A Service-Oriented Approach for Sensing in the Internet of Things: Intelligent Transportation Systems and Privacy Use Cases

Mohammad Hammoudeh, Gregory Epiphaniou, Sana Belguith, Devrim Unal, Bamidele Adebisi, Thar Baker, A.S.M. Kayes and Paul Watters,

***Abstract*—This paper presents a Sensing-as-a-Service run-time Service Oriented Architecture (SOA), called 3SOA, for the development of Internet of Things (IoT) applications. 3SOA aims to allow interoperability among various IoT platforms and support service-oriented modelling at high levels of abstraction where fundamental SOA theories and techniques are fully integrated into a practical software engineering approach. 3SOA abstracts the dependencies of the middleware programming model from the application logic. This abstraction allows the development efforts to focus on writing the application logic independently from hardware platforms, middleware, and languages in which applications are programmed. To achieve this result, IoT objects are treated as independent entities that may interact with each other using a well-defined message exchange sequence. Each object is defined by the services it provides and the coordination protocol it supports. Objects are then able to coordinate their resources to address the global objectives of the system. To practically validate our proposals, we demonstrate an intelligent transportation system and data privacy functional prototypes as proof of concepts. The use cases show that 3SOA and the presented abstraction language allow the amalgamation of macroprogramming and node-centric programming to develop real-time and efficient applications over IoT.**

***Index Terms*—Sensing as a Service, Internet of Things, Interoperability, Middleware, Coordination, Intelligent Transportation Systems, IoT Security and Privacy.**

I. Introduction

THE Internet of Things (IoT) is one of today's major digital disruptions. It expands to include our vehicles and homes, as well as newly developed wearable and implanted sensors, which bring fundamental transformations to many aspects of daily life [1], [2]. This technological innovation promises to optimise manufacturing processes to improve quality, functionality, and drive revenue growth. For instance, a recent study revealed that car manufacturers estimate the cost of production downtime ranges between £13,000 per minute to a high of £30,000 per minute [3]. One major car manufacturer lowered its factory outage time by 15% by combining multiple data sources to identify factors that led to robotic failures. Such automated systems can also reduce the need for human oversight to improve the productivity of the factory workers, who are still part of the workflow process. Typical IoT objects can operate in a proactive mode by instigating collaborative tasks and dynamically interacting with each other to accomplish both local and global goals, down from the hardware control layer up to the upper layers of the business process management software [4], [5], [6].

The authors of this paper advocate that service-oriented programming models may aid in integrating IoT in smart automation environments; through supporting interoperability and information exchange between heterogeneous devices. In the Service-Oriented Architecture (SOA), systems are composed by organising distributed capabilities into services. A service is a self-contained logical representation of a repeatable application activity that has a specific result, e.g., provide temperature data. Services expose their capabilities through standard service interfaces. In SOA, service interface definitions are obtainable from a service registry. SOA offers uniform means to discover, interact with, and use capabilities of distributed objects that may be under different ownership domains.

In resource-limited large-scale environments, such as IoT, services do not follow the classical service invocation approach of standard SOA. Hence, it is essential to return to the fundamental principles of SOA to investigate how to achieve platform and programming language independence, location transparency, dynamic deployment, and scalability in infrastructure-less networks. To meet IoT resource constraints and application requirements, we propose 3SOA. 3SOA implements a service-oriented approach at all levels of the IoT protocol stack to allow rapid development and management of composite applications. 3SOA introduces a service-oriented and a semantics' layer to the existing network stack. An IoT object is modeled at different levels of abstraction. At each level, a set of services with known attributes and functions are defined. Functions, e.g., multi-modal entity monitoring, are abstracted as lightweight services. A lightweight service has a small memory footprint or has minimalist syntax and features. To allow collaboration between various services running on distributed objects, 3SOA utilises "coordination" models [7]. Coordination models are used to support a separation between

M. Hammoudeh and B. Adebisi are with the Faculty of Science and Engineering, Manchester Metropolitan University, Manchester, United Kingdom, M1 5GD e-mail: m.hammoudeh@mmu.ac.uk).

G. Epiphaniou is with the Warwick Manufacturing Group, University of Warwick, UK.

S. Belguith is with the School of Science, Engineering & Environment, University of Salford, UK.

D. Unal is with the KINDI Center for Computing Research, Qatar University, Qatar.

T. Baker is with the School of Computer Science, Liverpool John Moores University, UK.

A. Kayes and P. Watters are with the School of Engineering and Mathematical Sciences, La Trobe University, Australia.

computation and collaboration between distributed services by the externalisation of interactions among them. Coordination offers runtime supports to changes in the application domain, which may occur at different levels.

To illustrate the application of coordination-enabled 3SOA, we present two use cases, Intelligent Transportation Systems (ITS) and data privacy. The demand for IoT in ITS is growing steadily [8], [9]. Additionally, a coordination-enabled 3SOA use case represents a meaningful and measurable objective of interaction between an actor and the system, such as a vehicle monitoring system to illustrate application composition and its implementation in 3SOA. Furthermore, the data-privacy use case captures one of the most critical non-function requirements of an IoT system [10], [11], [12], [6]. We use these use cases to test and validate the scalability and composability aspects of 3SOA.

The rest of the paper is organised as follows: Section II surveys recent IoT service-oriented middleware in the literature. Section II discusses related works emphasising their drawbacks. In Section III we give what in our view are the foremost challenges of the IoT that can be addressed using service-oriented software development approach. Section IV presents the blueprint of 3SOA. Section V-A gives the details of the abstract programming language that can be used to support the 3SOA. It also considers a scenario where coordination rules are used to compose a hybrid ITS IoT service. Section VII concludes the paper and highlights future research avenues.

II. SOA for IoT in the Literature

SOAs have received the most significant attention in the IoT literature as grounds for providing interoperability, management of big data, security, application development, scalability, amongst others. In this section, we offer a brief discussion from the literature of the theoretical underpinnings of this study. For a more comprehensive literature survey on SOA middleware for IoT and WSN, we refer readers to [13], [14].

Most IoT service-oriented middleware solutions in the literature are designed to provision dynamic and unknown network topology [15]. In this area of research, one stream of work focuses on abstracting IoT objects and turning them into services of the network (e.g., [16]), other researchers focus on information/data abstractions and their integration with services (e.g., [17]). However, a prevalent theme in the literature is addressing the challenge of the unknown topology via the use of service discovery techniques, commonly based on the classical service/resource approach of the Internet, pervasive environments and WSN, e.g., WS-discovery for Web Service or a RESTful discovery for RESTful services [18]. Other solutions propose their service description languages, where service functions and keywords are labeled with an informal (such as simple English) description [19].

Another common thread in the literature is the use of metadata and semantics to address device heterogeneity challenges, e.g., [20]. The use of ontologies to abstract objects, their domains, data repositories, context information and even service descriptions, is a widespread practice. Some researchers, e.g., [21], went as far as proposing virtual machine middleware for service composition to support the notion of virtual/semantic sensors. This model allows to abstract numerous aggregated physical devices in one service. In some research, e.g., [16], virtual sensors may represent transformations applied to a set of sensor streams to create an additional semantically meaningful stream. However, service composition of virtual sensors is not entirely dynamic, because the services are initially specified at design time and dynamically mapped onto the network only at runtime. Yet, the use of small predefined composition building blocks can support a flexible type of composition at the runtime. For instance, Badawy et al. [22] attempt to maximise the composite service quality at the IoT application layer by balancing service reliability and acceptable cost of computation time. This effort is limited to particular use cases and does not consider other IoT critical features such as power consumption.

Concerning scalability, most IoT SOA middleware research handles this requirement by adjusting the underlying network topology, e.g., [23]. This can be achieved via a fully decentralized [24] or a peer-to-peer infrastructure [25]. These approaches work well for the standard Internet, where there is a relatively small amount of service interactions, but they are not suitable for the sophisticated large number of interactions that will be routine in IoT ecosystems such as smart cities. Executing even a basic service discovery or composition in such environments may go beyond acceptable memory, processing and time constraints. Therefore, the scalability challenge must be addressed by modification of the discovery and composition techniques, rather than aiming at the design of optimal network topologies.

It is evident in the literature that additional research is essential to provide comprehensive SOA solutions meeting service and network providers' requirements. For instance, there is need for more work to handle conflict resolution, inaccurate metadata and data-point availability. To address some of these issues, 3SOA depends on the structured nature of physical information. 3SOA supports semantic and estimation models that implement these tasks transparently from the application. The process we envisage should be achieved in a fully automated mode.

III. SOA Optimisation for IoT

The service composition model to support complex service composition in IoT ecosystems is different from the one that is used in Web SOA. For instance, Quality of Service (QoS) constraints, such as real-time monitoring, restrict a full-fledged SOA approach. To meet such constraints and other requirements, 3SOA implements a service-oriented paradigm at all levels of the IoT protocol stack to allow rapid development and management of composite applications that dynamically discover and use distributed services in a peer-to-peer model. Focus is placed on the design of IoT services coordination to facilitate programming language independent asynchronous communication in stream-based, large-scale, infrastructure-less and loosely coupled IoT applications.

One of the main design factors of 3SOA is to deal with the huge volume of data generated by IoT. The heterogeneity of IoT operating platforms, programming models and

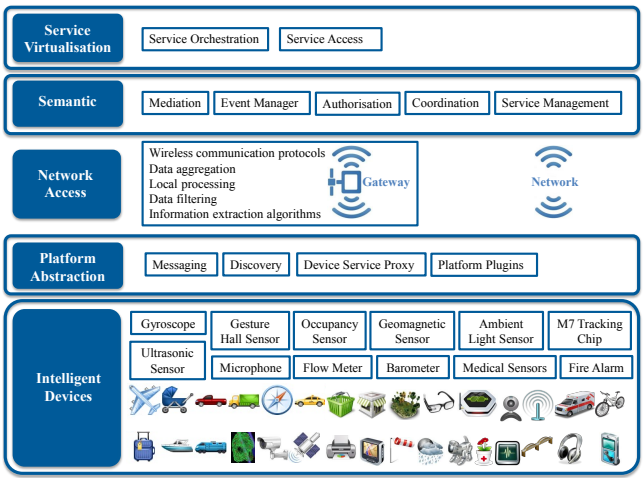

Fig. 1. Proposed 3SOA IoT-SOA integration framework.

data retrieval and processing renders information extraction a challenging task. Information extraction is defined as "the process of retrieving, filtering and processing unstructured data from IoT objects to discover or identify a specific state or condition". This task is not just a network self-configuration problem, but the key challenge here is the heterogeneity of the data retrieval and processing protocols. From the viewpoint of an integrated-system behaviour, information extraction tasks require considerable programming effort, particularly when considering constraints on the objects and network resources. To address this issue, 3SOA adds a service-oriented and semantics layer to the existing network stack.

