A Semantic-enabled Framework for Future Internet of Things Applications

(Invited Paper)

Umesh Bellur*, Pankesh Patel†, Saurabh Chauhan‡, and Yongrui Qin§,
*Indian Institute of Technology Bombay, India
umesh@cse.iitb.ac.in
†ABB Corporate Research, India
pankesh.patel@in.abb.com
‡Rklick Inc, India
chauhan.saurabh.b@gmail.com
§University of Huddersfield, UK
Yongrui.Qin@hud.ac.uk

***Abstract*—While the challenge of connecting Internet of Things (IoT) devices at the lowest layer has been widely studied, integrating and interoperating huge amounts of sensed data of heterogeneous IoT devices is becoming increasingly important because of the possibility of consuming such data in supporting many potential novel IoT applications. A common approach to processing and consuming IoT data is a centralized paradigm: sensor data is sent over the network to a comparatively powerful central server or a cloud service, where all processing takes place. However, this approach has some limitations as it requires devices to interact directly with a cloud which is not cost effective. First, it has high demands on the device's storage and computational capabilities. Second, as devices grow rapidly in a deployment area, sending all the data to a centralized cloud server requires high network bandwidth. Moreover, this often creates data privacy concerns as all raw data will be sent to a centralized place.**

To address the above limitations for building future Internet of Things applications, we present an early design of a novel framework that combines Internet of Things, Semantic Web, and Big Data concepts. We not only present the core components to build an IoT system, but also list existing alternatives with their merits. This framework aims to incorporate open standards to address the potential challenges in building future IoT applications. Therefore, our discussion revolves around open standards to build the framework, rather than proprietary standards.

I. Introduction

The Internet of Things (IoT) connects everyday physical objects and devices (such as washing machine, car, etc.) to the Internet [1], [2]. These devices share data about their surroundings via sensors and may carry actuators so that they could be remotely controlled by their users via smartphone applications. The integration of the large amount of sensed data generated by IoT devices is becoming increasingly important. This is in part because of the many different ways in which this data can be consumed through a number of novel scenarios (such as smart metering, smart electric car recharge stations, retail & logistics, and so on) that help people to achieve their goals and organizations to improve their business processes [3].

However, the true potential of IoT applications is yet to be realized and currently we see a major gap for building applications that connect the physical and cyber worlds. It requires a tremendous amount of manual efforts to integrate heterogeneous information and develop cross-domain IoT applications [4], [5]. The major requirements for fast development of IoT applications are -

Heterogeneity. IoT applications operate over an infrastructure consisting of a wide variety of heterogeneous devices, ranging from powerful devices (such as server, smart phones, laptop, raspberry pi[1]) to small devices (such as Arduino UNO[2]) operated by micro-controllers [6]. The powerful devices host common operating systems, implement common protocols and frameworks and offer powerful multimodal communication capabilities while small resource constrained devices exhibit different characteristics such as lightweight operating system (e.g., TinyOS[3], Contiki[4]) and operate on weaker communication protocols (e.g., Bluetooth, ZigBee).

To address heterogeneity issues, a commonly accepted technique is to design a common service-oriented framework/ middleware [7] and implement application logic using general-purpose programming languages (such as C, JavaScript, Java, and so on). This may be supplemented using rapid application development tools such as NodeRED[5] or Model-driven Development (MDD) tools such as IoTSuite [8][6]. While solutions exist for tackling issues for the powerful devices, these solutions cannot be directly applied for resource-constrained small devices [9]. As a result, the integration of such resource constrained devices into the IoT ecosystem remains difficult.

Interoperability. IoT devices are often not interoperable [10]. They follow different network protocols and exchange data using proprietary data formats. The second problem is vertical silos of IoT application development [11] - most of the available IoT solutions are designed for a single problem

[1]https://www.raspberrypi.org/
[2]https://www.arduino.cc/en/main/arduinoBoardUno
[3]https://en.wikipedia.org/wiki/TinyOS
[4]http://www.contiki-os.org/
[5]https://nodered.org/
[6]https://github.com/pankeshlinux/IoTSuite

domain, resulting into vertical application development for targeted applications and lacking interoperability, re-usability and resource sharing among IoT applications [4].

