A Software framework for Procedural Knowledge based Collaborative Data Analytics for IoT

Snehasis Banerjee
TCS Research & Innovation
Tata Consultancy Services
Kolkata, India
snehasis.banerjee@tcs.com

M Girish Chandra
TCS Research & Innovation
Tata Consultancy Services
Bangalore, India
m.gchandra@tcs.com

***Abstract*—The outburst of data generation by machines and humans, along with emergence of sophisticated data processing algorithms have created a demand for a wide number of data analytics based services and applications. The paper presents a collaborative framework and system to carry out a large number of data processing tasks based on semantic web technology and a combination of reasoning and data analysis approaches using software engineering guidelines. The paper serves as a first step for systematic fusion of symbolic and procedural reasoning that is programming language agnostic. This approach helps in reducing development time and increases developer's productivity. The proposed software system's logical functionality is explained with the help of a healthcare case study, and the same can be extended for other applications.**

***Index Terms*—Procedural Reasoning, IoT Analytics, Software Framework, Software Orchestration**

I. Introduction

Internet-of-Things (IoT) [1] and the services around it is poised for a disruptive growth in near future mainly because of huge number of sensor deployments and advances in networking technologies. In fact, Gartner predicts 21 Billion IoT Devices to be deployed by 2020 [2]. As data is considered the new 'oil' (fuel) for innovations in Industry 4.0 [3], it has been found that data analytics on IoT is in huge demand across all business domains and services around it [4].

Among the various sensors that are deployed (apart from camera and allied sensors) most produce one-dimensional (1-D) readings (say temperature sensor) or multiples of 1-D readings in combination (say accelerometer sensor). Doing analytics on such type of sensor data can be mapped to modules of algorithms belonging to signal processing and machine learning fields. This can be abstractly formalized in a flow diagram for IoT Analytics (Fig. 1).

A typical IoT workflow will comprise (a) Data retrieval from sensors (b) Pre-processing the data by applying various methods like formatting, noise cleaning and anomaly removal (c) Applying data transformation to another representation form like Fourier transform or Wavelet transform (d) Applying Feature Engineering principles of feature extraction from a region of interest followed by selecting features by application of a set of feature selection algorithms (e) finding a suitable Model for the data and problem post tuning modeling algorithm parameters (f) optional inferencing step to derive high level deductions from discovered model (g) Visualizing the results (in a way domain experts can understand).

All of the aforementioned steps need a repository of algorithms for each module and invocation as per need. The traditional approaches for IoT Analytics have confined themselves to quantitative analysis of the data based on the problem at hand. A standing issue is that different suites of algorithms are available in different programming language platforms and this leads to manual intervention or asynchronous and sequential analysis of an IoT Analytics task. The work presented here tries to address this problem. Some work [5] [6] [7] has been done on realizing a data analysis framework for IoT, however they mostly lack the semantic angle and efforts to create a generic framework where problems of different domains can be plugged. Utilization of semantic technologies in IoT has been surveyed in [8], however, only a few IoT applications are found to utilize semantic technologies or in general symbolic techniques of analysis. In this work, an attempt has been made to combine the symbolic and quantitative techniques suitable for a typical IoT Analytics workflow execution. To make this happen, the proposed system has both a Quantitative Analytics Engine and a Symbolic Reasoning Engine that works in synergy by an implicit central controller mechanism. Another burden in a typical IoT Analytics problem is the variety of domain dependent solutions, which is overcome by separating the logic from generic procedures and storing them in rules, ontologies and knowledge stores. This lays foundation for a domain-independent general purpose reasoning based software framework for each IoT Analytics module. A major issue in a Reasoning enabled Analytics system is lack of support of procedural evaluations and keeping check of facts that have turned obsolete or false. This is overcome by (a) sticking to non-monotonous reasoning following the philosophy of truth maintenance systems [9] and (b) allowing seamless integration of procedural execution into the heart of reasoning following software engineering principles.

The main contributions of this work are:
(1) a collaborative software framework and system is presented that leverages semantic techniques (2) enabling seamless procedural evaluation embedded in semantic rules targeted for IoT data analysis tasks (3) a case study in health-care that illustrates the utility of this approach.

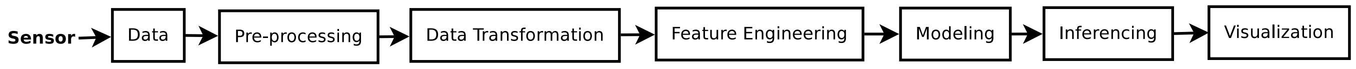

Fig. 1. A typical IoT Analytics Workflow most suited for one-dimensional sensor data

Section 2 discusses about the background for IoT Analytics work-flow from a knowledge based perspective. Section 3 describes the proposed system framework and section 4 shows the solution approach taken. Section 5 illustrates procedural reasoning approach of the system and its implications with a healthcare case study. Finally section 6 concludes the paper after laying down future scope of work.