In 3SOA, an object is modeled at different levels of abstraction. At each level, a set of services with measurable preconditions and expectations are defined. Functions such as the information extraction methods are abstracted as lightweight services to suit an object resource constraints. 3SOA uses coordination models [7] to allow peer-to-peer collaboration between different services running on distributed objects. Coordination languages are used to define the detailed logic structure for the processes of the service workflow. They support SOA to enable seamless runtime integration of heterogeneous information services in a distributed manner through semantic modeling primitives for coordinating such interactions, including support for location-aware computing. 3SOA extends the standard IoT reference architecture to enable semantics-based management of the complete service life cycle. In the following section, we introduce the 3SOA middleware.

IV. 3SOA Blueprints

3SOA, as depicted in Figure 1, consists of multiple layers that host various components. A description of these layers, and their constituent components, is below.

1) Service Virtualisation Layer (SVL): SVL presents the functional capabilities of the underlying IoT cloud as services. The network acts as a provider of remotely accessed services, abstracting each available object as an independent web service(s). SVL allows applications to interact with and utilise data from a variety of networked objects using standard interfaces. A service interface defines the services offered by an object, consults the registry, checks and registers events and other maintenance tasks. Moreover, a service knowledge base is built on the network gateway to enable service users to determine how many services are available and how to access them. To achieve this aim, the SVL layer performs many tasks, including probing the knowledge base for available services and transforming them into a semantic enhanced service description. Then, it creates lightweight formats of available service descriptions for publishing them on external servers to increase their visibility.

The service access function initialises servers and the communication function at the lower layer to enable sending commands directly to the network without passing through a task management registry. The service orchestration function allows users to compose complex services using the descriptions of the published service. This will enable objects to directly participate in business processes without demanding a process modeller or a process execution engine to learn about the details of the underlying hardware platform.

2) Semantic Layer (SL): The SL core objective is to provide the semantic model of the underlying IoT cloud by maintaining the structure and interactions of available services using Module Interconnection Languages (MIL) or another form of ontology, e.g., [20]. MIL is a powerful way to express and handle information transfer between heterogeneous modules, distributed network communication protocols, dynamic queries and updates by users and dynamically changing the specifications of interfaces. The MIL semantic is used to specify the basic structure of a system, in terms of its components. This specification includes the relationships and interactions among the components as well as the principles controlling the design and evolution of those components. It addresses the task of integrating independently-developed subsystems by providing rule-based semantic authorisations. This layer allows importing any IoT subsystem in MIL and translates it to Web Ontology Language (OWL) description.

A local repository maintains all available or connected service instances. A messaging system allows applications to accept any events or notifications and interact with objects, which are intermittently connected. This messaging system offers a runtime environment for the execution of composite services. It provisions business process composition through an execution service for under-specified Business Process Execution Language (BPEL) processes. The messaging system is an essential feature as the services offered at the object level are dynamic, e.g., they connect and disconnect.

The service management function supports configuration and control of service components (service management), offers external communication for application service components (data management), manages the networking stack (network management) and provides ways for controlling and configuring applications (application management). A key element of the management function is the business and process rule engines. IoT combines interaction and connection among objects and systems producing results as events or contextual data. Many events need filtering or forwarding to

post-processing systems, e.g., periodic data collection, whereas others require an immediate response, e.g., fire alarm. The rule engine provision the construction of decision logics and activate interactive and autonomous processes to support a more responsive IoT system.

3) Network Access Layer (NAL): NAL hosts a set of access and communication common facilities. These facilities include management services for data handling, data processing through filtering and aggregation techniques, time synchronisation, localisation, etc. Since all applications use these facilities, NAL is provided at the middleware layer of IoT objects.

It is possible to have multiple networks in the same IoT cloud, where each network has its own gateway. The communication component allows applications to directly send requests to the network, without passing across the task management registry. The control component is responsible for initializing the cloud-based SOA server and handles the interaction with internal services offered by the objects. The message processor deals with service registration with the registry component located at the upper layers, which allows quicker response to urgent requests.

4) Platform Abstraction Layer (PAL): Ideally, objects offer discoverable web services on IP networks. Such objects provide an interface to access services they offer, check the registry, and register events. Since services are discovered and executed across various development and operating platforms, PAL can leverage legacy components of a different platform that uses various technologies. The key challenge becomes how to compose heterogeneous objects into a larger application.

Where objects do not support SOA, e.g., they use data-centric or message-based communication protocols, service encapsulation becomes a complex task. Besides enabling the execution of services on objects, this layer also provides a unified view on remotely injecting or updating the software modules running on objects. It also allows homogeneous objects to communicate with each other and with the back-end devices using their native protocols.

PAL enables an adapter-oriented approach to tackle the wide range of communication protocols and sensor types offered by various objects. The main objective of this layer is to measure the physical properties of the environment and convert it into a form that can be carried to the upper layers for further processing. It receives and delivers sensor events to an event handler through a callback message routine, where messages are sent asynchronously to the destination for processing. This layer also receives request messages from upper layers, which then chooses a suitable adapter to forward each request to its intended cloud-connected object.

5) Intelligent Devices Layer (IDL): IDL hosts all the smart objects that will connect to the IoT cloud. Smart objects are equipped with sensors that allow active interaction between the environment and the digital world. Many types of sensors and actuators exist to achieve various objectives. This layer resides in all objects and offers interfaces to the fundamental hardware components. It includes all the functionality of a smart object.

Objects offer several internal services and have many sub-components, such as the Operating System (OS), acquiring sensor data, controlling actuators, etc. The OS component is made up of the networking stack, the scheduler and various sensor and hardware drivers. The embedded OS is a vital sub-component, which abstracts low-level details, including communication with the hardware platform, topology control, etc.

All objects are autonomously discovered, monitored and their status is presented to enterprise servers through local gateways. The dynamic discovery process allows the deployment of a generic software object, that by an autonomous hardware recognition, can identify what sensing functions the physical object can offer and what kind of actuators it can control. Moreover, it is feasible to implement new services during run-time to meet application requirements remotely. If autonomous recognition is not possible, the component can only offer pre-known hardware capabilities.

Most objects require connectivity to a data aggregator or gateway. Some objects that do not need connectivity to data aggregators, instead, their connectivity to backend applications/servers is provided over a Wide Area Network technology, such as 4G or LTE.

V. Coordination Support for 3SOA

In Section IV, a service-oriented middleware for the deployment of lightweight services in IoT has been presented. In this section, we present the high-level coordination abstractions used at the SVL and PAL to support application developers in implementation tasks, such as communication, group formation, etc. Coordination is the process of composing programs by binding together separate activities into an ensemble. In 3SOA, programmers expand two parts of the middleware to develop a new application:

1) A high-level abstract part to integrate a set of potentially heterogeneous software/hardware objects through interfacing with each object such that the collective set forms an application that runs on and takes advantage of a distributed system. In the field of software engineering massively parallel and distributed systems, coordination languages has proven to be an effective tool to enhance modularity, reuse of existing components, interoperability and portability. In 3SOA, platform-independent abstractions are used to declare the core elements of the system, specifically, nodes (including sensors and actuators), clusters or groups, communication and services. These different granularity units can be used to design and implement complex and large-scale applications. Through ports or channels, services can be composed to build complex applications. Services use ports to send out to their environment *control messages* or *events* to notify collaborating entities of their *state* or state changes. Often, high-level abstract coordination languages come with a translator, which takes the declarative part as input and produces the skeleton of the IoT application.
2) A platform-dependent part in which the programmers add the task implementation code to the skeleton to

implement the services specified in the declarative part. The application programmer uses the platform-specific facilities to provide the hardware, e.g., NesC for MicaZ motes and MicroPython for Pycom devices.

The resulting code, i.e., the skeleton and the task code, forms the 3SOA-based IoT application that will interact with lower middleware layers through APIs. In the next subsection, we describe the languages forming the family of coordination formalisms, which can be used in the declarative part of the middleware. Subsections V-B and V-C, give two use cases to demonstrate the expressiveness of the coordination models and their application in the development of service-oriented IoT applications.

A. 3SOA Abstract Coordination Languages

Based on the concept of coordination, many models and formalisms evolved for describing distributed and concurrent systems, e.g. [26], [19]. 3SOA does not impose any formalism or language, but it works best with "control-driven" coordination models [27]. In control-driven, or otherwise known as "task-oriented" coordination models, the state of coordination is defined in terms of the coordinated activities involved by the actors, i.e., the computational part of the system is viewed as black boxes with clearly defined input/output interfaces.

Processes in 3SOA interact with their environment through clearly defined interfaces, i.e., input/output ports. Consumer-producer relationships are established by setting up *channel* connections between input ports of producers and output ports of consumers. Moreover, processes can send events or control messages to their environment to notify interested processes with their *state*.

The 3SOA works with any coordination language or model that is CSP- or Occam-like formalism, e.g., [26]. The flexible design of 3SOA allows events to be represented as parametric with types and data values or as simple units representing state changes. Events can be advertised by broadcast mechanisms different to channel connections.

B. Intelligent Transportation Systems Use Case

This subsection describes how the primitives of a coordination language, namely USEME [28], is used to support effective collaboration between IoT objects to achieve common application goal. Also, it presents an ITS application scenario to illustrate how different IoT application services can be coordinated.

USEME is an intuitive and expressive high-level abstract language for service definition and composition and real-time support [28]. It meets all of the conditions defined in Subsection V-A. It has many desirable features, such as allowing programmers to define QoS constraints in the communication between services. In USEME, the use of *ports* and *services* to coordinate distributed application functions is based on a proactive system. A USEME system actively seeks to influence its environment, rather than just react to external stimuli. This property preserves the autonomy of each participant IoT object.

In Listings 1 to 3, we give the coordination abstractions in USEME to integrate two vehicle monitoring services into a single application that runs over a distributed IoT infrastructure. Listing 1 defines two ports templates. The *LidarPort* defines three operations. The asynchronous command *GetLidar* is used by the vehicle monitoring service to obtain the Lidar reading. A lidar (light detection and ranging) identifies lane markings and the edges of the roads. The event *DangerLidar* will inform the driver actor when a dangerous roadside object is detected. Another event, the *Lidar* event, is used to to send lidar readings to the vehicle control system periodically. The *UltrasonicPort* defines the *GetRUltrasonic*, which is used by the vehicle monitoring service to measure the position of objects close to the vehicle, e.g. curbs when parking. These ports are either provided or required by the three defined services. The *LidarService* and the *UltrasonicService* provide the *LidarPort* and *UltrasonicPort*, respectively, i.e., these services are information sources. The *VehicleMonitorService*, Listing 2 requires the ports provided by the *LidarService* and the *UltrasonicService* and provides the *DriverAlarmPort*.

