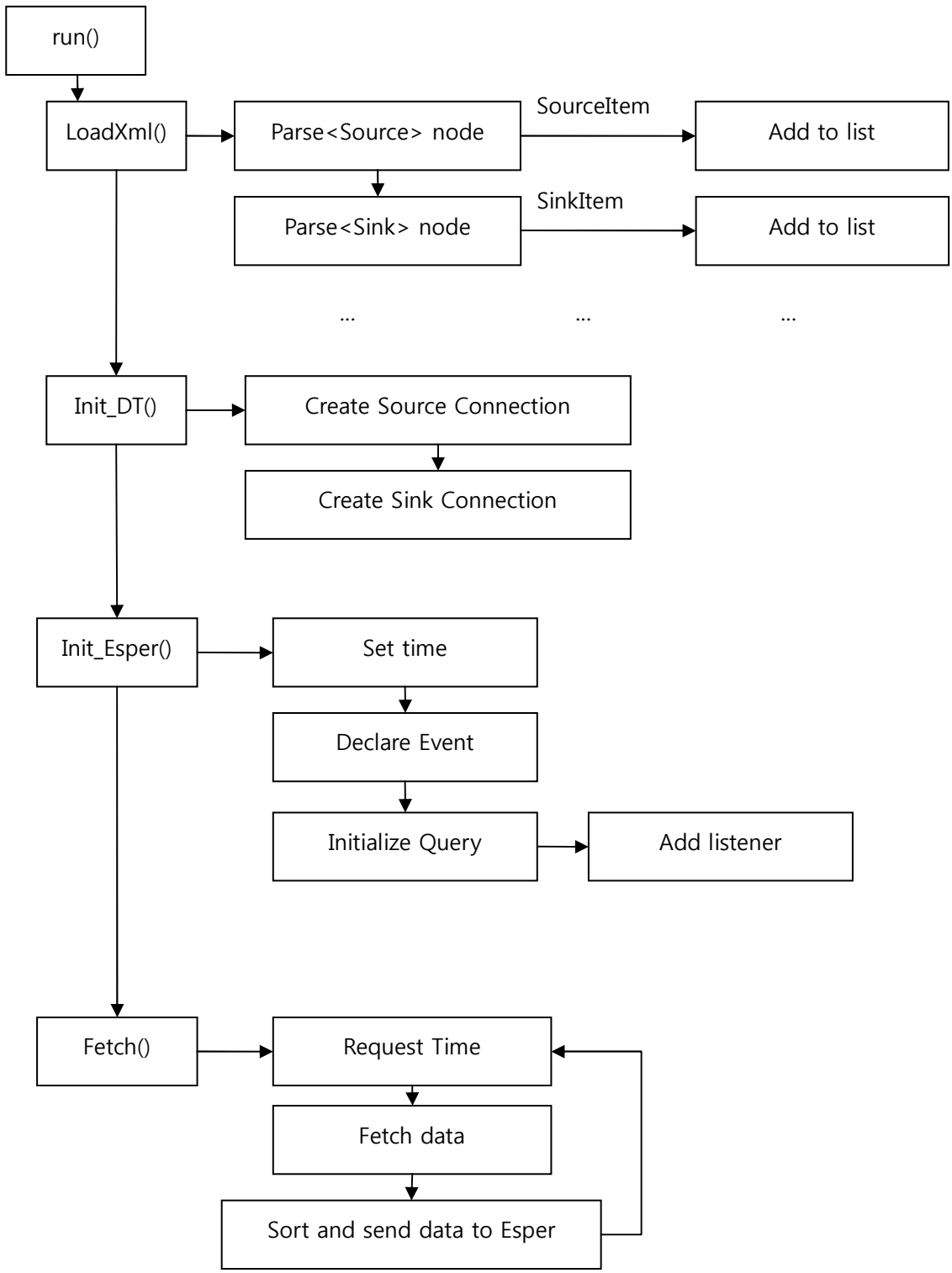
2. Design issues

(1) (2) (3) can be done simply using sapi library of data turbine, and Esper library. To integrate data turbine and Esper, data stream from data turbine is sent as an Esper event. Also Esper event listeners is attached to Esper queries whose result need to be saved.

