DIAT: A Scalable Distributed Architecture for IoT

Chayan Sarkar, *Graduate Student Member, IEEE*, Akshay Uttama Nambi S. N., *Graduate Student Member, IEEE*, R. Venkatesha Prasad, *Senior Member, IEEE*, Abdur Rahim, *Member, IEEE*, Ricardo Neisse, and Gianmarco Baldini, *Member, IEEE*

***Abstract*—The advent of Internet of Things (IoT) has boosted the growth in number of devices around us and kindled the possibility of umpteen number of applications. One of the major challenges in the realization of IoT applications is interoperability among various IoT devices and deployments. Thus, the need for a new architecture—comprising smart control and actuation—has been identified by many researchers. In this paper, we propose a Distributed Internet-like Architecture for Things (DIAT), which will overcome most of the obstacles in the process of large-scale expansion of IoT. It specifically addresses heterogeneity of IoT devices, and enables seamless addition of new devices across applications. In addition, we propose an usage control policy model to support security and privacy in a distributed environment. We propose a layered architecture that provides various levels of abstraction to tackle the issues such as scalability, heterogeneity, security, and interoperability. The proposed architecture is coupled with cognitive capabilities that helps in intelligent decision-making and enables automated service creation. Using a comprehensive use-case, comprising elements from multiple-application domains, we illustrate the usability of the proposed architecture.**

***Index Terms*—Dynamic service creation, Internet of Things (IoT) distributed architecture, usage control policies.**

I. Introduction

In TODAY'S connected world, there are several means of ephemeral communication among devices, e.g., Bluetooth, GSM, NFC, WiFi, and ZigBee. However, the idea now is not only to connect with other communicating devices opportunistically, but also to be aware of various real-world noncommunicable objects in the surroundings. This paradigmatic shift opens up new futuristic services. An important aspect of these services can be captured by the words of Mark Weiser. In his seminal paper [1], he provided a vision—"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it." This vision, in fact, is the driving force behind today's miniaturized technologies and communication substrates. Thus, we are about to witness a future where there will be thousands of inanimate objects for each person that will seamlessly communicate with each other to support everyday life in a smart way. In general, this paradigm is referred to as the "Internet of Things" (IoT). The idea is to form an intelligent network of these humongous numbers of devices, systems, and equipments. Furthermore, the ambit of IoT is expanding to include any "thing" that could be represented or identified in the cyber (virtual) world even without having any direct communication interfaces on these "things."

Many IoT applications have been identified, e.g., smart home, smart logistics, smart transportation, smart health care, and smart agriculture [2]. A common factor in all such applications is the inherent *smartness*. Being part of a "smart" application, various devices within an application domain can automatically collect data, share information among themselves, and initiate and execute services with minimal human intervention. Some of the desired characteristics of IoT objects (devices)[1] as well as IoT applications are listed below [3].

1) *Automation*: Automation is a key feature of any IoT device and application. Autonomous data collection, processing, contextual inference, collaborating with other IoT objects, and decision-making should be supported by any IoT infrastructure.
2) *Intelligence*: Objects in IoT should act intelligently. Building intelligence into these objects and empowering them to operate adaptively based on different situations is an important aspect. Situation and context awareness are the key entities for an intelligent system, which can operate with minimal human intervention.
3) *Dynamicity*: An object in an IoT ecosystem can move from one place to another place. The IoT ecosystem should be able to dynamically recognize and adapt these objects based on the environment. Thus, dynamic management and integration of these objects across different environments and applications is crucial for a unified IoT architecture.
4) *Zero configurations*: To support easy integration of devices in the IoT ecosystem, plug-and-play feature should be available. Zero-configuration support encourages an easy and decentralized growth of IoT systems [4].

The main challenge in IoT is to manage and maintain large number of devices and react smartly according to the data generated by them. Some answers addressing this challenge can be seen under the umbrella of future Internet and in projects

Manuscript received May 15, 2014; revised October 23, 2014; accepted December 16, 2014. Date of publication December 31, 2014; date of current version May 20, 2015. This work was supported by the European Commission's Seventh Framework Programme (EU FP7) under Grant 287708 and by the iCore project.