To address the interoperability challenges, in recent years there has been the use of semantic web technologies for IoT devices [12]. Semantic Web [13] extends the Web with machine interpretable representations, thus making data integration simple and establishing interoperability among different IoT devices. The semantic Web concept uses semantic technologies such as Resource Description Framework (RDF) to annotate sensor data, and uses RDF schema (RDFS) and ontologies that provide domain knowledge and vocabulary for modeling and describing RDF data [14]. Applying such semantic technologies can automate information retrieval and decision making, thus facilitate the development of various advanced and cross-domain applications. However, very few IoT applications currently utilize semantic technologies.

Scalability. It is expected that billions of smart devices will be generating data, much of which may need to be processed in real-time. This poses unprecedented challenges in dealing with huge amounts of streaming, uncertain, incomplete and redundant data in IoT [15]. Therefore, scalability in data processing and consumption will be of paramount importance to exploit the full potential of IoT data that may come from millions of data sources. To handle this scalability issue, challenges must be addressed in many aspects and at different levels (e.g., data cleaning, transforming, compression, stream processing, multi-streams processing, scalable storage and indexing, reasoning, and so on) [16].

Actionability. Actionable knowledge is expected to be generated from the results of scalable IoT data processing [17]. For example, instant reactions to emergencies or disasters will rely on actionability of IoT systems. In such contexts, actionability will require a real-time understanding about IoT data, in terms of underlying knowledge and implications from IoT data, which will be critical to plan real-world actions for handling any emergencies or disasters.

Cloud-based IoT approaches. There has been a recent proliferation of the use of cloud-based IoT approaches (such as AWS IoT[7], IBM Bluemix[8], Azure IoT Suite[9]). A common approach is to use a centralized paradigm: sensor data is sent over the network to a powerful central server, where all the processing takes place and appropriate decisions are taken to control actuators (e.g., controlling temperature using a heater). This approach delegates application development efforts and reduces the maintenance costs. However, the centralized approach has the following limitations [18], [19]:

- The centralized cloud approach has been proven to work well for scenarios where sensing devices have a reasonable amount of processing power and reliable network connections. However, it does not always hold in reality when a system is deployed in an environment such as fire monitoring or wildlife monitoring in forest. Moreover, requiring all devices to interact directly with cloud service is not cost effective solution because it requires resource intensive processing and complex protocols.
- As devices grow in the system, the overhead required by the centralized system can be very high. Sending all the data to a centralized cloud server requires prohibitively high network bandwidth. This is compounded by data privacy concerns and regulatory frameworks that prohibit the sensor data from crossing certain geographic boundaries.

It is critical to address the above requirements in building future Internet of Things applications. To this end, we present an early design of a novel semantic-enabled framework for building future IoT applications. This framework incorporates three layers: Physical Layer, Cloud Layer and Application Layer. In each layer, we present the core components and list existing alternatives with their merits. This framework aims to incorporate open standards to realize the potentials of future IoT applications and to avoid developing using proprietary solutions. Therefore, our discussion centers around open standards in each layer, rather than proprietary standards.

Outline. The remainder of this paper is organized as follows: Section II describes a conceptual design of a framework. We present relevant design issues and research questions to implement such a framework in Section III and conclude in Section IV.

II. Architecture

We believe that a system that enables good decision making and actions needs to be layered and towards that end we present an architecture in Figure 1 that is divided into three layers:

A. The Physical Layer

This layer consists of devices ranging from resource constrained devices to more powerful devices. These devices are responsible for sensing, collecting sensed data, and communicating data to the outside world. The sensed data is generally in a raw format and does not provide any explicit information primarily because there may not be enough processing capability and background knowledge. However, in order to address data interoperability problem, it is essential to annotate data before being sent. The data annotation task can be delegated to more powerful devices nearby that can be named as a `gateway`.