Listing 1. Ports and services definitions.

```
Port LidarPort
{
  CommandSync GetLidar(out int value);
  Event DangerLidar(out int value);
  Event Lidar(out int value);
}

Port UltrasonicPort
{
  CommandSync GetRespiration(out int value);
}

Service LidarService
{
  Description = ''Lidar service'';
  Provides LidarPort with constraints{
    on Lidar Period 600,
      Priority 2;
    on DangerLidar Deadline 5,
      Priority 1,
      Reliable;
  }
}

Service UltrasonicService
{
  Description = ''Ultrasonic service'';
  Provides UltrasonicPort with constraints{
    on GetUltrasonic Reliable;
  }
}
```

Listing 2. Vehicle monitor service definition.

```
Service VehicleMonitorService
{
  Description = ''VehicleMonitor Service'';
  Requires LidarPort with constraints{
    on DangerLidar Reliable;
  }
  Requires UltrasonicPort with constraints{
    on GetUltrasonic Deadline 1000,
      Priority 2,
      Reliable;
  }
    Provides DriverAlarmPort with constraints{
    on Emergency Deadline 1000,
      Priority 2,
```

```
      Reliable;
    on ActivateAlarm Reliable;
  }
}
```

After defining the provided and required ports in every service, the programmer can specify constraints in the service template, which must be satisfied by the service operation. In the *UltrasonicService*, the only constraint on the *UltrasonicPort* is to be run reliably. The *Lidar* function is set with a period of 10 milliseconds; hence, objects that publish this service in a group will inform other group members of the Lidar reading every 10 milliseconds. The *DangerLidar* is specified as a reliable operation with level 1 priority, highest priority, and a deadline of 5 milliseconds. If the object sending the notification does not receive an acknowledgment within 5 milliseconds, the event will be re-transmitted. In the *VehicleMonitorService*, Listing 2, both the *LidarPort* and the *DangerLidarPort* are required. The *DriverAlarmPort* port is used to notify the driver that an emergency has been detected and to activate an alarm to alert nearby drivers or emergency services.

Listing 3. Object and group definitions.

```
Sensor template LidarSensor(loc)
{
  Type = ''Surveying'';
  Publish LidarService
    in groups VehicletMonitorGroup(loc);
}

Sensor template UltrasonicSensor(loc);
{
  Type = ''Detection'';
  Publish UltrasonicService
    in groups VehicleMonitorGroup(loc);
}

Actor template DriverActor(loc);
{
  Publish DriverMonitorService
    in groups DriverMonitorGroup(loc);
}

Group template DriverMonitorGroup(loc);
{
  DeviceType = Both
  SensorType = ''Surveying'' or ''Detection''
  Cardinality = (Actor, 1)
    (''Surveying'', 3)
    (''Detection'', 3);
}
```

In Listing 3, we define two node templates, the *LidarSensor* and the *UltrasonicSensor*. Both templates accept *loc*, which describes where the node will be placed, e.g., front, back, left or right. The attribute *Type* defines the sensing modalities available for each node template. Both node types publish their services in the group *DriverMonitorServiceGroup*. The *DriverActor* is the actuator that implements the vehicle monitoring activity. *DriverMonitorServiceGroup* group template takes a parameter to specify where the group will be located. The *DeviceType* indicates that both sensors and actors can join the group. The *SensorType* attribute specifies the type of sensors that can join the group. The *Cardinality* attribute indicates that the group must have a maximum of one actor and three nodes for the two defined sensor types.

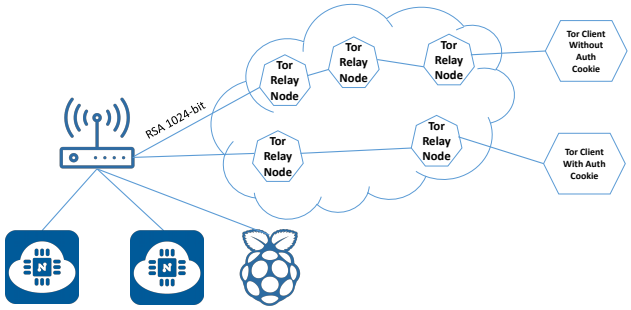

Fig. 2. The blueprint of the implemented IoT network.

C. Device Anonymity Use Case

We present a second use case to extend the open-source IoT platform Home Assistant to provide IoT objects as hidden services on the Tor anonymous network. A privacy service was selected to show the feasibility and applicability of the 3SOA in an under-investigated area of SOA. Moreover, security is an ever-increasing concern for IoT users which has not kept in pace with the growing challenges of cyber attacks.

Establishing a security model that uses the Tor network to house the IoT network would eliminate most security concerns that arise when implementing an IoT application. For instance, using Tor's end-to-end encryption will remediate security attacks such as Side-Channel Attacks, cryptanalysis attacks, and network attacks (e.g., Denial of Service). Also, keeping anonymity by using relay nodes to mask the object's IP address preserves the privacy of the system owner. Remote connections are also secured by encryption, and relay nodes mask the incoming IP and what the user is trying to access.

The solution proposed in this scenario is to use the Tor network to host a versatile web server that IoT developers can attach their API to. Figure 2 depicts two ESP8266 device equipped with vibration sensors are connected to a Raspberry Pi through a wireless router. The information from the ESP8266 device will be sent using the Message Queue Telemetry Transport (MQTT) protocol. The open source Mosquitto MQTT broker is used to handle MQTT tickets over the local IoT network. The broker is triggered every time there is an event to send. The information from the two ESP8266 devices is then aggregated so the Home Assist API can use it to display information to the user. After the information is sent to the API, the server advertises this information to the hidden service over the Tor network. The hidden service is then protected by encrypting all packets with RSA 1024-bit encryption. In Figure 2, the two clients are connecting to the hidden service. Both clients know the destination address of the hidden service, but only one had an authentication cookie for the service. The client without the authentication cookie will be re-directed to an unroutable address and the authentic client is routed through the service using Tor encryption.