For (4), all configurations, including connection information, and queries, are saved in an xml file. The xml file is parsed and stored at its corresponding class that represents the xml node. For example, data from xml node that represents 'source server', xml node <Source>, is stored at 'SourceItem' class, and after all the parsing is complete the data is used to create connection. As for queries, data from <Query> node is saved at class 'QueryItem'. Also time setting which required in (5) is saved at xml node.

For (5), system needs to request data window at certain time, and move the time of the window. This raises issue of setting Esper time. For Esper queries to have valid results, time of Esper must correspond to time of the data from data turbine. For instance, to have valid result for 'average of rain over 5 min', Esper need to receive data in timely matter. However, as all the data is already saved in the server, we don't have to actually wait for next data. Therefore, after receiving the new data from the data turbine, the time of the Esper is set to the time of new data, and then the data is sent to Esper. In addition, time granularity raises another problem. Some of the Esper queries assume that time advances with certain time granularity. For example, there can be an Esper query making output every 3 seconds. For this query, it is assumed that the time of Esper doesn't advance more than 3 sec at a time to have valid results. As a solution, when setting time, we check if the time difference from previous set time and if it is greater than time granularity, set also for intermediate the Esper time. For instance, if the time granularity is 3 seconds, and if we are advancing 7 seconds, we set time to at 3 seconds, 6 seconds, and 7 seconds. As a last issue, Esper need to receive data in time order, but we receive data from data turbine with channel basis. Therefore, we need to find out which data to send to Esper given data from all the channels. As a solution to this problem, we keep sorted linked list of channel index sorted by timestamp of the earliest data of the channel (the head of the data recieved fromt the channel). Therefore retrieving the first channel index of the sorted linked list ensures that the channel has the earliest data. We send the data to Esper and remove the data from the sorted linked list and remove from channel. If the channel has more data to send, insert the earliest timestamp of data left in channel into the sorted linked list, and repeat the process until all the data is gone.

3. Conceptual flow chart



4. Purpose of subsystems

I. Introduction

Task of DTESP consists of

I. Reading configuration file→LoadXml(), Item classes (SourceItem, SinkItem,… etc)
II. Data Turbine initialization and connecting→Init_DT()
III. Esper initialization→Init_Esper()
IV. Reading from Data Turbine and sending it Esper
→Fetch(), class ChannelIndexSortedByTime, class ReceivedDataFromChannel
V. Listens to result of Esper queries and send to Data Turbine→ EsperEventListener class

Codes for these task are divided into methods involving some inner classes, for instance, Reading configuration file is done at LoadXml() involving inner classes such as SourceItem, SinkItem, and etc.

II. Reading configuration file
At method LoadXml(), it reads xml file and parse data into corresponding data type, and add to list. The data will be needed for future initialization of Esper or Data Turbine. For instance, <Source> node will create SourceItem class and read the xml file and set the fields and added to list.

III. Data Turbine initialization and connecting
At method Init_DT(), it iterates through data classes created by parsing xml file and setups Data Turbine connection. For instance, it iterates through all SourceItem from the list which is created by II and creates Source class and connect to Data Turbine server.