Data representation. A popular data representation format in the semantic web, such as RDF, can be used as a data exchange format for IoT devices. However, IoT systems may involve small devices with a limited computing capability, and memory and communication constraints, the semantic web approaches often introduce challenges for the small devices that may not be present in the common scenarios of Semantic Web [20]. The work [20] emphasizes adding semantic web technologies for IoT devices and evaluates a number of different semantic representations for representing sensor measurements and device parameters in terms of energy efficiency for

[7]https://aws.amazon.com/iot/

[8]https://www.ibm.com/cloud-computing/bluemix/interet-of-things

[9]https://www.microsoft.com/en-in/internet-of-things/azure-iot-suite

data communication and processing. The authors evaluation finds JSON for Linked Data[10] (JSON-LD) and Entity Notation (EN) [21] is a compact and lightweight representation of RDF. Many non-RDF lightweight emerging standards are available for representing sensor measurements and device parameters. However, to enable intelligent functions such as reasoning and querying over sensor data, RDFizer must be implemented at the Gateway that can transform these non-RDF standards to the standardized data format such as RDF.

RDFizer. This component transforms varying formats to the standardized RDF format, enabling reasoning over sensor data in a uniform way. The work [22] presents an approach that transforms Sensor Markup Language (SenML)[11] formats to RDF. SenML is an industry-driven lightweight solution for representing sensor measurements and device parameters and it is being accepted by many vendors. SenML supports compact formats such as JSON[12] and Efficient XML Interchange (EXI)[13] for resource constrained devices [23].

RDF storage and processing. Once the data is converted into RDF standard, it needs to be stored for further processing. There are two possible approaches to store and process RDF data: the first approach is that RDF data can be transmitted through standard protocols to the cloud-layer for further processing, as proposed in work [12]; the second approach could be storing and processing data locally on device. There are many advantages of the latter approach [24] such as (i) scalability can be achieved at the device-layer because it distributes the computation among the large number of devices, (ii) the data transmission cost from devices to the cloud layer can be reduced because a device has to send only final results to the cloud-layer, and (iii) the local processing of data contributes to privacy as only processed data is sent rather than sending raw data points. In general, the storage of RDF on resource-constrained devices such as micro-controllers is not possible due to the textual representation of RDF (such as plain-text XML). Binary XML format is developed to overcome this problem for the constrained devices. EXI format is a promising compact and binary representation of the XML, proposed by the W3C. RDF on the Go [24] one of the first approaches that offers a full-fledged RDF storage for Android mobile device. RDF data is stored in the B-Trees provided by the lightweight version of Berkeley DB for mobile device. Apart from this work, other efforts are microJena[14] and MobileRDF[15] that store and query RDF data locally.

Reasoning. As a key enabling step for on-device processing, it is essential to push reasoning (it is a way to acquire new knowledge from RDF data) towards the edge of system, where various gateway devices are deployed for data collection and data aggregation. However, it is not clear whether existing reasoning engines (e.g., Jena[16], Pellet[17], RacerPro[18], Fact++[19]) could be used for devices, and how they would perform in resource-constrained devices [19]. The study in work [25] shows that reasoning engines take several hundred of Kilobytes of memory to reason each RDF triple. Thus, technically while it is possible to port a reasoner on devices with some code-level modifications, a reasoner can still consume vast resources of gateway devices [19].

B. The Cloud Layer

This layer hosts powerful servers and receives data streams from Physical layer through standard TCP/IP and provides suggestions that can be used by application layer.

Data ingestion. It is an entry point of getting data into cloud layer. It has two major roles: (1) scale to meet the demand of data producers (e.g., IoT devices and gateways) and (2) move data as fast as possible to the next component for further processing. It collects sensor data in various formats (e.g., JSON, EXI). Typically, the data collection is accomplished by querying sensors or receiving data streams from devices. Regardless of protocols used by IoT devices to send data to the data ingestion component, a few number of interaction patterns exist such as request/response [13], publish/subscribe [26], and stream [27]. To scale the ingestion service, frameworks such as Apache Kafka[20] and Amazon Kinesis[21] use a load balancer that can route a request from IoT devices to a running instance of the ingestion service. After collection, data are transported to other components through various message queuing protocols (e.g., ActiveMQ[22], RabbitMQ[23], and Apache Kafka) for further use.

Data preprocessing and enriching. Once data is ingested into the system, it is expected that data from Physical layer requires preprocessing before it becomes valuable for further analysis. Hence, the preprocessing must be done before the actual analytics takes place [28]. Secondly, data from Physical layer lacks background information, thus enriching data with background and additional domain-specific knowledge are important tasks [29].