Listing 4 shows the abstract definitions entered to the Home Assist extension to integrate the two ESP8266 devices into a composite service.

Listing 4. ESP8266 devices and service definitions.

```
Port SoundPort
```

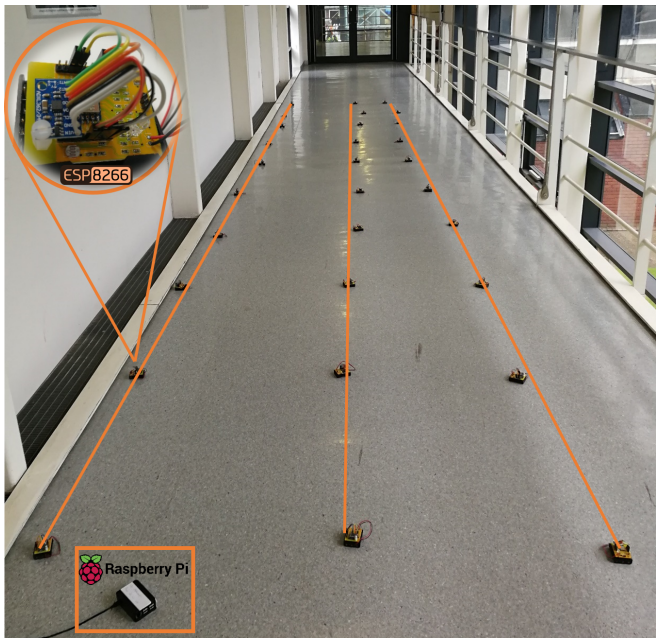

Fig. 3. Experimental setup of the 3SOA prototype.

```
{
  CommandSync GetSound(out int value);
  CommandAsync SetThreshold(in int value);
  Event HighSound(out int value);
  Event Sound(out int value);
}

Service SoundService
{
  Description = ''Sound Service'';
  Provides SoundPort;
}

Sensor template ESPSensor(id, loc)
{
  ID = ident;
  Location = loc;
  SensorType = ''Sound'';
  Publish SoundService in groups SmartHome(loc);
}

Create Sensor ESPSensor(1, ''kitchen'');
```

VI. Evaluation

We have implemented a 3SOA prototype and tested it based on the two use cases described in Subsections V-B and V-C. The purpose of the prototype is to provide a testbed to evaluate 3SOA scalability and composability. For this objective, the main technology selection consideration was to use an easily programmable hardware platform without requiring the use of low level languages. The devices must also be inexpensive, to allow large networks to be built. ESP8266 meets all the criteria mentioned above. It combines a 32-bit RISC processor, 32 KiB instruction and 80 KiB user data memory, an IEEE802-11a/b/n protocol engine, and a Wi-Fi PHY onto a single chip. ESP8266 is available ready-packaged on a module, with serial interfaces to connect a variety of sensors. The setup used in this prototype is shown in Figure 3. The use of ESP8266 has some obvious

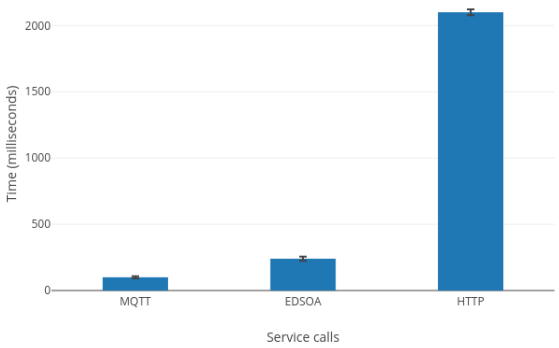

Fig. 4. The average end-to-end delay with a single sensor device. Vertical bar signifies the SVL/PAL and the event manager in 3SOA and EDSOA respectively.

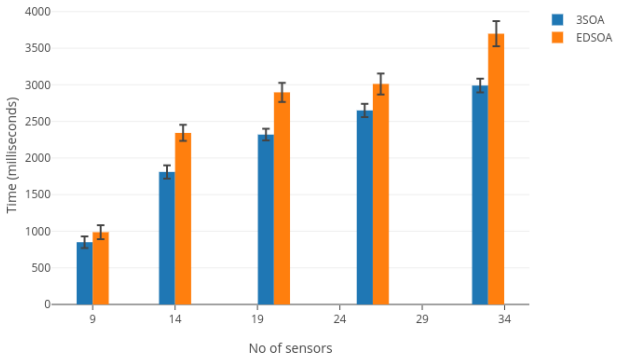

Fig. 5. SVL/PAL and the event manager overhead at different node density.

limitations, particularly high power consumption and network security vulnerabilities. While these are significant for an operational network, for a prototype, they are of less concern. For prototyping, the use of the IP stack and the IEEE802 wireless protocols make a great deal of the system software and infrastructure easily accessible.

We implemented 3SOA on 30 ESP8266 devices and one Raspberry Pi gateway to demonstrate its feasibility to be deployed in real-world settings. In the patient monitoring service we use simulated lidar and ultrasonic sensors, i.e., devices send dummy payload, due to lack of access to such resources. Most of the service-oriented middleware or frameworks for IoT in the literature, e.g., [29], provide only the implementation of the presented case studies. There are a small number of works that give insights on the scalability and composability of the proposed work. One specific approach that is similar to 3SOA is the Event-Driven SOA (EDSOA) [30]. Both approaches target infrastructure-less networks to deliver real-time distributed service composition. We compare the performance of 3SOA against that of EDSOA. This comparison is fair as the two

works are similar in spirit but different in approach.

We consider both qualitative and quantitative sides for the evaluation of 3SOA. Concerning the qualitative side of 3SOA, we add to advancing the IoT services through 3SOA's SVL/-PAL, towards the standard web services discovery protocols. We utilized MQTT for connectivity between IoT devices and enterprise applications. Furthermore, coordination languages were used as programming abstractions for application developers to quickly compose IoT applications with least knowledge of underlying binding technologies. As for the quantitative evaluation, we investigate end-to-end delay caused by SVL/PAL and the memory overhead.

The objective of measuring the latency introduced by 3SoA's SVL/PAL is to assess the feasibility of the publish-subscribe mechanism for resource-constrained IoT environments. In this context, resource-constrained referrers to IoT device hardware limitations in terms of communication range and bandwidth, memory, processing and power supply as well as to limited network infrastructure such as servers or gateways.

We run an experiment for both studied use cases. For comparison, we use both MQTT and HTTP (baseline) protocols. In the first experimental run, we consider that only one device has subscribed for the publication to the 3SOA with only one business process subscriber. Figure 4 shows the average latency over 10 readings. The HTTP incurs higher overhead due to increased payload size. On the other hand, the overhead caused by 3SOA's SVL/PAL is $8ms$, which is insignificant ($< 10\%$) compared to the total latency.

In the second experimental run, each device publishes the lidar and ultrasonic data that has been subscribed by 10 different business processes. We ran the experiment until each device generates 300 messages. The latency results are shown in Figure 5. We calculate the average latency for each device over 100 readings where the average delay by 3SOA is only $87ms$ in which the average latency caused by the SVL/PAL is $13ms$, which is only $4ms$ increase from the first experimental run.

Compared to EDSOA, 3SOA reduces the end-to-end delay before a service is composed of, up to $43ms$. The reduction is primarily due to the pre-defining composition rules in 3SOA. Although such rules are static, their execution is dynamic. In 3SOA, coordination rules can be executed in parallel on multiple devices. Devices that are in the same coordination state will be ready to communicate with other devices with no delays.

VII. Conclusion

To address the challenges facing programmers developing IoT applications, we adopt a service-oriented approach that relies heavily on semantics to describe devices, their data, and their physical attributes. This paper advocates 3SOA, a runtime approach, for the development of applications for service-oriented IoT. We demonstrated coordination abstractions as a practical method to alleviate the problems related to the development of complex, highly decentralised and parallel systems. IoT objects are abstracted and turned into autonomous services that interact with each other using well-defined interfaces. Each object is defined by the services it provides and the coordination protocol it supports. Objects are then able to coordinate their resources to address the global structure of the system.

After presenting 3SOA and associated object interaction and coordination, two uses cases were described to demonstrate the scalability and composability of the proposed work. These use cases show 3SOS's ability to coordinate the operation of heterogeneous objects and the composition of a complex application. Finally, a functional prototype was presented and evaluated to assess the feasibility and performance of the proposed 3SOA. 3SOA was found to outperform its best rivals in the literature in terms of end-to-end delay and service management overhead.

There are many future avenues to be followed, such as the development of graphic support to assist application developers to construct IoT applications in an interactive visual manner. Additionally, more investigation is needed on the group abstraction level as a critical entity for service composition and achieving interoperability. Such compositions must be thoroughly evaluated by joining nodes with various mission restrictions.

References

[1] R. Ande, B. Adebisi, M. Hammoudeh, and J. Saleem, "Internet of things: Evolution and technologies from a security perspective," *Sustainable Cities and Society*, p. 101728, 2019.

[2] H. Liao, Z. Zhou, X. Zhao, L. Zhang, S. Mumtaz, A. Jolfaei, S. H. Ahmed, and A. K. Bashir, "Learning-based context-aware resource allocation for edge computing-empowered industrial iot," *IEEE Internet of Things Journal*, pp. 1–1, 2019.

[3] S. Groschupf, "The Year of Big Sensor Data," http://sandhill.com/article/2014-the-year-of-big-sensor-data/, 2013, [Online; accessed 8-June-2018].

[4] D. Zhang, Y. Liu, L. Dai, A. K. Bashir, A. Nallanathan, and B. Shim, "Performance analysis of fd-noma-based decentralized v2x systems," *IEEE Transactions on Communications*, vol. 67, no. 7, pp. 5024–5036, July 2019.

[5] M. Walshe, G. Epiphaniou, H. Al-Khateeb, M. Hammoudeh, V. Katos, and A. Dehghantanha, "Non-interactive zero knowledge proofs for the authentication of iot devices in reduced connectivity environments," *Ad Hoc Networks*, vol. 95, p. 101988, 2019.

[6] S. Belguith, N. Kaaniche, M. Hammoudeh, and T. Dargahi, "Proud: Verifiable privacy-preserving outsourced attribute based signcryption supporting access policy update for cloud assisted iot applications," *Future Generation Computer Systems*, 2019.

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

[8] A. Muthanna, R. Shamilova, A. A. Ateya, A. Paramonov, and M. Hammoudeh, "A mobile edge computing/software-defined networking-enabled architecture for vehicular networks," *Internet Technology Letters*, vol. n/a, no. n/a, p. e109, 2019.

[9] S. Walker-Roberts, M. Hammoudeh, O. Aldabbas, M. Aydin, and A. Dehghantanha, "Threats on the horizon: Understanding security threats in the era of cyber-physical systems," *Journal of Supercomputing*, 2020.

[10] M. Chowdhury, A. Kayes, P. Watters, P. Scolyer-Gray, and A. Ng, "Patient controlled, privacy preserving iot healthcare data sharing framework," in *Proceedings of the 53rd Hawaii International Conference on System Sciences*, 2020.

[11] Z. Zheng, T. Wang, J. Wen, S. Mumtaz, A. K. Bashir, and S. H. Chauhdary, "Differentially private high-dimensional data publication in internet of things," *IEEE Internet of Things Journal*, pp. 1–1, 2019.

[12] M. Shafiq, X. Yu, A. K. Bashir, H. N. Chaudhry, and D. Wang, "A machine learning approach for feature selection traffic classification using security analysis," *The Journal of Supercomputing*, vol. 74, no. 10, pp. 4867–4892, 2018.

[13] R. Alshinina and K. Elleithy, "Performance and challenges of service-oriented architecture for wireless sensor networks," *Sensors*, vol. 17, no. 3, 2017.

[14] A. N. Lam and Ã. Haugen, "Applying semantics into service-oriented iot framework," in *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, vol. 1, July 2019, pp. 206–213.

[15] R. K. Behera, K. H. K. Reddy, and D. Sinha Roy, "Modeling and assessing reliability of service-oriented internet of things," *International Journal of Computers and Applications*, vol. 41, no. 3, pp. 195–206, 2019.

[16] C. Martín, M. Díaz, and B. Rubio, "Run-time deployment and management of coap resources for the internet of things," *International Journal of Distributed Sensor Networks*, vol. 13, no. 3, p. 1550147717698969, 2017.

[17] T. Fronimos, M. Koutsoubelias, S. Lalis, and T. Bartzanas, *A Service-Based Approach for the Uniform Access of Wireless Sensor Networks and Custom Application Tasks Running on Sensor Nodes*. Cham: Springer International Publishing, 2018, pp. 77–101.

[18] M. Hamzei and N. Jafari Navimipour, "Toward efficient service composition techniques in the internet of things," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3774–3787, Oct 2018.

[19] B. Cheng, D. Zhu, S. Zhao, and J. Chen, "Situation-aware iot service coordination using the event-driven soa paradigm," *IEEE Transactions on Network and Service Management*, vol. 13, no. 2, pp. 349–361, 2016.

[20] M. Boulakbech, N. Messai, Y. Sam, T. devogele, and M. Hammoudeh, "Iot mashups: From iot big data to iot big service," in *Proceedings of the International Conference on Future Networks and Distributed Systems*, ser. ICFNDS '17. New York, NY, USA: ACM, 2017.

[21] C.-L. Fok, G.-C. Roman, and C. Lu, "Servilla: a flexible service provisioning middleware for heterogeneous sensor networks," *Science of Computer Programming*, vol. 77, no. 6, pp. 663–684, 2012.

[22] M. M. Badawy, Z. H. Ali, and H. A. Ali, "Qos provisioning framework for service-oriented internet of things (iot)," *Cluster Computing*, pp. 1–17, 2019.

[23] J. Abdelaziz, M. Adda, and H. Mcheick, "An architectural model for fog computing," *Journal of Ubiquitous Systems and Pervasive Networks*, vol. 10, no. 1, pp. 21–25, 2018.

[24] C. Cambra, S. Sendra, J. Lloret, and L. Garcia, "An iot service-oriented system for agriculture monitoring," in *2017 IEEE International Conference on Communications (ICC)*, May 2017, pp. 1–6.

[25] I. Chen, J. Guo, and F. Bao, "Trust management for soa-based iot and its application to service composition," *IEEE Transactions on Services Computing*, vol. 9, no. 3, pp. 482–495, May 2016.

[26] S. Mount, M. Hammoudeh, S. Wilson, and R. M. Newman, "Csp as a domain-specific language embedded in python and jython." in *CPA*, ser. Concurrent Systems Engineering Series, P. H. Welch, H. W. Roebbers, J. F. Broenink, F. R. M. Barnes, C. G. Ritson, A. T. Sampson, G. S. Stiles, and B. Vinter, Eds., vol. 67. IOS Press, 2009, pp. 293–309.

[27] G. Ciatto, S. Mariani, M. Louvel, A. Omicini, and F. Zambonelli, "Twenty years of coordination technologies: State-of-the-art and perspectives," in *International Conference on Coordination Languages and Models*. Springer, 2018, pp. 51–80.

[28] E. Cañete, J. Chen, M. Díaz, L. Llopis, and B. Rubio, "A service-oriented approach to facilitate wsan application development," *Ad Hoc Networks*, vol. 9, no. 3, pp. 430 – 452, 2011.

[29] M. R. B. A. Rahim, N. Fisal, R. A. Rashid, and Z. Khalid, "A service oriented middleware for smart home and ambient assisted living," in *2015 1st International Conference on Telematics and Future Generation Networks (TAFGEN)*, May 2015, pp. 49–53.

[30] L. Lan, F. Li, B. Wang, L. Zhang, and R. Shi, "An event-driven service-oriented architecture for the internet of things," in *2014 Asia-Pacific Services Computing Conference*, Dec 2014, pp. 68–73.