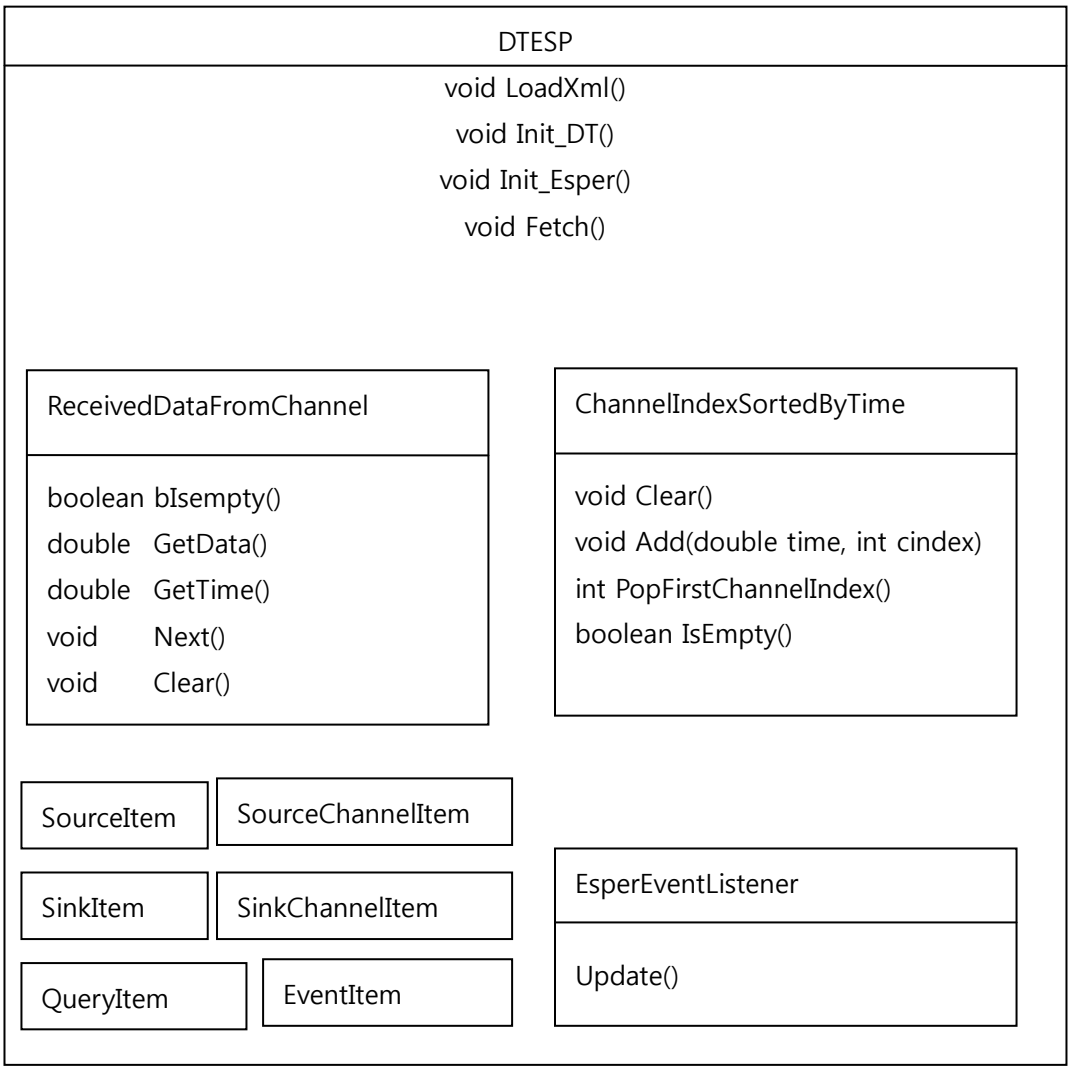
IV. Esper initialization
At method Init_Esper(), it iterates through data classes created by parsing xml file and setups Esper. For instance, it iterates through all QueryItem from the list which is created by II and creates Esper Queries.

V. Reading from Data Turbine and sending it Esper
At method Fetch(), it reads the sink channels and data from each channel is stored as one inner class ReceivedDataFromChannel. To send data to Esper in ascending time order, every earliest data of ReceivedDataFromChannel is inserted to inner class ChannelIndexSortedByTime which behaves as a sorted list. It always sends the first data in ChannelIndexSortedByTime so that data is sent in ascending time order over all channels.

VI. Listens to result of Esper queries and send to Data Turbine
EsperEventListener class is responsible for the task. When creating query at IV, if the result of the query need to be sent to Data Tubine, EsperEventListener is attached for each query for this task.