C. Sarkar, A. U. Nambi S. N., and R. V. Prasad are with the Faculty of Electrical Engineering, Mathematics, and Computer Science, Delft University of Technology, Delft 2628 ZH, The Netherlands (e-mail: chayan@ieee.org; akshay.narashiman@tudelft.nl; rvprasad@ieee.org).

A. Rahim is with CREATE-NET, Trento, Italy (e-mail: abdur.rahim@create-net.org).

R. Neisse and G. Baldini are with European Commission Joint Research Center (JRC), Ispra 21027, Italy (e-mail: ricardo.neisse@jrc.ec.europa.eu; gianmarco.baldini@jrc.ec.europa.eu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JIOT.2014.2387155

[1] We use the terms IoT object and IoT device interchangeably.

such as BUTLER [5], COMPOSE [6], FIND [7], FIRE [8], and IoT-A [9]. They deal with large-scale networking, cognitive networking, network of networks [10], as well as service-oriented architecture (SOA) development for a converged communication and service infrastructure, to mention a few. In the light of growing IoT infrastructure, more number of sensors and actuators make the system more intelligent and highly responsive. Higher amount of sensed information and precise control could help in achieving sophistication. Furthermore, there must be many devices (to substitute) to make services fault-tolerant and dependable. However, *paripassu*, they also impart difficulties of maintaining them, controlling them, and filtering enormous amount of data generated by them.

In this light, we enlist some of the key factors that dictate the challenges in IoT related research.

1) *Heterogeneity*: IoT devices are deployed by different persons/authorities/entities. These devices have different operating conditions, functionalities, resolutions, etc. Thus, enabling seamless integration of these devices is a huge challenge. The degree of complexity increases many folds when some of these simple devices are merged to form a complex network.
2) *Scalability*: The rapid growth of embedded technologies is leading to enormous deployment of miniaturized devices (sensors, actuators, etc.). As the number of devices grows, the data produced by these devices grow unboundedly. Thus, handling the growth of number of devices and information they produce is a massive challenge in IoT.
3) *Interoperability*: In an IoT application, there are many actors comprising human and nonhuman objects. An actor can play multiple roles based on the current situations and environment such as available resources in the IoT application, data provider, data consumer, and service provider. Seamless interaction among the various actors is crucial to envisage the vision of IoT. The interaction among different objects magnifies, especially when each actor is managed differently.
4) *Security and Privacy*: Due to the large number and the heterogeneity of the actors involved in IoT, ensuring data authentication, data usage control, data consistency, and protection of data are a few core issues. To evolve a holistic system design, information security, privacy, and data protection need to be addressed properly.

A. Contributions

In this paper, we propose a layered and distributed architecture for IoT, called Distributed Internet-like Architecture for Things (DIAT). We believe that it has a potential to tackle many of the technical challenges described earlier. It also supports the desired characteristics of IoT objects and applications. Our key contributions are listed below.

1) We define a layered IoT architecture. Layering binds similar functionalities together and provides a hierarchical structure for the functionalities. It also provides decoupling of orthogonal features (e.g., security, privacy) and encourages their independent development.
2) We ensure integration of the desired characteristics of IoT systems with the help of various cognitive functionalities defined at various layers of the architecture. Specifically, we show how dynamic service creation and modeling can be achieved without having any human intervention.
3) We describe how an usage control mechanism can be used to support security and privacy in a distributed and dynamic environment where the context of the IoT objects can change.
4) With the help of a composite (cross application domain) use-case, we showcase that the proposed architecture (DIAT) has *distributiveness*. It is also capable of tackling various technical challenges such as heterogeneity, scalability, and interoperability.

This paper is organized as follows. First, we briefly discuss some of the existing IoT architectural design principals in Section II. In Section III, we describe our proposal of a layered architecture for IoT in detail. Then, we describe some of the key cognitive features of the architecture that provides the desired characteristics of IoT in Section IV. In Section V, we provide a use-case example to validate our proposal. Finally, we conclude this paper in Section VI.