Storage. It receives data from ingestion service and stores it for further use. Different types of storage architectures are possible based on a purpose and data format. For instance, RDF data from physical layer could be stored in triple store[24]. Moreover, to facilitate semantic reasoning further, the triple storage stores various ontologies, datasets, and rules. In a second scenario, it may be possible that data could be stored in traditional DBMS system such as RDBMS, HBase[25],

[10]http://json-ld.org/
[11]https://tools.ietf.org/html/draft-jennings-senml-10
[12]http://www.json.org/
[13]https://www.w3.org/TR/exi/
[14]http://poseidon.ws.dei.polimi.it/ca/?page_id=59
[15]http://www.hedenus.de/rdf/
[16]https://jena.apache.org/documentation/inference/
[17]https://www.w3.org/2001/sw/wiki/Pellet
[18]https://www.w3.org/2001/sw/wiki/RacerPro
[19]http://owl.man.ac.uk/factplusplus/
[20]https://kafka.apache.org/
[21]https://aws.amazon.com/kinesis/streams/
[22]http://activemq.apache.org/
[23]https://www.rabbitmq.com/
[24]https://www.w3.org/wiki/LargeTripleStores
[25]https://hbase.apache.org/

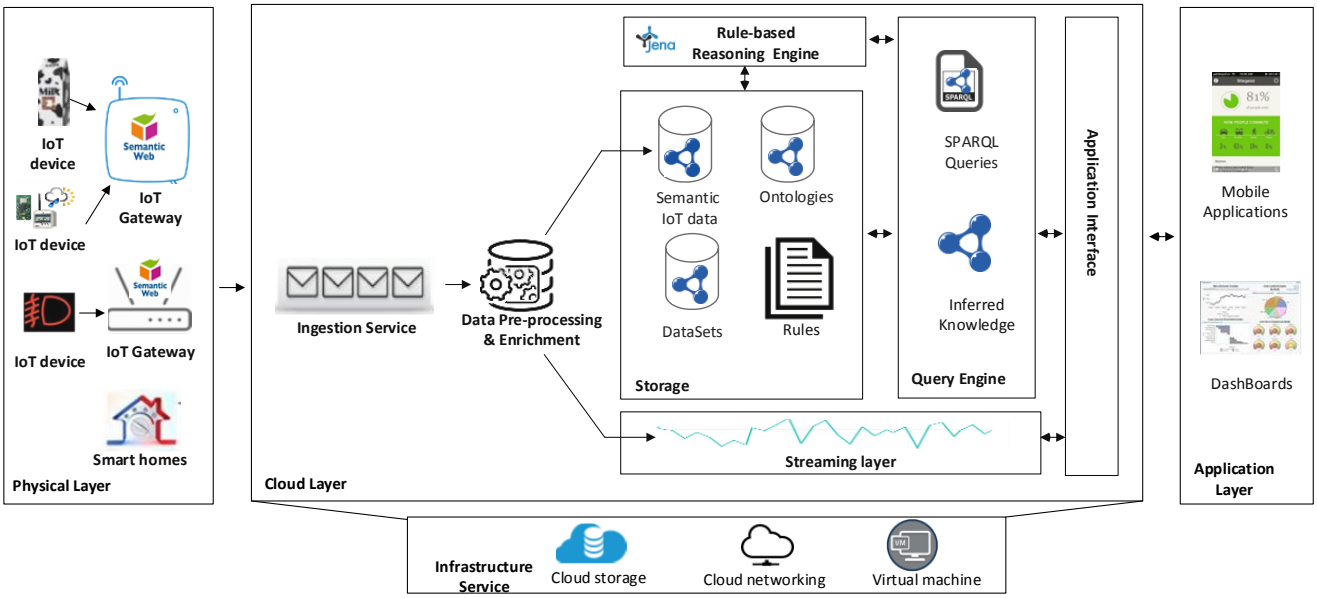

Fig. 1. Semantic-enabled Architecture for the Internet of Things

Hadoop[26], Cassandra[27], and so on. In such DBMS systems, data are at rest and when a user from application layer submits a query, the system returns results.