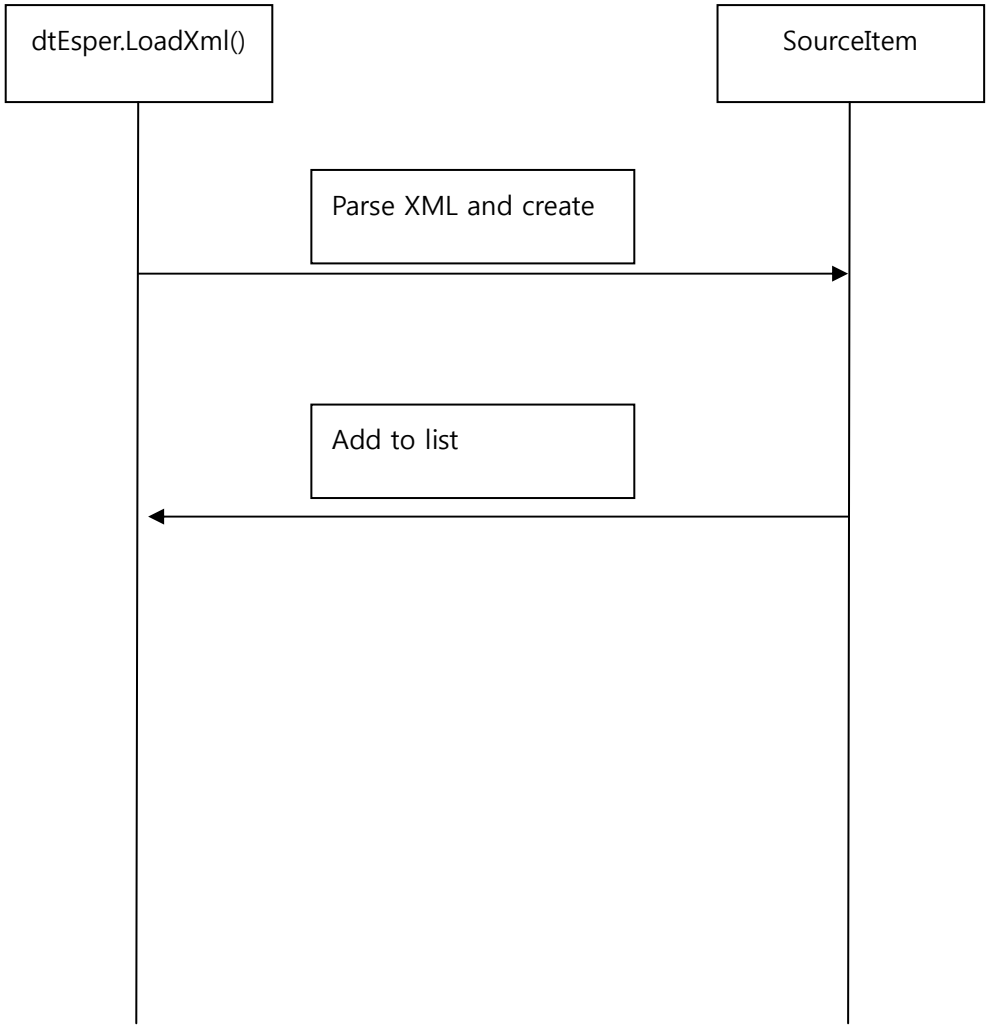
5. Entity diagram

ReceivedDataFromChannel, and ChannelIndexSortedByTime class is used in Fetch() method to send data in ascending time order.
EsperEventListener class is created in Init_Esper() method to receive result from Esper and send to Data Turbine.
SourceItem, SinkItem, SourceChannelItem, SinkChannelItem, QueryItem, and EventItem represents xml configuration.