II. Related Work

Many proposals attempt to define an architectural model for IoT that are usually applicable to a specific application domain. For example, Castellani *et al.* have proposed an architecture for a smart office application [11]. Their main focus is only to interconnect wireless sensor networks and actuator networks to the Internet as a web service. Similarly, the EPC global IoT architecture has mainly focused on the RFID network and smart logistics system [12]. There is no suitable unified architecture till date that is appropriate for a global IoT infrastructure. The existing architectural proposals are defined for a particular type of application. From a security perspective, these proposals list requirements for each type of application (e.g., identification and usage control), but do not propose or adopt specific security frameworks.

Many researchers have suggested layered architectures for IoT. For example, Gronbaek *et al.* [13] and Dai *et al.* [14] have proposed architectural model similar to the open systems interconnection (OSI) architecture. Tan *et al.* have also suggested a layered architecture for the IoT [15]. However, their design failed to emphasize how to tackle the key challenges in IoT such as interoperability, scalability, and security. There are other works that provide directions for the development of IoT architectures. Ning *et al.* have provided a comparative study between man-like neural system and social organization framework for IoT architecture development [16]. Kovatsch *et al.* have proposed a centralized web-resource-based architecture for decoupling the application development from the domain of heterogeneous devices [17]. However, these approaches neither guarantee scalability nor tackle interoperability of objects belonging to various application domains. Moreover, the security and privacy issues are briefly acknowledged in all these proposals—some authors consider security at the network level

TABLE I
COMPARISON AMONG ARCHITECTURAL PROPOSALS

IoT characteristics	DIAT	Butler [5]	Compose [6]	IoT-A [9]
Heterogeneity	Yes	Partially	Yes	Yes
Scalability	Yes	No	No	–
Interoperability	Yes	No	Partially	Yes
Security and privacy	Yes	Yes	Partially	Yes
Automation	Partially	Yes	Partially	Yes
Distributiveness	Yes	Partially	No	–
Layered design	Yes	No	Yes	No

only while others restrict themselves to mentioning security and privacy issues that should be considered in an IoT architecture.

From the IoT perspective, every object can be seen as a potential service provider. However, these tiny services might not provide a complete application/solution. IoT applications need to holistically combine such multiple tiny services to provide a complete functionality. They add up in right measures and sequence to provide a complete solution to users. In SOA, computers run an arbitrary number of services, and services can exchange information among themselves without human intervention and without the need to make changes to the underlying program itself. Thus, SOA approaches are considered to be applicable for IoT by many researchers [18], [19]. However, the architecture needs to be adopted in such a way that the technical challenges (e.g., interoperability, security, and privacy) related to IoT can be tackled.

Overall, there are many pieces of solutions to enable IoT for smart applications, but a comprehensive and holistic design is required. In this paper, we attempt to provide such a comprehensive design. We compare our proposal (DIAT) with some of the existing efforts based on potentiality of tackling technical challenges and availability of the key characteristics of IoT (Table I). From the table of comparison, it is evident that our proposal shows a great promise for a global IoT architecture.

III. IoT Distributed Architecture

We follow a layered architecture for the IoT infrastructure. In IoT applications, similar to a SOA, multiple services from individual service provider (or even individual objects) need to be merged with minimal human intervention. To fetch services from the objects without human interaction and to ensure easy collaboration among services, the objects need to expose their services in a homogeneous way [20]. Hence, an abstract representation of the objects along with their features and capabilities is needed. On top of that, smart collaboration and coordination are required among the set of services that can serve toward a common goal. To create and manage the services, various service requests need to be properly analyzed before the execution. Thus, the functionalities of IoT infrastructure are grouped into three layers, which are: 1) virtual object layer (VOL); 2) composite virtual object layer (CVOL); and 3) service layer (SL). These three layers are responsible for object virtualization, service composition and execution, and service creation and management, respectively.

To support some of the key features of IoT, such as automation and intelligence, a set of cognitive functionalities is integrated at each layer of the architecture. The three layers of the IoT architecture and their main functionalities are put together as a stack, called IoT Daemon (Fig. 1). The IoT Daemon forms the basis for the distributed architecture, which we call the DIAT. Please note that the lowest layer (VOL) is mainly responsible for virtual representation of objects and acts as an interpreter between the physical and the cyber worlds. Features such as management, coordination, and automation are implemented in the upper layers. Therefore, the cognitive functions as well as the resource requirement (e.g., processing power, and memory) increases from lower to higher layers. Here, we do not consider building physical communication among the devices since it has been dealt in depth in the literature.

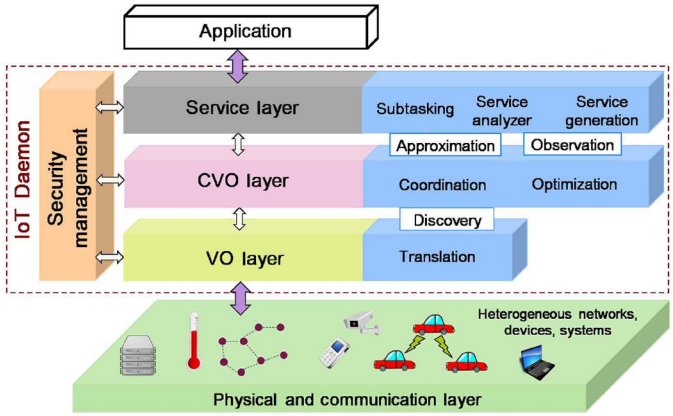

Fig. 1. Layered architecture for the IoT.

In the IoT Daemon, the management of security and privacy policies is performed by the security management (SM) cross-layer module. The SM is responsible for the evaluation of policies that should be enforced by the SL, CVOL, and VOL.

A. VOL

The VOL is responsible for virtualization of physical objects or entities. It hosts the virtual representations of the real-world (physical) objects, called virtual object (VO). A VO provides a semantic description of the capabilities and features of the associated real-world objects. Semantic modeling of VO enables efficient information exchange by expressing the information and its relationship among other VOs. Through semantic technologies, the VO layer provides universal methodologies to access all physical objects. Furthermore, the VOL plays the role of bridging the gap between the physical and the cyber world. As VOs are the digital representation of physical objects, the objects can be accessible in the cyber world through its corresponding VOs. Indeed, the VOs act as a translator between the digital and physical worlds. This abstraction helps to tackle the heterogeneity in various devices, systems, and networks and ensures interoperability and reusability of objects.

B. CVOL

An individual VO represents its corresponding physical object and it is constrained by the capability of that object. In

reality, multiple physical objects need to be engaged in order to accomplish a particular task. They need to communicate and coordinate among themselves to fulfill the goals. A CVO does this job. At runtime, a CVO is created by forming a mash-up of one/multiple VOs (and/or other existing CVOs) based on the necessity of a task. During the execution, a CVO dictates how the individual entities (VOs and/or CVOs) in its mash-up should work. A CVO plays the role of a coordinator. Additionally, it tries to optimize the operations among its entities by doing smart scheduling. The CVO layer (CVOL) hosts the CVOs along with managing their creation and execution.

Before forming a CVO, the CVOL has to identify a set of suitable VOs (and other existing CVOs) that can collaboratively accomplish a service request. Thus, a *discovery* mechanism is identified as one of the key features of the CVOL. Semantic description of a VO (CVO) can easily describe its features and capabilities, and helps in the discovery mechanism. Discovery mechanism is out of the scope of this paper; therefore, it is not discussed in a detailed manner.

C. SL

The SL is responsible for creation and management of services. On one hand, it handles various service requests from users. On the other hand, it initiates service requests on its own in order to enable automatic service creation. Additionally, whenever a service request is received, the SL analyzes and splits it into smaller subtasks. The SL also decides how these subtasks are assembled to reach the final goal. This (abstract) description of a decomposed service request is provided to the CVOL to execute the task. Using the discovery mechanism, the CVOL identifies the mash-up of VOs and CVOs. If an exact match for VO (CVO) is not found (according to the requirement), an alternative VO can be selected if it can fulfill the requirement partially. Similarly, an exact service match (as requested) might not be available always and a close alternative can serve the purpose. This paradigm is termed as *approximate service* [21], [22].

D. SM

To address the security and privacy challenge in the DIAT architecture (Fig. 1), a cross-layer SM module is defined. This module uses a more expressive security policy language for usage control, which supports event-based authorizations and obligations with temporal operators. The expressiveness of our language is a clear novelty in contrast to existing role-based or XAMCL [23] languages adopted by other IoT architectures [24].

The SM module supports modeling, specification, and enforcement of security policies for all the layers of the DIAT architecture. The modeling is done using the collection of interrelated metamodels to represent the data, identities, context, roles, structure, and behavior. The main goal of the SM is to control the usage of data, resources, and services of the IoT objects. Using the SM, it is also possible to model the trust-relationships and risk; but these modeling support is out of the scope of this paper. The collection of metamodels and runtime components used in the SM is called SecKit [25], and is described later in more details (Section IV-C).

E. IoT Daemon

All the proposed layers and their functionalities together is referred as IoT Daemon. As all the layers are associated with certain cognitive capabilities and SM features, the IoT daemon also consists of cognitive functionalities and SM. It acts as the basis of the distributed architecture in DIAT. The abstracted view of an IoT daemon is shown in Fig. 1. Every object with some processing power and memory runs its own IoT daemon. However, the footprint of the daemon varies with the capability of the devices.

A full-fledged IoT daemon contains three layers with all the functionalities. But some tiny embedded devices might lack processing power and memory to run a full version of the daemon, e.g., wireless sensors and actuators. As a result, a depreciated IoT daemon may run on those devices with limited set of functionalities. For example, a basic VOL with the capability of hosting at least one VO is executed in a wireless sensor node.

Using IoT Daemon, our proposed architecture tackles all the challenges in IoT mentioned previously. The VOs hosted by the VO layer are the abstract representation of the corresponding physical devices in the cyber/digital domain. Communication with the devices is performed through the VO layer, which hides heterogeneity of devices in terms of capabilities and features.

The VO layer of an IoT daemon hosts the VOs that are directly associated with it. As a result, centralized VO hosting is not required. The same applies to the upper layers as well. A VO can connect with any other VO or CVO and they may not be hosted on the same device. This helps to overcome the memory and other resource requirements of the hosting entities. Thus, DIAT is scalable.

The presence of IoT daemon in every object can relieve the burden of configuring each device manually, i.e., ensuring the scalability and zero-configuration requirement [4]. The APIs and interconnections among the three layers ensure interoperability. Furthermore, each IoT daemon runs its own SM components. The SM may allow deployment of security policies from other trusted IoT daemons. When needed, usage control policies may be deployed to protect the data transfer among various VOs, CVOs, and services running in different daemons. The security toolkit (SecKit) ensures security and privacy in DIAT.

To address a service request, often, data from multiple VOs are combined to draw a meaningful conclusion. These operational logics are applied at the higher layers (CVO and Service). Due to the layered design, lower layers of an IoT daemon residing in Device A, e.g., are able to communicate with upper layer of any other IoT daemon residing in Device B. As a result, all the functionalities need not be hosted in a centralized location. This depicts that the proposed architecture supports distributed operation among various IoT entities.

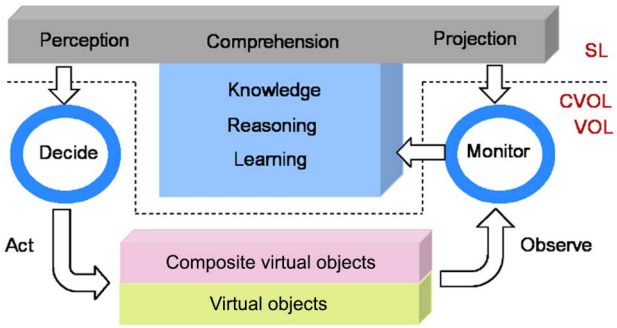

Fig. 2. Functionalities and workflow of an observer.

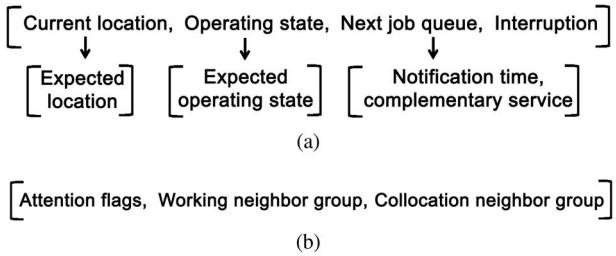

Fig. 3. Contextual information vector for a (a) human object and (b) nonhuman object.

IV. