Open Access

Open Journal of XXX (OJXX)
Volume X, Issue X, 20XX
http://www.ronpub.com/ojxx
ISSN XXXX-XXXX

Multi-Layer Cross Domain Reasoning over Distributed Autonomous IoT Applications

Muhammad Intizar Ali [A], Pankesh Patel [B]
Soumya Kanti Datta [C], Amelie Gyrard [D]

[A] Insight Centre for Data Analytics, National University of Ireland, Galway Ireland, ali.intizar@insight-centre.org,
[B] ABB Corporate Research, India, pankesh.patel@in.abb.com,
[C] Communication Systems Department, EURECOM, France, dattas@eurecom.fr
[D] Univ Lyon, MINES Saint-Etienne, CNRS, Laboratoire Hubert Curien, France, amelie.gyrard@emse.fr

Abstract

Due to the rapid advancements in the sensor technologies and IoT, we are witnessing a rapid growth in the use of sensors and relevant IoT applications. A very large number of sensors and IoT devices are in place in our surroundings which keep sensing dynamic contextual information. A true potential of the wide-spread of IoT devices can only be realized by designing and deploying a large number of smart IoT applications which can provide insights on the data collected from IoT devices and support decision making by converting raw sensor data into actionable knowledge. However, the process of getting value from sensor data streams and converting these raw sensor values into actionable knowledge requires extensive efforts from IoT application developers and domain experts.
In this paper, our main aim is to propose a multi-layer cross domain reasoning framework, which can support application developers, end-users and domain experts to automatically understand relevant events and extract actionable knowledge with minimal efforts. Our framework reduces the efforts required for IoT applications development (i) by supporting automated application code generation and access mechanisms using IoTSuite*, (ii)* by leveraging from Machine-to-Machine Measurement (M3) framework to exploit semantic technologies and domain knowledge*, and (iii)* by using automated sensor discovery and complex event processing of relevant events (ACEIS Middleware) *at the multiple data processing layers and different stages of the IoT application development life cycle. In the essence, our framework supports the end-users and IoT application developers to design innovative IoT applications by reducing the programming efforts, by identifying relevant events and by suggesting potential actions based on complex event processing and reasoning for cross-domain IoT applications.*

Type of Paper and Keywords

Regular research paper: *IoT, Cross-Domain Reasoning, Discovery, IoT Applications Design and Development.*

1 Introduction

The growing popularity of IoT and easy access to sensing technologies are leading to a great increase in the number of sensors available in our surrounding. These sensors produce a tremendous amount of data in a streaming fashion. However, a true value from this large amounts of data can only be realized by harnessing these large amounts of sensor streams and analyze these streams in real-time to detect relevant events [4]. A large number of IoT applications are being designed to process sensor data streams and provide various valuable services. IoT

technologies have a great potential of bringing a very positive impact on many aspects of our day-to-day lives [3, 28]. Nowadays, we can see different innovative IoT applications are designed in various areas such as agriculture and smart farming, health and fitness, smart home, smart cars and smart-x applications in smart cities. While an easy and cheap access to sensor technologies (e.g. raspberry pi) have made it possible for everyone to design and build their own innovative IoT applications, still it is not easy to learn underlying technologies required to build a complete IoT application. There are a few IoT toolkits available to support IoT application development but they are still in their infancy and usually, developers have to tackle many development and domain related issues before designing any IoT application.

With the growing popularity of IoT, we can easily foresee that in the near future there will be a massive deployment of IoT devices in various domains, bringing tremendous challenges and opportunities for scientific and economic activities. The biggest challenge for IoT applications is to bridge the gap between the physical and the cyber world [22, 27]. A few of the sample questions any IoT developer faces before designing an IoT applications are;

Q1: *What kind of data is produced by this sensor?*

Q2: *How can I access data from this sensor?*

Q3: *What kind of meaningful events be detected and extracted from sensor data?*

Q4: *Can I quickly design and build my own IoT application?*

Q5: *Can I combine multiple sensors data in a single IoT application?*

Q6: *How can I easily make multiple IoT silo applications inter-operable?*

In this paper, our aim is to support IoT application developer by reducing the efforts and expertise required to build any IoT application and facilitate developers to easily get answers to the above mentioned questions. We propose a framework for *Multi-layer Cross-domain Reasoning over Distributed Autonomous IoT Applications*, our framework reduces the application development effort. The framework is designed by combining 3 existing frameworks, namely (i) IoTSuite, (ii) M3 Framework, and (iii) ACEIS Middleware. The main functionalities of our main framework can be described as;

- **Understanding Sensor Data and Identifying Relevant Events (Q1 & Q3):** Our framework support developers and end-users to understand the structure of data produced by any sensor. It uses semantic technologies to identify relevant events from sensor data. A high level specification or domain area is provided to our framework, these specifications are used to identify relevant information model and ontologies already designed for the given domain. Reasoning and querying over the relevant ontologies help to identify relevant events which can be produced and monitored using the given sensor.

- **Providing automation at different phases of application development life-cycle (Q2):** Our toolkit provides a set of high-level modeling languages to specify each development concern and abstracts the heterogeneity related complexity. It integrates code generation, task-mapping, and linking techniques. Code generation supports the application development phase by producing a programming framework that allows stakeholders to focus on the application logic, while our mapping and linking techniques together support the deployment phase by producing device-specific code to result in a distributed system collaboratively hosted by individual devices.

- **Reducing the time spent for developing WoT application (Q4):** In order to create inter-operable and cross-domain SWoT applications, developers have to perform various tasks such as designing an application, semantically annotating data and interpreting data. To perform these tasks, developers have to learn semantic web technologies and tools, which is a time consuming process and can take a substantial amount of time. Reducing this gap as much as possible can be done by empowering a framework that assists developers in designing inter-operable applications with minimal knowledge of semantic web technologies.

- **Reducing the learning curve required by WoT developers to integrate semantic web technologies (Q5 & Q6):** Fast prototyping of semantic-based WoT applications by hiding the use of semantic web technologies as much as possible is required to avoid the developers' burden on designing ontologies, semantic annotators and reasoning mechanisms to enrich their data. An extensive work with Web frameworks (e.g. Drupal, Wordpress) has been done to design pre-defined templates to automatically generate websites to avoid users dealing with Web technologies. Based on this idea, pre-defined templates to design SWoT applications can be created.

Outline. The remainder of this paper is organized as follows: We emphasize the need of our framework by presenting a motivating scenario in Section 2. In Section 3 we present an overview of the existing frameworks for IoT application development. Section 4 presents our multi-layer cross domain reasoning framework and its underlying components. We evaluate the information flow and applicability of our framework in Section 5. We discuss state of the art technologies in Section 6 before concluding in Section 7.

2 MOTIVATING SCENARIO

Consider a real world scenario, where Alice an enthusiastic IoT developer wishes to design and build innovative IoT applications. She has access to basic IoT hardware such as raspberry pi and a few sensors such as temperature, humidity, and proximity sensor etc. Alice wishes to design her own innovative IoT applications performing basic smart home automation tasks. The home automation system should be capable of performing various daily tasks automatically such as controlling heating system, lights, and burglar alarm system. Starting to develop such application, Alice needs to understand the data produced by sensors, their access mechanisms, and relevant events. This can be easily done by using a combination of functionalities supported by M3 and IoTSuite, where M3 processes a single sensor data and provides a concrete list of relevant events for that particular sensor, while IoTSuite generates code to access data from the sensor. We call single sensor based reasoning and code generation support as the first layer of IoT application for a singular device.

Now consider that Alice can process high-level events by combining data from multiple sensors e.g. a combination of temperature and humidity sensor can detect events like the presence of fog. Again, a combination of M3 & IoTSuite can identify relevant multi-sensor level events and facilitate code generation to detect these events. We call multiple sensors based reasoning and code generation support as a second layer of IoT application for multiple sensing devices.

Using the singular sensor and multiple sensors based reasoning and developing support, Alice is able to build a complete home automation system. Additionally, Alice also owns a smart car, which is well equipped with modern sensing technologies and a smart car supporting software (developed using IoTSuite & M3) is also available for communication among various sensors within the car as well as with external sources of information. Both the home automation and smart car applications are performing their required tasks within the specified domain of each application.

Now, consider a cross-domain reasoning and events processing scenario, where both applications could leverage from data and information collected from each of these two applications as well external information sources. These combined rich sources of information can extend the functionality of existing applications by benefiting from the knowledge derived by another application in a completely different domain. For example, a smart home automation system at Alice home operates using a pre-planned schedule for home heating system after considering daily routine patterns of Alice arrival and departure times form the home to work. On a busy Monday evening on her way back to home from work, Alice is stuck into a severe traffic jam and her car automation system reports an expected delay of more than an hour than her usual arrival time at home. Our framework can support building applications that could potentially process the information from car automation system to deploy actuation over the home automation system for delaying the triggering of an automated heating system and thus conserving the energy consumption. ACEIS can support an automated discovery and integration of relevant IoT streams by querying over cross domain IoT applications.

We also envision another scenario related to autonomous vehicles and their intersection with the IoT. Consider, Alice is traveling on such a vehicle. The car must be able to detect environmental situations (e.g. fog, heavy precipitation) and react to them automatically. For this purpose, the vehicular sensors (e.g. location, speed) can obtain data from other platforms like local environment sensors (e.g. humidity, precipitation) and combine them to determine if there is fog. Our cross-domain reasoning framework can support building intelligent applications which can process data form multiple silo IoT applications by (i) converting the heterogeneous sensor data formats into a uniform format (e.g., RDF) and (ii) providing a uniform and interoperable mechanism for cross-domain reasoning and computation. Various actuation suggestions can be generated by our framework e.g., in the case of dense fog in the environment, the vehicle driver can receive suggestions to turn on fog lamps and reduce the speed to a certain level.

3 EXISTING FRAMEWORKS

Our framework for multi-layer cross domain reasoning is a combination of three frameworks, to provide a necessary background for readers, in this section, we summarize the following frameworks.

- IoTSuite [11] creates a necessary infrastructure that enables IoT applications[1]. It takes high-level

[1] https://github.com/pankeshlinux/IoTSuite

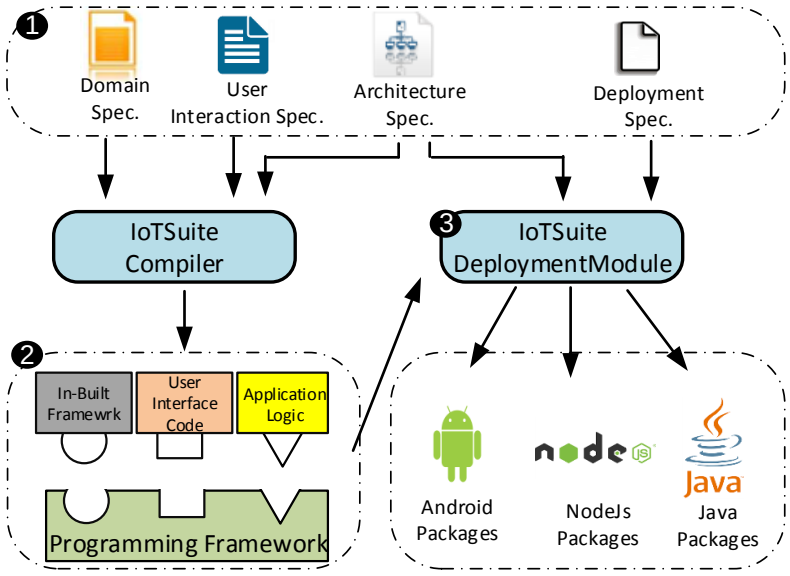


Figure 1: IoTSuite – Application Development Framework

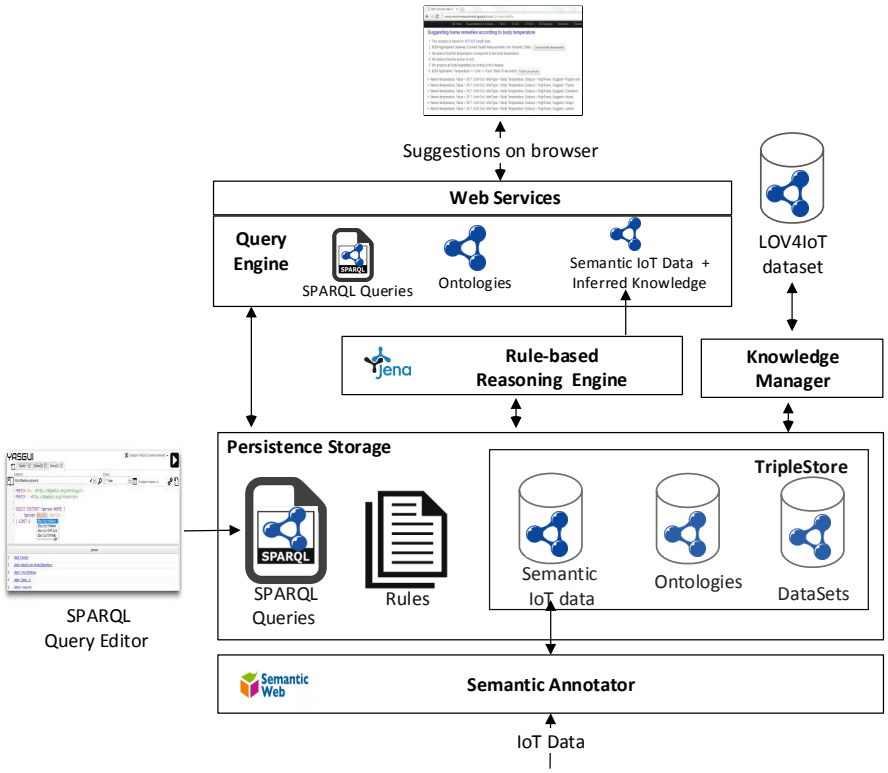


Figure 2: M3 Framework architecture

specifications as input, parses them, and generates device-specific code to result in a distributed software system collaboratively hosted by individual IoT devices. It is described in Section 3.1.

- The Machine-to-Machine Measurements (M3) framework[2] [21] takes semantic data as inputs, reasons over them by leveraging semantic web technologies and provide suggestions to users. It is described in Section 3.2.
- ACEIS Middleware[3] can process IoT application request to automatically discover and integrate relevant IoT streams to address application requests. It can also perform complex event processing over streams and their events. ACEIS is discussed in detail in Section 3.3.

3.1 IoTSuite

A generic framework of IoTSuite is depicted in Figure 1, in what follows we present necessary steps to develop IoT application using IoTSuite:

Specifying high-level specification. This step involves the writing of high-level specifications (Step ① in Figure 1). This step involves the writing of four specifications: (1) Domain specification: It includes the writing of domain-specific concepts such as sensors (it observes entities of interest), actuators (it affects the environment), and storage (it stores information about entities of interest). (2) Architecture specification: It includes the writing specification of computational components and interactions with other components. Computational services are fueled by sensors and storage. They process inputs data and take appropriate decisions by triggering actuators. (3) User interaction specification: It includes data exchange between an application and a user. (4) Deployment specification: It describes a device and its properties of a target deployment.

Compiling high-level specification. This step generates a framework (Step ② in Figure 1) in a general-purpose programming language. The framework contains abstract classes, corresponding to each concept defined in high-level specifications. The abstract classes contain concrete and abstract methods as well as interfaces. The concrete methods are used to hide interactions with other software components. The abstract methods are implemented to write application-specific logic (an example of application logic could be – open a window when an average temperature value is greater than a certain threshold). The generated interfaces implement user interfaces that connect UI elements to concrete methods of the generated framework.

Generating deployment packages. It consists of two steps (Step ③ in Figure 1). The first step is to map a set of computational components (specified in an architecture specification) to a set of devices (specified in a deployment specification). The second step combines the mapping outputs and the generated code of Step ②, and generates device-specific packages as final outputs that result into a distributed software system collaboratively hosted by individual devices.

3.2 M3 framework

Figure 2 represents M3 framework, M3 contains the following sub-components;

Semantic annotator. It transforms varying formats to the standardized RDF format. A common RDF format enables reasoning over sensor data in a unified way. It annotates sensor data according to the M3 taxonomy [21, p. 93], which is an extension of W3C Semantic Sensor Network (SSN).

Storage. It stores M3 ontologies, datasets and rules as well as annotated RDF sensor data in a triple store [21]. Moreover, M3 compatible SPARQL queries are stored as flat files to assist developers.

Knowledge manager. It updates the storage with a domain-specific knowledge that is further used in a reasoning process. M3 uses Linked Open Vocabularies for the Internet of Things (LOV4IoT)[4]. The LOVIoT provides domain ontologies, datasets, and rules that could be reused to design cross-domain IoT applications.

Reasoning engine. It infers high-level knowledge using Jena inference engine and M3 rules. The M3 rules are extracted from LOV4IoT and they are re-designed in compliance with M3 taxonomy [23].

Query engine. It executes SPARQL queries and provides suggestions to users. The query engine executes SPARQL queries overloaded M3 ontologies, datasets, and knowledge deduced from the reasoning engine in order to provide suggestions to users. M3 implements the query engine using ARQ[5], a SPARQL process for Jena.

3.3 ACEIS middleware

The ACEIS core module serves as a middleware between low-level IoT data streams and upper-level Smart City

[2] https://github.com/pankeshlinux/SWoTSuite

[3] https://github.com/CityPulse/Stream-Discovery-and-Integration-Middleware

[4] http://sensormeasurement.appspot.com/?p=ontologies

[5] https://jena.apache.org/documentation/query/

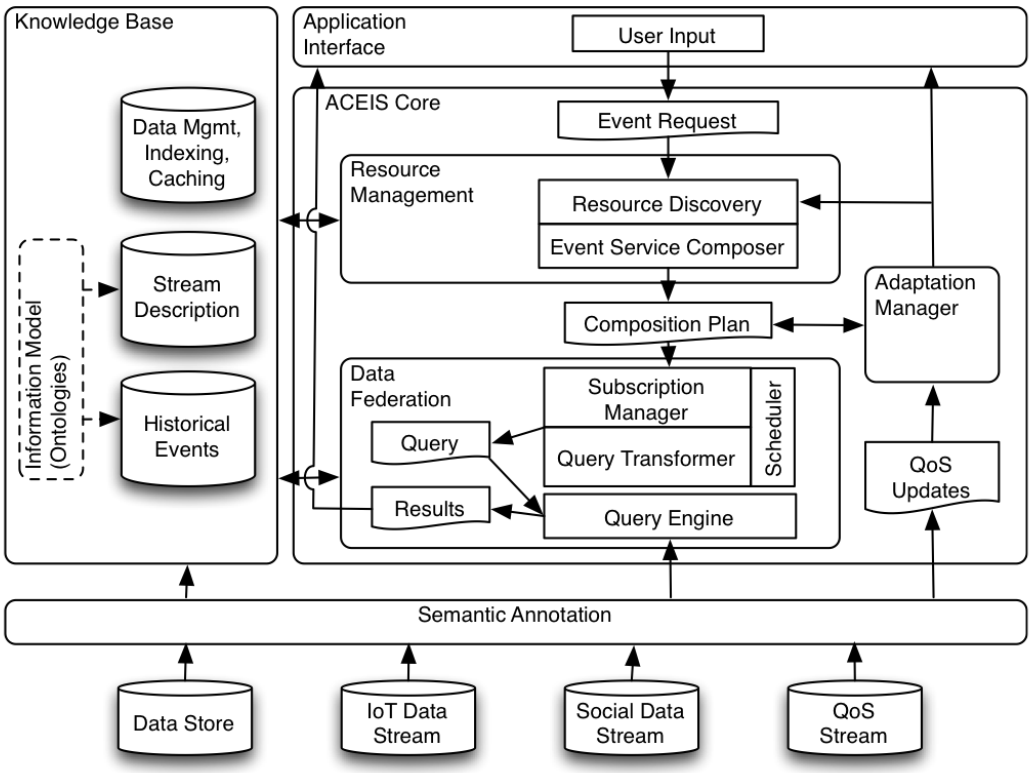


Figure 3: ACEIS Architecture

applications [17]. ACEIS core is capable of discovering, composing, consuming and publishing complex event processing capabilities as reusable services [18]. We call these services (primitive or complex) event services. ACEIS architecture is depicted in Figure 3. ACEIS core consists of two major components: resource management and data federation & complex event processing. In the following, we introduce their functionalities and interactions.

Resource Manager. The resource management component is responsible for discovering and composing event services based on static service descriptions. It receives event requests generated by the application interface containing users functional/non-functional requirements and preferences, and creates composition plans for event requests, specifying which event services are needed to address the requirements in event requests and how they should be composed. Resource management component contains two sub-components: resource discovery component and event service composer. The resource discovery component uses conventional semantic service discovery technique to retrieve IoT services delivering primitive events. It deals with the primitive event requests specified within event requests. The event service composer creates service composition plans to detect the complex events specified by event requests based on event patterns. We refer readers to [19] for further details of the composition algorithm used by the event service composer.

Data Federation & Complex Event Processing. The data federation component is responsible for implementing the composition plan over event service networks and process complex event logics using heterogeneous data sources. The composition plan is firstly used by the subscription manager which will make subscriptions to the event services involved in composition plan. Later, the query transformer transforms the semantically annotated composition plan into a set of stream reasoning queries to be executed on a stream query engine. The query transformer produces two kinds of stream queries: regular event queries that detect the complex events specified by event requests and constraint validation queries that monitor the constraints specified in event requests. Thus the query engine produces two kinds of results: (i) event query results are forwarded to the application interface and (ii) constraint violations are detected by constraint validation queries and sent to the adaptation manager. Adaptation manager decides whether an automatic adaptation is possible. If so, it creates and deploys a new composition plan that conforms with the constraints to

replace the existing one. It dispatches a notification to the application interface.

4 Multi-Layer Cross Domain Reasoning over Distributed Autonomous IoT Applications

We propose a generic framework for cross-domain reasoning over multiple IoT applications by combining the strengths of three different frameworks designed to serve specific tasks in their respective domains. Our proposed framework is capable to extensively support IoT application developers to understand data produced by individual sensors, combine multiple sensors to get meaningful events, and perform cross-domain reasoning over multiple autonomous distributed application for knowledge extraction and use this knowledge for actuation across multiple distributed and autonomous applications.

4.1 System Architecture

Figure 4 presents an overall architecture of the framework. Our cross domain reasoning framework facilitates an easy access of data from sensors by automatically generating and deploying applications for IoT devices using IoTSuite. These applications hide the technicalities of accessing physical plane (e.g., sensor hardware specifications), communication plane (e.g., network protocols) and data access plane (e.g., data wrappers, APIs).

The M3 framework supports a multi-layer reasoning facility, at the single sensor based reasoning level it processes individual sensor data (e.g., temperature sensor data) and suggests potential events (e.g., cold, warm etc.). At multiple sensors based reasoning level, it processes multiple sensor data in combination (e.g., temperature, humidity etc.) to suggest relevant events (e.g., fog, rain, and snow etc.). At the cross-domain application level reasoning, M3 supports processing of events collected from autonomous applications by extracting additional knowledge to deduce additional events. Additional extracted knowledge and deduced events are processed by complex event processing engine in ACEIS middleware to produce actionable knowledge. Once the actionable knowledge is extracted IoTSuite can trigger actuation for individual sensors or for the complete IoT application.

In the following, we present elements of the architecture (Figure 4) and we describe the functionality of each element realized using our frameworks: IoTSuite, M3, and ACEIS middleware.

4.2 System Components for Data Processing & Reasoning Layers

In this section, we discussed different data processing layers, reasoning layers at singular sensor, multiple sensors and applications' levels. Below, we briefly introduce different components used at these layers.

4.2.1 Device plane

This layer consists of a variety of devices ranging from resource constrained devices (e.g., microcontroller) to powerful devices (e.g., smartphone, desktop computer). An IoT application may execute on a network consisting of different types of devices. For example, a smart home application consists of devices, including sensing devices (e.g., temperature sensor), actuating devices (e.g., heater), user interface devices (e.g., smart phone, monitor), storage devices (e.g., profile storage on different database systems such as MySQL or MongoDB). Moreover, each device may exhibit heterogeneous platforms. For instance, a device could be running Android mobile OS, Raspbian on Raspberry PI, a desktop computer with OS such as GNU/Linux, Windows, or microcontroller with no OS.

To address the above mentioned heterogeneity challenges at the device plane, IoTSuite lets developers write a domain specification. In a domain specification, concepts such as sensors, actuators, storage are specified in a high-level manner to abstract low-level platform specific details from developers. For instance, while writing a temperature sensor code in a domain specification, developers do not have to think about a temperature sensor is hosted on which a platform (e.g., Android OS, Raspbian OS). This complexity is taken care by IoTSuite.

4.2.2 Communication plane

This layer is responsible for communicating data from devices at the device plane to the outside world. The communication with devices is realized using different communication protocols. Each protocol exhibits its own interaction patterns. Some of the common interaction modes with devices are request/response, publish/subscribe, stream, and command. An IoT application executes on a network consisting of heterogeneous devices, each may have different interaction mode and implement a different protocol. This heterogeneity at the communication plane largely spreads into the application code and makes the portability of code difficult.

To address these challenges, IoTSuite provides high-level abstractions that abstract heterogeneous interactions among devices. So, a developer does not have to handle these low-level details. Moreover, to integrate

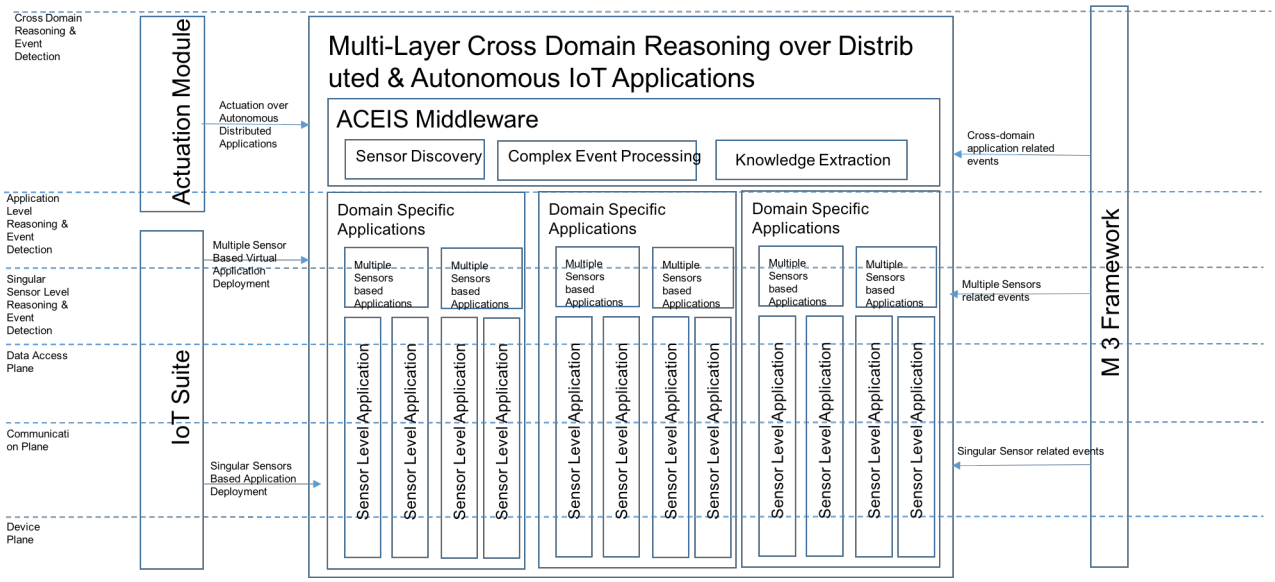


Figure 4: An overview of the cross-domain reasoning framework

a communication protocol, IoTSuite implements well-defined interfaces [31]. The implementation of these interfaces with a protocol library which integrates the support of a communication protocol into a system.

4.2.3 Data Access plane

This layer is responsible for accessing devices using different interaction modes and protocols available at the communication plane. The accessed data is generally in a raw format and may not provide any explicit information. In order to address data interoperability problems among heterogeneous devices, this layer annotates raw sensor data and transforms it into a format that can be used further at the layer above. A popular data representation format, such as Resource Description Framework (RDF), can be used as a data exchange format for IoT devices. However, sometimes devices may not be able to annotate data with RDF because of its resource constraints such as memory, processing & transmission power, and bandwidth. An alternate solution is to use a lightweight format such as Sensor Markup Language (SenML)[6] and transform it to RDF format at the application level.

To perform the above mentioned functionality, M3 framework interacts with devices using various interaction modes and protocols. It gets sensor metadata (such as measurement type, sensor type, and value) from devices and converts sensor metadata in a unified description using semantic web technologies such as RDF/XML. Sensor metadata is semantically annotated using M3 ontology. This is an essential step to provide a basis for reasoning, described in the next section.

4.2.4 Singular Sensor Reasoning & Event Detection

This layer takes RDF data as inputs and processes them further to derive new knowledge and facts. It is essential to push a part of reasoning over RDF data on a device because it has several advantages such as: (i) scalability can be achieved at a device because it distributes the computation, (ii) data transmission cost from a device to a centralized node could be reduced because a device has to send refined results rather than raw data, and (iii) the local data processing contributes to privacy as only pre-defined processed data is sent.

The above mentioned functionality is implemented in M3. It integrates Jena inference engine and M3 rules. The M3 rules are extracted from LOV4IoT and they are re-designed in compliance with M3 taxonomy. Moreover, M3 ports a lightweight version of the reasoning engine to enable the reasoning process on Android devices [15].

4.2.5 Application Level Reasoning & Event Detection

At the application level reasoning and event detection, we used multiple sensors within an application, a combination of two independent sensor data can be used to detect additional events which are not possible to detect from two individual sensors. For example, a

[6] https://tools.ietf.org/html/draft-jennings-senml-10

combination of temperature and humidity sensors can suggest a possibility of detecting fog.

4.2.6 Cross Domain Reasoning & Event Detection

IoT applications are currently designed while keeping a single application domain in view, most of these applications target a domain specific problem. We used cross domain reasoning techniques to monitor and process events from totally independent applications and a cross domain reasoning over a combination of data from these two separate applications supports additional knowledge extraction and inference which was not possible from data generated by a single application. As described earlier in the motivating scenario, a cross domain reasoning allows to deduce additional events from silo IoT applications and can be turned into useful actuation for different applications, e.g., a delay in the traffic and estimated travel time can be used to delay the predefined schedule of heating system by the number minutes of delay reporting by traffic navigation system or a fog forecast from weather stations can be used by smart car to make decisions and suggestions such as turning on fog lamps and reducing the car speed based on the density of fog.

4.2.7 Actuation Module

It consumes the extracted knowledge from the cross-domain reasoning layer and uses this information to take appropriate actions of an application. An automated actuation from the cross-domain reasoning layer could be: (1) triggering an actuator (e.g., switch off a heater by an application), (2) event-based interaction with users such as notifying users (e.g., inform users whenever a dangerous situation such as fire around), or (3) periodic updates to users (e.g., displaying average energy consumption of a house on users dashboard or mobile device periodically). To define user interactions, IoTSuite provides a set of abstract interactors, similar to work [6], which denotes information exchange between an application and a cross domain reasoning layer. A developer specifies abstract interactors in a user interaction specification [10]. The compilation of this specification generates code that can be deployed on mobile devices or display devices, which let users interact with applications.

4.2.8 Sensor Discovery & Complex Event Processing

To lower the barrier between on-demand application request by end-users and IoT infrastructure, this module enables dynamic service discovery and automatic service composition over IoT infrastructure. We use ACEIS middle-ware to map the end-users or application request to discover relevant IoT resources and sensors. ACEIS is capable of processing application requests on the fly and can trigger its universal discovery module to search relevant sensors that can answer application request. ACEIS considers every sensor data stream as a service, hence the granularity level of the discovery module is at the sensor level rather than the application level. This feature allows cross-domain application requests to process individual sensors data from multiple applications.

Once the relevant data stream are identified ACEIS can compose sensor data streams by using complex event processing techniques. A composition plan generated by ACEIS ensures that a combination of individual sensor data streams provides answers to the application request specified by the end-user.

5 System Information Flow

In this section, we evaluate the feasibility of our system by showing information flow between various components of the framework. We provide various code snippets and examples to showcase the overall procedure which can be followed by IoT applications' developers while designing any application. In what follows, we show different examples showing support provided by our framework at the different levels of the cross-domain reasoning frameworks.

5.1 Reasoning and Application Development Support for Singular Sensor Based Applications

Listing 1 provides a sample set of events for temperature sensors. M3 gets user input about a sensor type (e.g. temperature sensor) and its specific domain (weather forecast). M3 uses its internal repository of ontologies and provides a set of events with their specific thresholds. These events are helpful for the developers to monitor and detect relevant events for that sensor.

```
[Cold:
        (?measurement rdf:type m3:Temperature)
        (?measurement m3:hasValue ?v)
        greaterThan(?v,0)
        lessThan(?v,10)
         ->
        (?measurement m3:isRelatedTo weather-dataset:
            Cold)
]

[SunnyTemperature:
        (?measurement rdf:type m3:WeatherTemperature)
        (?measurement m3:hasValue ?v)
        greaterThan(?v,25)
    ->
```

```
        (?measurement m3:isRelatedTo weather-dataset:Sunny)
]

[WinterTemperature:
        (?measurement rdf:type m3:WeatherTemperature)
        (?measurement m3:hasValue ?v)
        lessThan(?v,10)
        ->
        (?measurement m3:isRelatedTo naturopathy-dataset:
            Winter)
]
```

Listing 1: Rule interpreting temperature measurements

Listing 2 shows a sample excerpt of the code generated by IoTSuite, which is automatically generated and creates templates for developers to specify their desired values e.g. monitoring interval, events threshold etc.

```
structs:
  TempStruct
    tempValue: double;
    unitOfMeasurement : String;
  SmokeStruct
    smokeValue:double;
    unitOfMeasurement:String;
resources:
  sensors:
    periodicSensors:
      TemperatureSensor
        generate tempMeasurement:TempStruct;
        sample period 1000 for 6000000;
    eventDrivenSensors:
      SmokeDetector
        generate smokeMeasurement:SmokeStruct;
        onCondition smokeValue > 650 PPM ;
  actuators:
    Alarm
      action On();
```

Listing 2: IoTSuite Sensor code example.

5.2 Reasoning and Application Development Support for Multi Sensor Based Applications

Listing 3 shows an example with two different kinds of measurement (temperature and precipitation) to deduce more complicated events (e.g., Snowy). The rule also takes into account the context, indeed a temperature measurement can be generated in different locations (outside, inside, body thermometer). The rule explicitly explains that the rule applies only for weather temperature measurements through the triple `?measurement rdf:type m3:WeatherTemperature`.

```

@prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns@prefix
    m3: http://sensormeasurement.appspot.com/m3
@prefix weather-dataset: http://sensormeasurement.appspot.com/
    weather-dataset/

[Snowy:
    (?measurement rdf:type m3:WeatherTemperature)
    (?measurement m3:hasValue ?value)
    le(?value,0)

    (?measurement2 rdf:type m3:Precipitation)
    (?measurement2 m3:hasValue ?value2)
    greaterThan(?value2,0)
  ->
  (?measurement m3:isRelatedTo weather-dataset:Snowy)
  (?measurement2 m3:isRelatedTo weather-dataset:Snowy)
]
]
```

Listing 3: Rule requiring measurement from two sensors

IoTSuite also creates templates containing sample code to design a dashboard for IoT applications. Listing 4 contains a sample code to generate a visualization module to design a dashboard for temperature and humidity sensors.

```
structs:
  VisualizationStruct
    tempValue:double;
    humidityValue:double;
resources:
  userInteractions:
    DashBoard
      notify DisplaySensorMeasurement(
          sensorMeasurement :
          VisualizationStruct);
```

Listing 4: IoTSuite Dashboard code.

5.3 Reasoning and Application Development Support for Cross-Domain Applications

Listing 5 shows a logical rule example implemented as a Jena Rule language. The rule checks the type of the measurement (eg., precipitation) according to the dictionary compliant with the M3 framework. The value is compared to specific values to be able to deduce higher level, in this example `greaterThan(?value,20)`. The main novelty of such rules is to follow Linked Data principles. The property `m3:isRelatedTo` enables the linking with external domain knowledge (e.g., weather dataset).

```

@prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns@prefix
    m3: http://sensormeasurement.appspot.com/m3
@prefix weather-dataset: http://sensormeasurement.appspot.com/
    weather-dataset/

[HeavyRain:
        (?measurement rdf:type m3:Precipitation)
        (?measurement m3:hasValue ?value)
        (?measurement m3:hasUnit ?unit)
        greaterThan(?value,20)
        lessThan(?value,50)
    ->
    (?measurement m3:isRelatedTo weather-dataset:HeavyRain
        )
```

```
]
```

Listing 5: Rule to interpret precipitation measurement

Listing 6 shows that a transport dataset is using the weather dataset through the property `m3:hasRecommendation`. This example demonstrates that from a precipitation measurement, new knowledge can be inferred and linked to cross-domain knowledge: weather and transport. These rule-based mechanisms can be integrated into a smart car, with a precipitation sensor deployed on the car to automatically switch on wipers for instance.

```
<transport:SafetyDevice rdf:about="Wiper">
    <rdfs:label xml:lang="en">Wipers</rdfs:label>
<m3:hasRecommendation rdf:resource="&weather-dataset;Rainy"/>
<m3:hasRecommendation rdf:resource="&weather-dataset;LightRain"/>
<m3:hasRecommendation rdf:resource="&weather-dataset;HeavyRain"/>
</transport:SafetyDevice>
```

Listing 6: RDF instance to link cross-domain datasets (weather and transport)

5.4 Cross-Domain Reasoning in Mobile Applications

Cross-domain semantic reasoning is possible in high-end smartphones and tablets due to the availability of resources. Also, at present, mobile devices have become the de-facto human-to-IoT interfacing system. To integrate the building blocks of the M3 Framework in a mobile application, its life-cycle is categorized into following steps.

Discovery and Provisioning. The first step triggers a discovery method which searches for available sensors and domains. We assume that the sensors are described using IETF standard protocols like CoRE Link Format[7] and each the descriptions are registered into a central registry database. During the first step, the Android application connects to the registry and queries the database. It returns one/more URI(s) corresponding to the sensor(s). A detailed mechanism of the discovery procedure is mentioned in [14]. This is followed by provisioning where a sensor or a combination of sensors and domain(s) of operation (corresponding to the overall application scenario) are communicated to the M3 cloud.

Provisioning M3 Templates. Following the discovery and provisioning, the M3 cloud returns a set of cross-domain templates to the Android application. Based on the application logic, one of the templates should be chosen. This step can also be configured in the background to run as an Android service.

Semantic Reasoning and Suggestions. Once a template is chosen, the M3 cloud internally generates a template containing rules, ontologies, datasets, domain knowledge necessary for cross-domain reasoning. The template is downloaded into the Android application. The sensor(s) are queries directly to get sensor metadata. Following the semantic reasoning, SPARQL queries are executed to generate suggestions. These suggestions are presented to the end-users who can select an action. It triggers an actuation module from the application. Figure 5 shows this mechanism embedded in Android powered devices. In the same way that the wiper suggestion has been proposed previously, in the case, fog lamp suggestion is provided.

Figure 5: Semantic reasoning and suggestions in the Android application

The advantage behind the above three steps is that the process is generic enough to accommodate with a range of cross-domain M3 templates. This allows the same application to be reused in multiple application logics, which in turn reduces the time for development and shortens time-to-market.

6 Related Work

To address the research challenges and questions discussed in Section 1, a commonly accepted technique is to design a framework or middleware. Initial approaches for IoT platforms design focused on data collections and sensors interoperability, while later on the focus shifted towards the provision of value added services. In this article, we mainly focus on semantics based solutions proposed for IoT applications.

[7] https://tools.ietf.org/pdf/rfc6690.pdf

6.1 Semantics based IoT Middlewares, Frameworks and Toolkits

We classify the existing frameworks/middleware into three broad categories: (1) to hide IoT system complexity, middleware (such as ACEIS) exposes services (such as service discovery, service composition) through various APIs and let developers to use their APIs to develop an application, (2) to address interoperability issues in IoT, frameworks (such as M3) leverage semantic web technologies and provide intelligent suggestions in cross-domain to users, and (3) to reduce IoT application development complexity, toolkits (such as IoTSuite) provides high-level abstractions to develop an IoT application.

In the following, we present a state of the art of each category briefly:

Semantic Based IoT Middlewares. Different middlewares for IoT data collection have been proposed over the past [2, 16, 32]. OpenIoT is designed to reduce the heterogeneity issues and provide a universal platform for IoT data collection and acquisition [2]. The Global Sensor Network (GSN) middleware facilitates flexible discovery and integration of physical sensors [1], while X-GSN is its extension to support virtual as well as physical sensors [8]. In the context of smart cities, various semantic based IoT data platforms are proposed [28]. ACEIS middleware is part of large-scale realtime data analytics platform of CityPulse Framework and contains various on-the-fly sensor discovery and composition tools for real-time data analytics over IoT streams.

Semantic-Enabled Framework. Similar to the M3 framework, Chen et al. have discussed intelligent processing for IoT data related to domain specific-applications [12]. The need for cross-domain applications with semantic interoperability and data management in IoT applications are described in [25]. They clearly explain a lack of standardization related to ontologies and data formats but do not provide any solutions. The authors of [29] have presented know-how on semantically annotating IoT sensor data and the need of domain ontologies. This is further explained in [26] where the authors employ domain-specific ontologies and ontologies matching and alignment tools to build IoT applications. However, they do not explicitly describe the issues encountered if developers want to combine the domain ontologies.

IoT Toolkits. To reduce IoT application development effort, a macro-programming is a popular approach. Developers use high-level programming constructs (such as visual programming constructs that can be dragged and dropped) around APIs provided by a middleware to develop various applications. For instance, Node-RED is a programming tool for wiring together IoT devices, APIs, and online services. However, one of the limitations of this approach is that platform-dependent design prevents its portability and re-usability across different platforms.

To address development effort and platform-dependent design issues, Model-driven Development (MDD) approach has been proposed. It applies the basic separation of concerns [24] 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. MDD tools such as IoTSuite, DiaSuite [9] adopts MDD approaches.

6.2 Delineation from similar IoT Frameworks

In this section, we briefly discuss two of the most well-known frameworks for IoT application design e.g. FIWARE and SOFIA2 Platform. We then discuss novelties presented in our framework and how our proposed framework can support application developers to easily build semantics based IoT applications from cross-domain reasoning and also compliments individual components designed and developed for the above mentioned existing frameworks.

FIWARE. FIWARE[8] aims at providing an open cloud based system to design, develop and deliver cost-effective Future Internet (FI) applications and services including the IoT. The main ingredients of the architecture are Generic Enablers (GE) [13], [5] which allows IoT service enablement. Through the GEs, physical devices (e.g. sensors, actuators) become available, search-able, accessible and usable by high level IoT applications and services that belong to various vertical markets. The GEs relevant to IoT are - (i) IoT Service Enablement (further categorized into IoT Backend and IoT Edge) which is hosted in a Cloud Data center [30], (ii) Data/Context Management which enables development of smart, personalized and context aware IoT applications, (iii) Applications, Services and Data Delivery which is responsible for creating an IoT ecosystem of applications, services and business intelligence, (iv) Security which is centered around identity [7] & access management and access control policies.

SOFIA2 Platform. This platform[9] [20] provides several functionalities to the IoT ecosystems–(i) implementing

[8] http://www.fi-ware.org/

[9] http://sofia2.com

standard data formats (e.g., JSON), protocols (e.g., MQTT), RESTful web services, (ii) Independence of IoT device firmware (e.g., Android, Linux, iOS), (iii) Support for Java based application toolkits, (iv) scalability and extendibility. The SOFIA2 platform performs well for vertical IoT markets.

6.3 Progress Beyond State-of-the-art

While both FIWARE and SOFIA2 are considered successful platforms for IoT applications development support, we consider that our proposed approach for multi-layer and cross domain reasoning progresses beyond state-of-the-art for two main reasons, (i) provision of cross-domain reasoning at the horizontal layer among multiple IoT applications and (ii) a comprehensive support for application developer to automatically build cross domain IoT applications. The above mentioned current platforms mainly focused on vertical applications design while we promote horizontal reasoning over distributed autonomous IoT applications. Moreover, the interoperability among the mentioned platforms among is not tested while our work successfully integrated three separate frameworks into one. It is worth mentioning that contrary to the single application design as advocated in FIWARE and SOFIA, we emphasize on cloud-based applications, where two totally independent applications can benefit from cross-domain reasoning and get fruitful insights from data analytics which were not possible from an application designed to build a single domain dependent applications. Additional to interoperability and cross domain reasoning, we also believe that our work on facilitating application developers by providing useful templates and reduce the learning curve of application development is also a considerable advancement beyond state of the art. We expect that our framework will gain the attention of a wider community and have good potential for noticeable impact on IoT research and development community.

7 Conclusion & Future Work

In this paper, we propose a multi-layer cross-domain reasoning framework, which combines various features of three separate frameworks in order to support a complete application design and development framework for cross-domain IoT applications. Our main aim in this paper was to showcase the feasibility of this approach and emphasize the usability of our framework to assist and expedite the IoT application development process.

In future, we intend to extensively evaluate the performance of our framework and attract a large community of application developers to get benefit from application development support provided by our framework.

References

[1] K. Aberer, M. Hauswirth, and A. Salehi, "A middleware for fast and flexible sensor network deployment," in *Proc. of VLDB2006*. VLDB Endowment, 2006, pp. 1199–1202.

[2] N. K. AIT, J. E. B. AL, and A. G. AL, "D2.3 openiot detailed architecture and proof-of-concept specifications," 2013.

[3] M. I. Ali, N. Ono, M. Kaysar, K. Griffin, and A. Mileo, "A semantic processing framework for iot-enabled communication systems," in *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II*, 2015, pp. 241–258. [Online]. Available: https://doi.org/10.1007/978-3-319-25010-6_14

[4] M. I. Ali, N. Ono, M. Kaysar, Z. U. Shamszaman, T.-L. Pham, F. Gao, K. Griffin, and A. Mileo, "Real-time data analytics and event detection for iot-enabled communication systems," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 42, pp. 19–37, 2017.

[5] P. Andriani, L. Briguglio, L. Lombardo, M. Nigrelli, D. Pellegrino, J. S. Torres, and A. Voulkidis, "Fiware generic enablers as building blocks of a marketplace for energy," in *eChallenges e-2015 Conference*, Nov 2015, pp. 1–10.

[6] E. Balland, C. Consel, B. N;Kaoua, and H. Sauzéon, "A case for human-driven software development," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486970

[7] L. Barreto, A. Celesti, M. Villari, M. Fazio, and A. Puliafito, "Identity management in iot clouds: A fiware case of study," in *2015 IEEE Conference on Communications and Network Security (CNS)*, Sept 2015, pp. 680–684.

[8] J.-P. Calbimonte, S. Sarni, J. Eberle, and K. Aberer, "Xgsn: An open-source semantic sensing middleware for the web of things," in *7th International SSN Workshop*, 2014.

[9] 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.

[10] 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

[11] S. Chauhan, P. Patel, A. Sureka, F. C. Delicato, and S. Chaudhary, "IoTSuite: A Framework to Design, Implement, and Deploy IoT Applications: Demonstration Abstract," in *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*, NJ, USA, 2016, pp. 37:1–37:2. [Online]. Available: http://dl.acm.org/citation.cfm?id=2959355.2959392

[12] S. Chen, H. Xu, D. Liu, B. Hu, and H. Wang, "A vision of iot: Applications, challenges, and opportunities with china perspective," *IEEE Internet of Things Journal*, vol. 1, no. 4, pp. 349–359, Aug 2014.

[13] S. K. Datta and C. Bonnet, "Smart m2m gateway based architecture for m2m device and endpoint management," in *2014 IEEE International Conference on Internet of Things (iThings), and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom)*, Sept 2014, pp. 61–68.

[14] S. K. Datta, R. P. F. D. Costa, and C. Bonnet, "Resource discovery in internet of things: Current trends and future standardization aspects," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, Dec 2015, pp. 542–547.

[15] S. K. Datta, A. Gyrard, C. Bonnet, and K. Boudaoud, "onem2m architecture based user centric iot application development," in *2015 3rd International Conference on Future Internet of Things and Cloud*, Aug 2015, pp. 100–107.

[16] J. A. Fisteus, N. F. García, L. S. Fernández, and D. Fuentes-Lorenzo, "Ztreamy: A middleware for publishing semantic streams on the web," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 25, pp. 16–23, 2014.

[17] F. Gao, M. I. Ali, E. Curry, and A. Mileo, "Qos-aware stream federation and optimization based on service composition," *International Journal on Semantic Web and Information Systems (IJSWIS)*, vol. 12, no. 4, pp. 43–67, 2016.

[18] F. Gao, M. I. Ali, and A. Mileo, "Semantic discovery and integration of urban data streams," in *Proceedings of the Fifth International Conference on Semantics for Smarter Cities-Volume 1280*. CEUR-WS. org, 2014, pp. 15–30.

[19] F. Gao, E. Curry, M. I. Ali, S. Bhiri, and A. Mileo, "Qos-aware complex event service composition and optimization using genetic algorithms," in *International Conference on Service-Oriented Computing*. Springer, 2014, pp. 386–393.

[20] J. F. Gomez-Pimpollo and R. Otaolea, "Smart objects for intelligent applications - adk," in *2010 IEEE Symposium on Visual Languages and Human-Centric Computing*, Sept 2010, pp. 267–268.

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

[22] A. Gyrard, P. Patel, S. Datta, and M. Ali, "Semantic web meets internet of things (iot) and web of things (wot)," in *The 15th International Conference on Semantic Web (ISWC).(Oct 2016)*, 2016.

[23] A. Gyrard, M. Serrano, J. B. Jares, S. K. Datta, and M. I. Ali, "Sensor-based linked open rules (s-lor): An automated rule discovery approach for iot applications and its use in smart cities," in *Proc. of the 26th International Conference on World Wide Web Companion*, 2017.

[24] 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

[25] D. Miorandi, S. Sicari, F. D. Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497 – 1516, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570870512000674

[26] F. Paganelli, S. Turchi, and D. Giuli, "A web of things framework for restful applications and its experimentation in a smart city," *IEEE Systems Journal*, vol. 10, no. 4, pp. 1412–1423, Dec 2016.

[27] 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, Perth, Australia, April 3-7, 2017*, 2017, pp. 263–267. [Online]. Available: http://doi.acm.org/10.1145/3041021.3054736

[28] D. Puiu, P. M. Barnaghi, R. Toenjes, D. Kuemper, M. I. Ali, A. Mileo, J. X. Parreira, M. Fischer, S. Kolozali, N. FarajiDavar, F. Gao, T. Iggena, T. Pham, C. Nechifor, D. Puschmann, and J. Fernandes, "Citypulse: Large scale data

analytics framework for smart cities," *IEEE Access*, vol. 4, pp. 1086–1108, 2016. [Online]. Available: http://dx.doi.org/10.1109/ACCESS.2016.2541999

[29] A. Sheth, "Internet of things to smart iot through semantic, cognitive, and perceptual computing," *IEEE Intelligent Systems*, vol. 31, no. 2, pp. 108–112, Mar 2016.

[30] S. Sotiriadis, L. Vakanas, E. Petrakis, P. Zampognaro, and N. Bessis, "Automatic migration and deployment of cloud services for healthcare application development in fiware," in *2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, March 2016, pp. 416–419.

[31] D. Soukaras, P. Patel, H. Song, and S. Chaudhary, "IoTSuite: A ToolSuite for Prototyping Internet of Things Applications," in *The 4th International Workshop on Computing and Networking for Internet of Things (ComNet-IoT), co-located with 16th International Conference on Distributed Computing and Networking (ICDCN)*, 2015, p. 6.

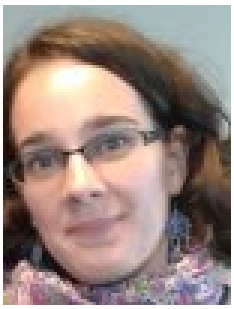
[32] R. Tönjes, P. Barnaghi, M. Ali, A. Mileo, M. Hauswirth, F. Ganz, S. Ganea, B. Kjærgaard, D. Kuemper, S. Nechifor *et al.*, "Real time iot stream processing and large-scale data analytics for smart city applications," in *poster session, European Conference on Networks and Communications*, 2014.

8 Author Biographies

Dr. Muhammad Intizar Ali is an Adjunct Lecturer, Research Fellow and Research Unit Leader of Reasoning, Querying, and IoT Data Analytics Unit at Insight Centre for Data Analytics, National University of Ireland, Galway. His research interests include Semantic Web, Data Integration, Internet of Things (IoT), Linked Data, Federated Query Processing, Stream Query Processing and Optimal Query Processing over large scale distributed data sources. He is actively involved in various EU funded and industry-funded projects aimed at providing IoT enabled adaptive intelligence for smart city applications and smart enterprise communication systems. He is serving as a PC member of various journals, international conferences and workshops. He is also actively participating in W3C efforts for standardization in RDF Stream Processing Community Group. Dr. Ali obtained his Ph.D. (with distinction) from Vienna University of Technology, Austria in 2011.

Pankesh Patel is a Research Scientist at ABB Corporate Research-India. He focuses in building software development methodologies and tools to easily develop applications in the cross-section of Software Engineering, Cyber-Physical Systems, and Internet of Things. He obtained his Ph.D. from the University of Paris VI (UPMC) and INRIA (The French Institute for Research in Computer Science and Automation) Paris, France.

Soumya Kanti Datta is a research engineer in EURECOM, France since 2012 and is working on French national and EU H2020 research projects. His research focuses on innovation, standardization and development of next-generation technologies in Internet of Things, Smart Cities and Cyber Security. He has published more than 60 research papers and articles in top ACM and IEEE Conferences, Magazines and Journals. Soumya has also served several IEEE Conferences and Workshops in many capacities. He is also actively involved in oneM2M, W3C Web of Things Working Group and contributing to their standard development activities. He obtained an M.Sc in Communications and Computer Security from Telecom ParisTech (EURECOM), France.



Amelie Gyrard is a post-doc researcher at Ecole des Mines de Saint-Etienne, France, working within the Connected Intelligence - Knowledge Representation and Reasoning team. Her research interests are on Software engineering for Semantic Web of Things and Internet of Things (IoT), semantic web best practices and methodologies, ontology engineering, reasoning and interoperability of IoT data. She holds a Ph.D. from Eurecom since April 2015 where she designed and implemented the Machine-to-Machine Measurement (M3) framework. She also disseminated her work in standardizations such as ETSI M2M, oneM2M, and W3C Web of Things.