Reasoner. It is about deriving new knowledge and facts that do not exist in storage. A reasoning engine (i.e., reasoner) uses pre-defined rules stored in storage to make conclusions. Current reasoners can handle RDFS, OWL vocabularies, and RDF data formats. Some of common reasoning engines are Jena, Pellet, RacerPro, and Fact++. They use different rules languages to specify rules. Some reasoner support some of the popular rule languages such as SWRL[28] and RIF[29], whereas some have implemented their own rule syntaxes. Sensor-based Linked Open Rules (SLOR) [30], based on Jena rules syntax, is used for sharing and reusing rules for IoT applications.

Query engine. It executes queries from user and provides suggestions. The query engine can be implemented using ARQ[30], a SPARQL[31] processor of Jena. It loads ontologies, datasets stored in the storage, knowledge deduced from the reasoning engine, and executes SPARQL queries in order to provide suggestions.

Application interface. It interacts with users at application layers as well as cloud layer. It lets users to query cloud layer through APIs and presents data as results. The user queries through APIs are transformed into a SPARQL query and this query is submitted to query engine for results.

[26]http://hadoop.apache.org/
[27]http://cassandra.apache.org/
[28]https://www.w3.org/Submission/SWRL/
[29]https://www.w3.org/2001/sw/wiki/RIF
[30]https://jena.apache.org/documentation/query/
[31]https://www.w3.org/TR/rdf-sparql-query/

Infrastructure service. It provides required hardwares for computation, storage and networking, which are essentials to deploy and run above mentioned components. Amazon Web Services (AWS)[32] and Microsoft Azure[33] are some of the popular vendors that host cloud services.

C. The Application Layer

The Application layer offers support to build meaningful IoT applications on top of cloud layer. The cloud-layer exposes services through APIs. Application developers create applications on top of these services. To develop IoT applications, several approaches have been proposed in the closely related fields of Sensor Network, Pervasive Computing, Internet of Things, and Software Engineering in general. These approaches are summarized in Table I.

General-purpose Programming Languages (GPLs). Developers create IoT applications using programming languages (e.g., C, C++, JavaScript, Java) by targeting particular APIs provided by the cloud layer. The key advantage of such approach is that it allows the development of extremely efficient systems based on the complete control over code. However, this approach may not be easy to use for those (such as domain experts – doctors, civil engineers, etc.), who may have limited programming expertise.

Macro-programming. To address the limitations of the GPL approach, an alternative is macro-programming. It provides abstractions to specify high-level collaborative behaviors while hiding low-level details such as message passing or state maintenance from developers. Developers use high-level programming constructs (such as visual programming constructs

[32]https://aws.amazon.com/
[33]https://azure.microsoft.com

that can be dragged and dropped) around APIs provided by the cloud layer to develop various applications. However, one of the limitations of this approach is that platform-dependent design prevents its portability and re-usability across different platforms.

Model-driven development (MDD). To address development effort and platform-dependent design issues, MDD approach has been proposed. It applies the basic separation of concerns [31] principle both *vertically* and *horizontally*. *Vertical separation* principle reduces the application development complexity by separating the specification (Platform Independent Model–PIM) of the system functionality from its platform (Platform-Specific Model–PSM) such as programming languages and run-time systems. *Horizontal separation* principle reduces the development complexity by describing a system using different system views, where each view describes a certain facet of the system.

By following the MDD guidelines, many benefits can be achieved. For instance, by separating different concerns of a system at a certain level of abstractions and by providing transformation engines to convert these abstractions to target platforms, platform-independent design can be achieved, thereby improving productivity (*e.g.*, re-usability, extensibility) in the application development process.

III. Open questions and research challenges

This section summarizes the important (open) research challenges and issues found in designing a platform for building and deploying future Internet of Things applications. Like any emerging area, these challenges and design issues are not completely new, instead, they are evolved versions of many challenges and questions found in closely related fields, such as Service-oriented computing, Edge computing, Cloud Computing, Pervasive computing etc.