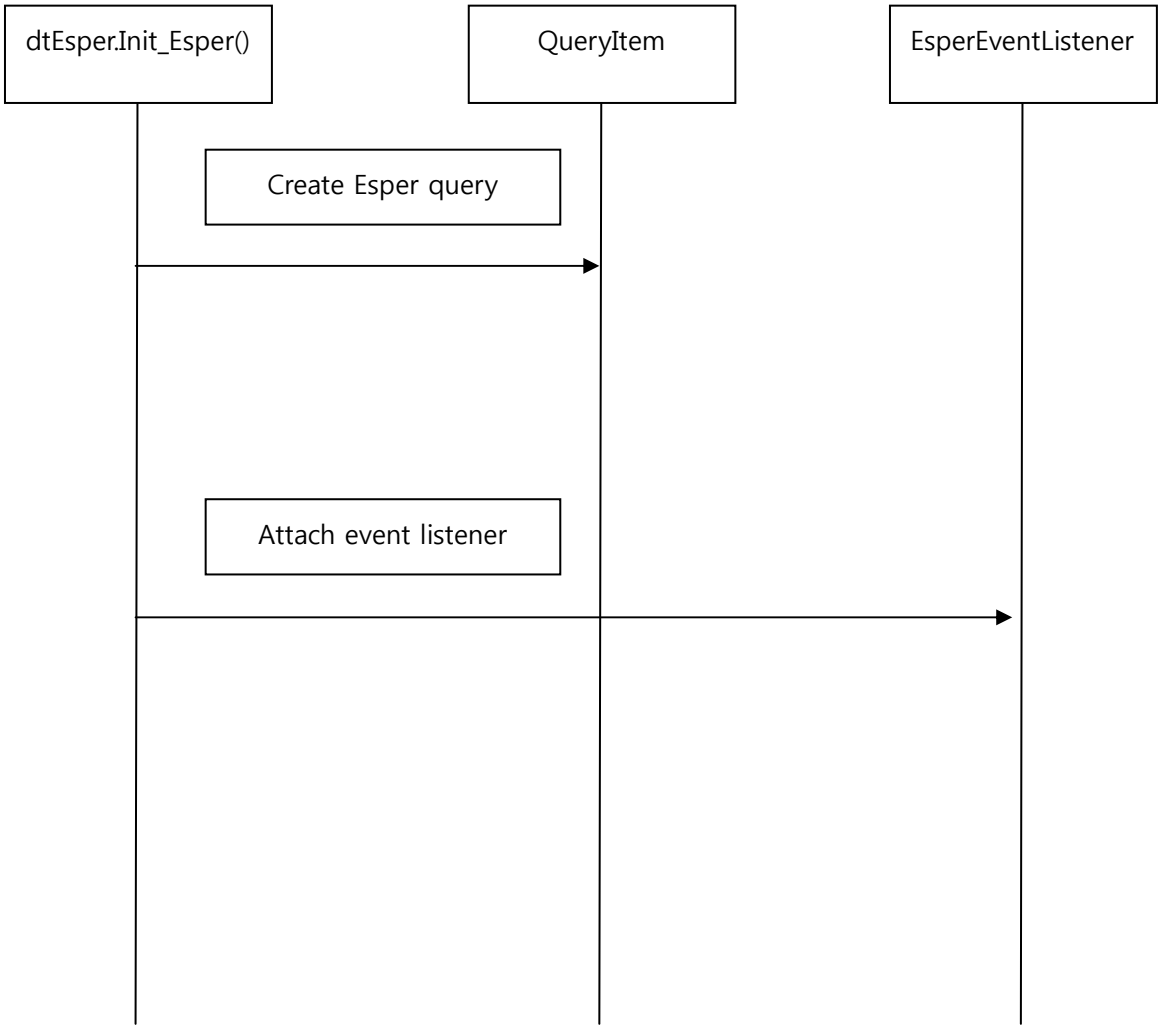


6. Interaction

[Example of parsing <Source> node]



[Example QueryItem and EsperEventListener in Init_Esper method]



[Example ReceivedDataFromChannel and ChannelIndexSortedByTime used in Fetch() method]