II. Background

The semantic web technologies that can be used in developing IoT Analytics applications are enlisted here:

a) RDF[1]: A RDF triple, also called a fact, contains three components: (a) subject, which is an RDF URI reference or a blank node (b) predicate, which is an RDF URI reference (c) object, which is RDF URI reference, literal or blank node. An example in triple (<subject><predicate><object>) form: <sensor:TemperatureSensor><rdf:property><temp:Celsius>

b) Ontologies: used for defining class relationships and set theoretic constraints on relationships. This is typically manifested in OWL[2], a knowledge representation language in machine interpretable form. OWL enables reasoning from RDF enabled sources. Some of the well known ontologies in IoT space are OGC[3], SensorML of SWE[4] and SSN[5]; eg. :
<sml:input name="SurfaceTemperature"> <sml:
ObservableProperty definition="http://sensors.ws/ont/
NMMO/sensor/SeaSurfaceTemperature"/ > < /sml:input>

c) SPARQL[6]: used to query RDF models; eg. :
SELECT avg(?value) WHERE {
<sensor:TemperatureSensor> prop:hasValueInCelsius ?value}

d) Semantic Rules: Reasoning with rules is typically based on first-order predicate logic or Description Logic (DL) to make conclusions from a sequence of statements (premises) derived by predefined rules. A reasoner deduces facts from existing semantic data and ontologies based on predefined rules. Common reasoning and inference engines such as Jena[7] and Pellet[8] are based on different rule languages and have support for ontologies and OWL. Some of the reasoners support SWRL[9] and RIF[10] rule languages, whereas others have implemented their own human readable rule syntaxes. For reasoning on IoT Analytics workflow, Jena was found suitable for its support of custom rules and ease of implementation of extensions. A Jena Rule looks like:
[transitiveRule: (?P rdf:type set:TransProp)
(?A ?P ?B) (?B ?P ?C) − > (?A ?P ?C)]

e) Semantic Graph Databases: used when large amounts of facts are to be stored and processed either in batch or on-demand. These are also called RDF Databases and popular ones include Virtuoso[11], Jena TDB[12] and AllegroGraph[13].

A. Reasoning

Computer Reasoning relies on precise rules leading from a set of well-formed statements [10]. One way to classify reasoning approaches is in terms of monotonicity. In case of Non-monotonic reasoning (such as truth maintenance system), facts can become false in future, which directly relates to practical scenarios. This requires monitoring of facts so that when a fact becomes invalid because of a context change, facts derived as a consequence of that fact is removed and so on as a chain reaction. Reasoning techniques involves working with predicates and facts. A predicate on a domain, S, maps every element of S to either true or false, that is one of T,F, which can be viewed as a relation on $S \times \{T,F\}$. A function on a domain, S, maps every element of S to a unique element of the co-domain (or range), R, so can be viewed as a relation on $S \times R$. Any function, $f : S \rightarrow R$, can be converted to a predicate, $p : S \times R$ by defining $p(s,r) \Leftrightarrow f(s) = r$. Similarly, any predicate p:S can be converted to a function $f : S \rightarrow \{T,F\}$ by defining $f(s) = T \Leftrightarrow p(s)$ and $f(s) = F \Leftrightarrow \neg\, p(s)$. Functional predicates are useful when in a reasoning enabled system, procedural evaluation is needed to evaluate inferences based on dynamic facts.

B. Knowledge based Data Analytics

Knowledge has been found to play a crucial role in data analytics. Most of the time, the solutions are dependent on knowledge of domain, algorithms and application. The data analysts use these varied knowledge to filter out non-consistent models as well as apply correct steps in deriving inferences from data at hand. A survey [11] on Semantic Web and Data Analytics convey with examples how semantic web technologies can become an integral part of a typical data analysis workflow. It specifically mentions 3 types of ontologies namely (a) Domain: that exxpress background knowledge about the application domain, (b) Ontologies for data analytics process: that define knowledge about the data mining process, its steps and algorithms and their possible parameters, (c) Metadata ontologies: Describe meta knowledge about the data, such as provenance, format and source.

[1]Resource Description Framework: https://www.w3.org/RDF/
[2]Web Ontology Language: https://www.w3.org/OWL/
[3]Open Geospatial Consortium: www.opengeospatial.org
[4]Sensor Web Enablement: www.sensorml.com/standards.html
[5]Semantic Sensor Networks: w3.org/2005/Incubator/ssn/ssnx/ssn
[6]SPARQL Protocol and RDF Query Language: www.w3.org/sparql
[7]Jena under Apache License: https://jena.apache.org
[8]Pellet Reasoner: https://www.w3.org/2001/sw/wiki/Pellet
[9]Semantic Web Rule Language: www.w3.org/Submission/SWRL
[10]Rule Interchange Format: www.w3.org/TR/rif-overview
[11]OpenLink Virtuoso: https://virtuoso.openlinksw.com/
[12]TDB: https://jena.apache.org/documentation/tdb
[13]Franz AllegroGraph: https://allegrograph.com

Some recent works on knowledge based data analytics are: (a) [12] Publish-Subscribe based Knowledge Analytics that uses automated schema mapping to overcome data heterogeneity and uses semantic inferences on ontology enhanced data, (b) [13] shows how Knowledge graphs can aid in knowledge discovery for data mining, (c) [14] shows how formal semantics in ontologies can be incorporated into data analytics as the formal structure of ontology makes it a natural way to encode domain knowledge, (d) [15] describes an ontology-based system that deals with instance-level integration and data preprocessing. It uses a preprocessing ontology to store information about the required transformations.

III. System Description

The proposed system can be represented as a tuple $\{$ D, M, W, O, R, Q, K, A, L $\} \rightarrow$ I, where D = data, M = metadata, W = workflow templates, O = ontologies, R = rules, Q = queries, K = other knowledge sources, A = algorithm repository of registered procedural evaluation, L = learning algorithm recommendation knowledge and I = inferences drawn post application of reasoning and learning.

The major modules of the system (Fig. 2) are as follows:

1) Data Handler: the purpose of this module is to (a) handle bursts of data when running in live mode by using priority queues (b) using batch to process large data files when in offline mode. The module checks any inconsistency in data (such as mis matched rows and columns, data format support) and raises a flag to the user accordingly.

2) Workflow Manager: this dedicated module comprises of a workflow engine whose main task is to compose and execute data analytics workflows. This is achieved by composing an apt workflow from templates based on given data, meta-data and knowledge around it. The workflow manager controls the workflow instance in execution and can enable to and fro data and control exchange between Analytics Engine module and Reasoning Engine module. Further details of this module are kept out of scope to focus on the knowledge aspects.

3) Metadata store: this module contains both specific and generic metadata around algorithms used for analytics. Also meta-data related to given dataset is used as a knowledge input at different stages of analysis.

4) Analytics Engine: this module comprises of the list of analytics algorithm instances (such as those drawn from machine learning and statistical analysis) and their mappings to specific data analytics task. The workflow in execution links with this module when any data analytics task (such as classification) or sub task (such as data transformation) needs to be carried out.

5) Knowledge Store: this module consists of knowledge that is related to datasets, application, domain, workflow templates and algorithms. The knowledge can be in form of (a) ontologies such as OWL (b) facts such as RDF (c) formatted schematic files such as XML (d) flat files (such as csv) with tags or mappings for machine understanding of content (e) RDF stores (f) external databases with defined schema (for relational) or entities (for graph based).

6) Data Unifier: this module is responsible for unification of different data and knowledge formats into a consistent form, which is required for reasoning. Based on standard templates and keywords, mappings are carried out to represent data and knowledge into standard semantic form like RDF and OWL. One important thing in this respect is that the raw data is not converted into RDF form, only the meta-data around the data such as inherent columns and externally supplied metadata is. This saves overhead of re-converting the RDF facts to the initial numeric or categorical values. Each numeric value is tagged as a literal (such as string, double, etc.) in RDF, that adds individual literal tags to each instance of data value. RDF schemas are descriptive such as vocabulary definition, not prescriptive like relational schemas where data must take the prescribed form. This will create a large RDF data file - hence this approach is avoided. The pointers to data sources are kept as RDF facts in the current working memory. However, aggregation of data values that may lead to meaningful inferences can be stored as RDF facts such as mean, median, standard deviation of data fields.

7) Ontology Repository: this module maintains a list of references to all ontology type knowledge files with description. Users are free to add their own custom ontologies or link to an ontology on the web.

8) Rule Repository: this module contains all rules registered in the system via Rule Registry, including procedural rules. Rules have names and are mapped to general usage or specific applications so that selective invocation can be done on demand. System has a large repository of in-built rules which include set theoretic operations as well as commonly used logical expressions. Rules need to be binded to a Reasoner instance for execution. There is a provision to add custom rules including those having procedural functions.

9) Query Repository: this module contains queries which are used in order to view the inferences drawn via rule firing. The repository is pre-filled with simple template SPARQL like semantic queries which can be extended by user as per application demand. Some logging queries are pre-registered that track execution of the Reasoning engine.

10) Reasoning Engine: this important module is concerned with three principal tasks: a) supporting procedural evaluation when firing rules b) provision of apt workflow recommendation based on inferences drawn on given problem and dataset c) algorithm selection for each of the steps of a workflow in execution. The 'Knowledge Bus' connects the Reasoning module to the different knowledge repositories discussed above. A Listener service is registered in the Knowledge Bus that checks if any change occurs in any of the repositories. A provision is kept to store inferred knowledge and facts in their apt repositories for future use.

Reasoning engine is composed of the following sub-modules:

(a) Working Memories: all the relevant facts and ontologies are loaded in a dedicated memory space which is called working memory in computer reasoning terminology. It usually has a limited capacity that is responsible for temporarily holding

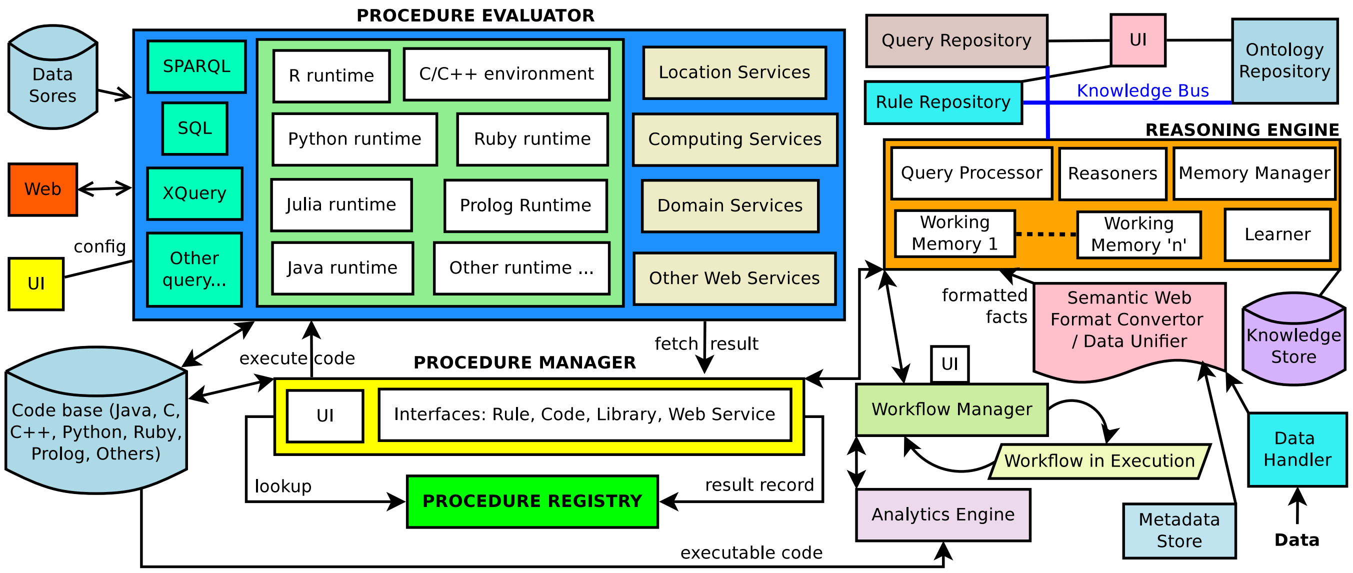

Fig. 2. Proposed Collaborative Framework for Data Analytics based on Hybrid Knowledge Processing

information available for processing. With advancements in distributed computing and memory technology, it is possible to hold large amounts of facts in memory for fast processing. Facts (or tuples) lying in a working memory under the non-monotonous reasoning paradigm are considered as true, and obsolete facts are deleted from it. In other words, working memory represent the current logical state of a system. The patterns of facts loaded in a working memory is based on the patterns existing in left hand side of rules (productions) and query expressions for a specified workflow execution.

(b) Query Processor: this module supports two types of queries: 1) Ad-hoc: this can be issued by users on demand to query the working memory for situations like result visualization 2) Registered queries: this are pattern seeking queries already linked to the working memory and trigger at regular intervals or on some condition satisfaction.

(c) Reasoners: this module consists of a library of 1) predefined standard reasoners such as RDFS or OWL reasoners, where logical inferences are fixed 2) custom reasoners made by binding custom rules including procedural rules to embed logic as per need. A choice of hybrid, forward and backward chaining are available to the user to choose. If the number of rules for a problem is large, then forward chaining is better as rules get triggered only when matching patterns in data are observed. Forward chaining being data driven, rule processing continues until no more rules can be applied or some cycle limit is met. As this can generate a huge number of inferences (new RDF facts) that can bloat working memory, forward chaining is advised for usage in event driven scenarios. Backward chaining on the contrast is a goal driven inference technique. It starts from possible conclusion or goal and aims for necessary data. This makes backward chaining suitable when number of rules are few and when working memory has limitations in capacity. For small datasets and rules, the earlier two approaches or a hybrid one usually suffice, but for large amounts of data and large number of rules, Rete is the optimal choice of reasoning. The Rete algorithm provides a generalized logical description of an implementation of functionality responsible for matching facts against rules in a pattern-matching production system, ie. a rule engine. Rete avoids iterating through the data elements by storing the current contents of the conflict set in memory, only adding and deleting items form it as data elements are added and deleted from memory. Rete algorithm is designed to sacrifice memory (by creating and maintaining Rete Network of Alpha and Beta Nodes) for increased speed. Rete has three components: 1) Alpha Network - left side of the node graph (rule patterns) that forms a discrimination network and selects elements from working memory based on simple conditions 2) Beta Network - the right side of node graph which mainly performs joins between elements in working memory, 3) Agenda - typically implemented as prioritized queues, that finds the order in which a rule is fired in a match-resolve-act cycle.

(d) Memory Manager: this module retrieves resources from various repositories via Knowledge Bus. It manages the working memories and allocates and deallocates memory areas to individual reasoner instances based on need. What to load in the working memory is defined based on the dataset, application and problem description set by the user, apart from support of direct manual loading from the user's end. The system dedicates a working memory for each application instance. Due to the nature of knowledge and applications, it is often found that there is a significant overlap of the knowledge needed across use cases. Hence, keeping redundant facts across disjoint working memories adds to overhead in processing as well as space. This can be overcome by having multiple working memories with sharing support [16].

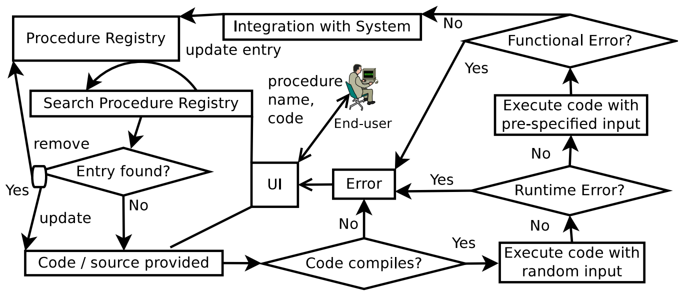

Fig. 3. Flowchart of Procedure Update Validation

(e) Learning Module (Learner): This module learns based on execution logs of the reasoner. System learns which facts and rule bindings need to be kept in memory in cache based on frequency of usage.

11) Procedure Manager: this module is the central control to manage procedural evaluations while reasoner is in execution. When a new procedural rule needs to be registered, the rule binding happens in the reasoner (implicit reasoner registry) while the procedure mapping happens in Procedure Registry. Procedure Registry maintains list of algorithm codes with pointer, names, arguments, parameters and meta-data. Users can define their own procedures to extend the built in functions of the reasoning vocabulary of keywords and pointers to executable methods. Fig 4. illustrates the user's perspective on semantic rule formation as per demand of the solution logic. The registry contains following types of functions (methods or procedures):
a) Standard functions: some frequently used functions like standard deviation b) Defined functions: functions added by the user c) Shared functions: functions added by other users and shared for re-usability. d) Web Service functions: web service APIs registered in system e) External functions: external tools and database connections binded to registry.

12) Code Base: the executable files are kept here. Support for various languages such as Java, C/C++, Ruby, Python, Julia, Prolog, R is kept as a) different languages has different functionality that can be leveraged, b) users usually have their own preference of languages, c) different library of functions are available for each language and a specific required code library may be available only in one language.

13) Procedure Evaluator: this is composed of runtime environments of different programming languages, where the values are passed and code gets executed. Connectivity to external databases (such as those listed in LOADB[14] and DBPedia[15]) supporting queries via SPARQL, SQL and other querying methods are maintained to link aptly at the time of procedural evaluation. Various services and their APIs are kept registered to use the functionalities offered. Services include Location (like Google Distance Matrix API), Computing (like evaluation via Wolfram Alpha) and Domain (like information around disease outbreaks). Inter connections between runtimes are kept to minimize context switching.

[14]http://www.loadb.org
[15]https://wiki.dbpedia.org/

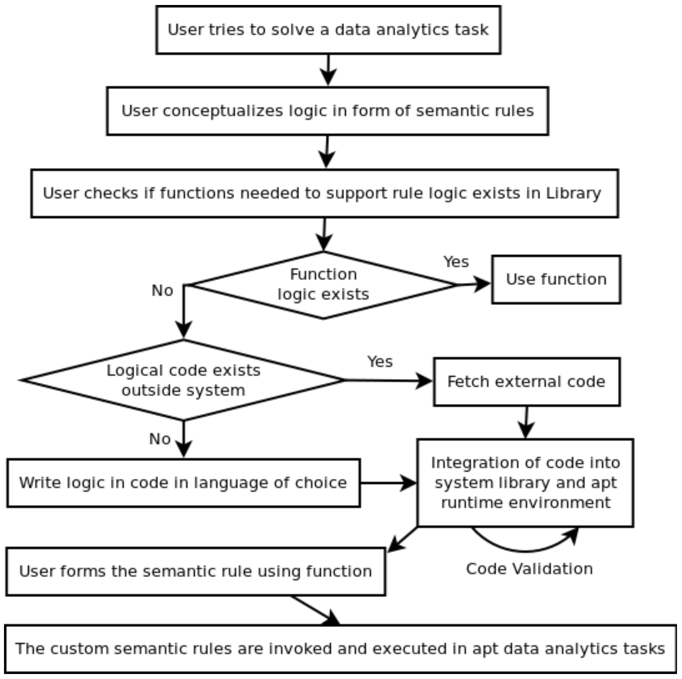
User tries to solve a data analytics task
User conceptualizes logic in form of semantic rules
User checks if functions needed to support rule logic exists in Library
Function logic exists
Yes
Use function
No
Logical code exists outside system
Yes
Fetch external code
No
Write logic in code in language of choice
Integration of code into system library and apt runtime environment
User forms the semantic rule using function
Code Validation
The custom semantic rules are invoked and executed in apt data analytics tasks

Fig. 4. Flowchart of custom semantic rules formation from user perspective

The system has multiple front end user interfaces binded to specific modules that serve as means to update files, configurations and defining relations and placeholders in the system. Due to limited space, discussion on UI is skipped.

IV. Method Description

The proposed system's underlying principle has been chosen to be non-monotonic forward chaining deductive reasoning supporting functional predicates to suit needs of IoT workflows. Logic can be expressed both in form of semantic rules and SPARQL like graph pattern queries. However, to maintain the truth maintenance aspect, logic in this case is restricted to semantic rules having provision for fact removal. SPARQL is used as a way to view and retrieve answers to queries (patterns under scrutiny). Rete [17] and its improved variations [18] [19] have stood the test of time for handling fast pattern matching in a non-monotonous reasoning scenario. For IoT based workflow execution, knowledge decisions based on Jena's implementation of Rete forward reasoner was found suitable as Rete is able to handle large number of rules and fast fact deletion based on its alpha-beta networks and efficient conflict resolution strategies.

A. Handling External Procedures

External procedures in any supported programming language can be added to the system. Fig 3. shows how a user can register a procedure to be binded in Procedure Registry. User starts with searching for a procedure name in the existing registry and if entry found, can view the code if it is public and can update (also remove) the same if access rights are granted. Else, user creates a new entry of procedure by linking or uploading code in language of choice. The system checks (in the runtime environment) if the code gets compiled and executes code with random input to see if any runtime errors or exceptions happen. Next, system asks user to provide a set of input and expected output and sees if the procedure yields functionally correct results. Next, the procedure gets integrated with the Registry and a user will be able to use the function names and arguments in defining custom rules. Any errors in the process are shown to user for rectification.

B. Collaborative Development

The users can do the following in the UI connected to Procedure Registry: a) Updating / adding/ removing custom functions (algorithmic methods) b) Sharing functions with all users or a group or keeping it private c) Viewing source code or other implementation details of shared functions with apt permissions d) Commenting, rating and providing suggestions for different Registry functions e) Tag the self created functions with meaningful meta-data f) In the particular session, users can turn on and off certain functions. This comes handy in case of overriding functions like the different interpretations of a function bearing the same name.

Fig 4. illustrates the user's perspective on semantic rule formation as per demand of the solution logic. In order to solve a data analytics task, user thinks of the problem at hand and how to conceptualize a solution about it. Usually the path taken to solve that problem can be thought of semantically connected steps whose logic can be expressed in rules. User next checks in the system where building blocks of rules (combining which a complete rule expression will be formed) exist in the system, that were registered by other developers. If a building block (in this case, a functionality) exists in the rule processing system, then the user composes a rule based on logical requirement of the task. Else, the user is given a choice to add required procedural logic as building blocks of rules to be added to the system. If the procedure already exists outside the system, user can just fetch the code and let the system integrate it automatically. Else the user can write the code in an editor and integrate in the system. Code is validated before registering the function with a name in the system. User forms a set of rule expressions to execute the same in the system to serve a specific task on a given data.

Functions or methods stored in the system are analyzed about their time and space requirements, frequency of usage, modification, extension and other details. This aids in creating effective recommendation of functions when a user searches the Registry for existing procedural functions. User created functions can be rated, shared and extended to for a collaborative development ecosystem. Functionality of added functions can be tested and validated, with feedback given to the initial procedural code provider.

System creates a new URI (uniform resource identifier) for the new function being added with its programming language as a sub part if URI. This helps in distinguishing functions with same names based on domain and nomenclature. This URI assignment is done automatically following standard pre-defined standard templates. An example is shown:- http://com.algorepo/number/c/2/1/resample
where number is the domain, c represents programming language; 2 shows number of input arguments; 1 means number of declared return values; finally the name of the function. System creates a field and knowledge store entry to check the number of accesses to the calling function (this is updated on each call as hit) to monitor popularity in collaborative platform. Another field is created to monitor average memory and time requirement the given function consumes. A trailing number is added to the function name at the end of generated code that can be used to call the function when writing the rule so that the user do not have to type the whole URI to refer to that code. This comes as a overlay in the UI when function is used in a rule. Example: *resample2*, when two functions (may be expressed in different languages) already exist. This works like: function name + previous counter value + unit increment.

V. Procedural Reasoning

While using Semantic Web technology for reasoning covers the description logic aspects, enabling procedure calls from bounded variables while reasoner is executing makes a reasoner powerful in handling procedural reasoning. Some example procedural reasoning scenarios along with corresponding logic expressions in rule syntax is shown with the help of a healthcare use case.

The task of 2016 PhysioNet / CinC Challenge[16] was to classify heart sound recordings (phonocardiogram / PCG) into normal and abnormal recordings. Main steps of the task are pre-processing, segmentation and modeling (classification). The following logic snippets show the approach's usefulness for pre-processing step for illustration purpose.

A section of domain knowledge used for the problem:
1. <problem:heartSound><sig:windowSizeInSeconds> 5
2. <problem:heartSound><sig:upperBoundFreqInHz> 400

A section of meta-data facts supplied for the problem:
1. <data:dataInstance><sig:samplingRateInHz> 2000
2. <data:dataInstance><file:format><file:wav>
3. <data:dataInstance><data:hasSource> 'file path'

A section of algorithmic knowledge stored in ontology:
1. <sig:MedianFilter><sig:type><sig:NonLinearFilter>
2. <sig:LowPassFilter><sig:type><sig:LinearFilter>
3. <sig:LinearFilter><sig:type><sig:Filter>
4. <task:PreProcessing><task:subTask><sig:Filter>

A section of procedural rules (custom define) used:
1. [AutoResample: (?data <data:hasSource> ?url) . (?data <problem:category> ?p) . (?data <sig:samplingRateInHz> ?val1) . (?data <sig:upperBoundFreqInHz> ?val2) -> c:resampleA(?url, ?val1, ?val2, ?urln) . (?p <task:completion>
<sig:AutoResample>) . (?data <data:newSource> ?urln)]
2. [LowPassFilter: (?data <data:hasSource> ?url) . (?data <problem:category> ?p) . (?p <sig:upperBoundFreqInHz>

[16]https://www.physionet.org/challenge/2016/

?val) -> c:lowpass(?url, ?val, ?urln) . (?p <task:completion> <sig:LowPassFilter>) . (?data <data:newSource> ?urln)]

The above procedural rule no. 1 attempts to resample data using a custom algorithm. Resampling data to a lower rate without missing patterns is recommended to enable faster processing. The auto resampling algorithm takes in sampling rate and the upper frequency range with in which meaningful frequencies exist and using principles of Nyquist Theorem and sampling [20], resamples data to an optimal sampling rate. The rule takes the data instance source, checks the problem tagging and sees if any facts about sampling rate and upper bound of frequency exists, calls the autoresampling algorithm (c: is the initial prefix notation to denote custom procedural calls) coded and registered by user and outputs completion fact and processed data source. Procedural rule no. 2 takes a data source, checks the problem tagging and sees if any domain knowledge of cutoff frequency exists, and next does a procedural call to low pass filter algorithm and gives out a link to processed data as well as adding a fact in working memory that lowpass filter operation is done. In this case, an available algorithm is used that is found in standard signal processing libraries of Matlab or Python. Hence it is showcased how symbolic and procedural knowledge based processing can be used for data analysis tasks.

VI. Conclusion

The paper has presented a collaborative framework and system to carry out a large number of data processing tasks based on semantic web technology with support of procedural rules that is programming language agnostic. Example procedural rules were explained with the help of a healthcare use case. Future work will include extensions to 2-D (image) and 3-D (video) sensor data. Fast and incremental reasoning is a need of the system. Hence, to suit event-based IoT scenarios an improved Rete [19] [18] algorithm in the reasoning module is planned for integration. Finally, the system will be tested for different Analytics pipelines to gather further information on usability of procedural rules. The approach reuses other developer's efforts in semantic rule formation, thereby reducing development time.

References

[1] P. Sethi and S. R. Sarangi, "Internet of things: architectures, protocols, and applications," *Journal of Electrical and Computer Engineering*, vol. 2017, 2017.

[2] N. Eddy, "Gartner: 21 billion iot devices to invade by 2020," *InformationWeek, Nov*, vol. 10, 2015.

[3] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business & Information Systems Engineering*, vol. 6, no. 4, pp. 239–242, 2014.

[4] M. S. Mahdavinejad, M. Rezvan, M. Barekatain, P. Adibi, P. Barnaghi, and A. P. Sheth, "Machine learning for internet of things data analysis: A survey," *Digital Communications and Networks*, 2017.

[5] M. Strohbach, H. Ziekow, V. Gazis, and N. Akiva, "Towards a big data analytics framework for iot and smart city applications," in *Modeling and processing for next-generation big-data technologies*. Springer, 2015, pp. 257–282.

[6] K. Hwang and M. Chen, *Big-data analytics for cloud, IoT and cognitive computing*. John Wiley & Sons, 2017.

[7] D. Pizzolli, G. Cossu, D. Santoro, L. Capra, C. Dupont, D. Charalampos, F. De Pellegrini, F. Antonelli, and S. Cretti, "Cloud4iot: A heterogeneous, distributed and autonomic cloud platform for the iot," in *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2016, pp. 476–479.

[8] P. Barnaghi, W. Wang, C. Henson, and K. Taylor, "Semantics for the internet of things: early progress and back to the future," *International Journal on Semantic Web and Information Systems (IJSWIS)*, vol. 8, no. 1, pp. 1–21, 2012.

[9] H. Beck, "Reviewing justification-based truth maintenance systems from a logic programming perspective," Tech. Rep. INFSYS RR-1843-17-02, Institute of Information Systems, TU Vienna. July, Tech. Rep., 2017.

[10] R. S. Michalski, "Inferential theory of learning as a conceptual basis for multistrategy learning," *Machine Learning*, vol. 11, no. 2-3, pp. 111–151, 1993.

[11] P. Ristoski and H. Paulheim, "Semantic web in data mining and knowledge discovery: A comprehensive survey," *Web semantics: science, services and agents on the World Wide Web*, vol. 36, pp. 1–22, 2016.

[12] C. Esposito, M. Ficco, F. Palmieri, and A. Castiglione, "A knowledge-based platform for big data analytics based on publish/subscribe services and stream processing," *Knowledge-Based Systems*, vol. 79, pp. 3–17, 2015.

[13] P. Ristoski, "Exploiting semantic web knowledge graphs in data mining," Ph.D. dissertation, 2018.

[14] D. Dou, H. Wang, and H. Liu, "Semantic data mining: A survey of ontology-based approaches," in *Semantic Computing (ICSC), 2015*. IEEE, 2015, pp. 244–251.

[15] D. Perez-Rey, A. Anguita, and J. Crespo, "Ontodataclean: Ontology-based integration and preprocessing of distributed data," in *International Symposium on Biological and Medical Data Analysis*. Springer, 2006, pp. 262–272.

[16] S. Banerjee and D. Mukherjee, "Towards a universal notification system," in *Proceedings of the 2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)-Volume 03*. IEEE Computer Society, 2013, pp. 286–287.

[17] C. L. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem," in *Readings in Artificial Intelligence and Databases*. Elsevier, 1988, pp. 547–559.

[18] B. Berstel, "Extending the rete algorithm for event management," in *Temporal Representation and Reasoning, 2002. TIME 2002. Proceedings. Ninth International Symposium on*. IEEE, 2002, pp. 49–51.

[19] T. Gao, X. Qiu, and L. He, "Improved rete algorithm in context reasoning for web of things environments," in *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*. IEEE, 2013, pp. 1044–1049.

[20] E. J. Candes, J. Romberg, and T. Tao, "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information," *IEEE Transactions on information theory*, vol. 52, no. 2, pp. 489–509, 2006.