Maintaining Quality of Services across layers. As we have observed, the architecture (described in Figure 1) helps in tasks such as transferring data from authorized sensors to the interested destinations, where they are encrypted, compressed, filtered, processed, analyzed and/or stored. These subtasks are performed by different providers among different Cloud-centric IoT layers (Edge Device, Network, Cloud). IoT applications consumer has a certain level of expectations about the level of quality at which the application will perform. For example, consumers expect low latency and real-time analysis from systems built for health monitoring, environment monitoring or even listing trending topics in social media. In such systems, data freshness plays a significant role on resulted decisions where any delay might result in a late decision in monitoring systems or out of date trending topics that do not match the real-time data anymore. That means, even if the system functions correctly, its output might not match the main purpose of the system. Therefore, there is a need to ensure that systems/applications can function correctly with the required level of quality.

Security. Devices and software components in the cloud layer are installed in much protected environments and they can afford to implement very sophisticated security mechanisms. Meanwhile, devices and software in the physical layer and application layer need to operate very close to customers (e.g., a gateway device is collecting plants data in a smart factory). Moreover, these devices may not have resources and may not implement security mechanisms as cloud layer does, due to cost and performance issues. This makes the gateway devices more vulnerable to security attacks. Dividing the functionality among gateway devices and cloud layer while maintaining resource constraints and performance intact could be an interesting question to investigate.

Dynamism. As we have seen, the architecture (described in Figure 1) typically consists of edge devices, network components, gateways in the Physical layer and a number of components in the Cloud layer. One of the significant characteristics of devices in the physical layer is that they are evolving due to several reasons. For instance, devices suffer from software and hardware aging and software upgrades, which result in change of device properties. Runtime parameters (such as low-battery power) may degrade the data transmission rate. Similarly, in the Cloud layer, application performance may be affected by short-lived behaviors such as spikes in resource consumptions. These factors residing in both the cloud and physical layers can lead to a requirement of intelligent and automatic reconfiguration of assigned software and hardware resources to achieve objectives.

IV. Conclusion

In this paper, we have presented design requirements for a comprehensive IoT framework such as heterogeneity, interoperability, scalability, and actionability. We list typical shortcomings of recent proliferation of cloud-based approaches. To address these existing issues, we have presented an early design of a semantic-enabled framework that leverages concepts from Semantic Web, Internet of Things, and Big Data analytics, and leverages application development techniques such as macro-programming and model-driven development approaches. We do not only present necessary components to build a framework, but we also list existing technology alternatives with its merits.

Our early efforts to realized the proposed framework. We have been implementing the aforementioned framework. The early efforts towards realizing the proposed framework are mentioned below:

1) We have developed IoTSuite[34], A Toolkit for Prototyping Internet of Things Applications[35].It aims to reduce application development efforts in IoT. The current version of this project has been tested on several IoT technologies such as Android, Raspberry PI, Arduino, and JavaSE-enabled devices. It runs on different messaging protocols such as MQTT, CoAP, WebSocket, and applies

[34]Open source version is available at URL:https://github.com/pankeshlinux/IoTSuite/wiki

[35]Demo is available at URL: https://www.youtube.com/watch?v=nS_Je7IzPvM

Approach	Description	Examples	Benefits	Limitations
General Purpose Lang.	Developers use programming languages around APIs to program applications provided by cloud layer.	Node.js, C, Python	Developers can write efficient systems based on complete control over application code.	Higher development effort and it requires a significant programming expertise.
Macro-programming	They provide high-level (e.g., drag-and-drop) programming constructs around APIs to program applications provided by cloud layer	Node-RED, FRED[32] , MIT App Inventor [33]	Reduce development efforts compared to GPLs.	Platform-dependent design that prevents its portability and reusability across different platforms.
Model-driven Dev.	It separates different concerns of a system at a certain level of abstractions and provides transformation engines to convert them to target platforms.	DiaSuite[34] , IoTSuite[35]	Re-usable, Platform-independent, Extensible design.	Long development time to build a MDD system.

TABLE I
SUMMARY: EXISTING APPROACHES AVAILABLE FOR PROGRAMMING IOT APPLICATIONS.

server technologies such as Node.js. It covers database such as MySQL, MongoDB, and Microsoft Azure Cloud services.

2) We have been designing and developing a toolkit that assists developers to create interoperable cross-domain IoT applications. We have developing SWoTSuite: a Toolkit for Prototyping Cross-domain Semantic Web of Things Applications. An early version of this work is presented at ISWC 2016[36] and the demo video is available at URL[37]. We presented this toolkit in our tutorial on "Semantic Web meets Internet of Things and Web of Things [2nd Edition]" [38] and demonstrated with a smart city use case [4] at 26th International World Wide Web (WWW) Conference, 2017[39].

REFERENCES

[1] J.-P. Vasseur and A. Dunkels, *Interconnecting smart objects with IP: The next internet*. San Francisco, CA, USA: Morgan Kaufmann, 2010.

[2] P. Patel, A. Kattepur, D. Cassou, and G. Bouloukakis, "Evaluating the Ease of Application Development for the Internet of Things," Technical Report, Feb. 2013. [Online]. Available: https://hal.inria.fr/hal-00788366

[3] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787 – 2805, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128610001568

[4] P. Patel, A. Gyrard, S. K. Datta, and M. I. Ali, "Swotsuite: A toolkit for prototyping end-to-end semantic web of things applications," in *Proceedings of the 26th International Conference on World Wide Web Companion*, ser. WWW '17 Companion. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2017, pp. 263–267. [Online]. Available: https://doi.org/10.1145/3041021.3054736

[5] S. Chauhan, P. Patel, F. C. Delicato, and S. Chaudhary, "A development framework for programming cyber-physical systems," in *Proceedings of the 2Nd International Workshop on Software Engineering for Smart Cyber-Physical Systems*, ser. SEsCPS '16. New York, NY, USA: ACM, 2016, pp. 47–53. [Online]. Available: http://doi.acm.org/10.1145/2897035.2897039

[6] P. Patel, B. Morin, and S. Chaudhary, "A model-driven development framework for developing sense-compute-control applications," in *Proceedings of the 1st International Workshop on Modern Software Engineering Methods for Industrial Automation*, ser. MoSEMInA 2014. New York, NY, USA: ACM, 2014, pp. 52–61. [Online]. Available: http://doi.acm.org/10.1145/2593783.2593784

[36]https://arxiv.org/abs/1609.09014

[37]https://www.youtube.com/watch?v=BMP8CXtXzGo

[38]http://semantic-web-of-things.appspot.com/?p=WWW2017Tutorial

[39]http://www2017.com.au/

[7] V. Issarny, G. Bouloukakis, N. Georgantas, and B. Billet, *Revisiting Service-Oriented Architecture for the IoT: A Middleware Perspective*. Cham: Springer International Publishing, 2016, pp. 3–17. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-46295-0_1

[8] S. Chauhan, P. Patel, A. Sureka, F. C. Delicato, and S. Chaudhary, "Demonstration abstract: Iotsuite - a framework to design, implement, and deploy iot applications," in *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, April 2016, pp. 1–2.

[9] F. Fleurey, B. Morin, A. Solberg, and O. Barais, *MDE to Manage Communications with and between Resource-Constrained Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 349–363. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-24485-8_25

[10] M. Sheng, Y. Qin, B. Benatallah, and L. Yao, *Managing the Web of Things: Linking the Real World to the Web*. Elsevier Science & Technology Books, 2017. [Online]. Available: https://books.google.co.in/books?id=9NtbvgAACAAJ

[11] P. Patel, A. Gyrard, D. Thakker, A. P. Sheth, and M. Serrano, "Swotsuite: A development framework for prototyping cross-domain semantic web of things applications," *CoRR*, vol. abs/1609.09014, 2016. [Online]. Available: http://arxiv.org/abs/1609.09014

[12] A. Gyrard, "Designing cross-domain semantic Web of things applications," Theses, Télécom ParisTech, Apr. 2015. [Online]. Available: https://pastel.archives-ouvertes.fr/tel-01217561

[13] T. Berners-Lee, J. Hendler, O. Lassila *et al.*, "The semantic web," *Scientific american*, vol. 284, no. 5, pp. 28–37, 2001.

[14] G. Antoniou and F. Van Harmelen, *A semantic web primer*. MIT press, 2004.

[15] G. De Francisci Morales, A. Bifet, L. Khan, J. Gama, and W. Fan, "Iot big data stream mining," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: ACM, 2016, pp. 2119–2120. [Online]. Available: http://doi.acm.org/10.1145/2939672.2945385

[16] W. Fan and A. Bifet, "Mining big data: Current status, and forecast to the future," *SIGKDD Explor. Newsl.*, vol. 14, no. 2, pp. 1–5, Apr. 2013. [Online]. Available: http://doi.acm.org/10.1145/2481244.2481246

[17] A. Sheth, P. Anantharam, and K. Thirunarayan, "Applications of multimodal physical (iot), cyber and social data for reliable and actionable insights," in *10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*, Oct 2014, pp. 489–494.

[18] M. Chiang and T. Zhang, "Fog and iot: An overview of research opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, Dec 2016.

[19] W. Tai, J. Keeney, and D. O'Sullivan, "Resource-constrained reasoning using a reasoner composition approach," *Semantic Web*, vol. 6, no. 1, pp. 35–59, 2015.

[20] X. Su, J. Riekki, J. K. Nurminen, J. Nieminen, and M. Koskimies, "Adding semantics to internet of things," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 8, pp. 1844–1860, 2015.

[21] X. Su, J. Riekki, and J. Haverinen, "Entity notation: enabling knowledge representations for resource-constrained sensors," *Personal and Ubiquitous Computing*, vol. 16, no. 7, pp. 819–834, 2012. [Online]. Available: http://dx.doi.org/10.1007/s00779-011-0453-6

[22] X. Su, H. Zhang, J. Riekki, A. Kernen, J. K. Nurminen, and L. Du, "Connecting iot sensors to knowledge-based systems by transforming

senml to rdf," *Procedia Computer Science*, vol. 32, pp. 215 – 222, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050914006176

[23] J. Heuer, D. Peintner, S. Käbisch, J. Hund, and D. Anicic, "Web of things technologies for embedded applications."

[24] D. Le-Phuoc, J. X. Parreira, V. Reynolds, and M. Hauswirth, "Rdf on the go: An rdf storage and query processor for mobile devices," in *Proceedings of the 2010 International Conference on Posters & Demonstrations Track - Volume 658*, ser. ISWC-PD'10. Aachen, Germany, Germany: CEUR-WS.org, 2010, pp. 149–152. [Online]. Available: http://dl.acm.org/citation.cfm?id=2878399.2878437

[25] M. d'Aquin, A. Nikolov, and E. Motta, *How Much Semantic Data on Small Devices?* Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 565–575. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-16438-5_46

[26] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys (CSUR)*, vol. 35, no. 2, pp. 114–131, 2003.

[27] C. C. Aggarwal, *Data Streams: Models and Algorithms (Advances in Database Systems)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[28] S. García, S. Ramírez-Gallego, J. Luengo, J. M. Benítez, and F. Herrera, "Big data preprocessing: methods and prospects," *Big Data Analytics*, vol. 1, no. 1, p. 9, 2016.

[29] N. Sawant and H. Shah, *Big Data Application Architecture Q&A: A Problem - Solution Approach*, 1st ed. Berkely, CA, USA: Apress, 2013.

[30] A. Gyrard, M. Serranoy, J. Bosco Jaresy, S. K. Datta, and M. Intizar Ali, "Sensor-based linked open rules (S-LOR): An automated rule discovery approach for IoT applications and its use in smart cities," in *AW4city 2017, 3rd International ACM Smart City Workshop, In conjunction with WWW 2017, 26th World Wide Web International Conference, April 3, 2017, Perth, Australia*, Perth, AUSTRALIA, 04 2017. [Online]. Available: http://www.eurecom.fr/publication/5144

[31] V. Kulkarni and S. Reddy, "Separation of Concerns in Model-Driven Development," *IEEE Software*, vol. 20, no. 5, pp. 64–69, 2003. [Online]. Available: http://dx.doi.org/10.1109/ms.2003.1231154

[32] "Fred: a cloud based visual programming tool for the iot." [Online]. Available: http://sensetecnic.com/iot-platform/

[33] "Google app inventor." [Online]. Available: http://appinventor.mit.edu/explore/

[34] D. Cassou, J. Bruneau, C. Consel, and E. Balland, "Toward a tool-based development methodology for pervasive computing applications," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1445–1463, 2012.

[35] P. Patel and D. Cassou, "Enabling high-level application development for the internet of things," *Journal of Systems and Software*, vol. 103, pp. 62 – 84, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121215000187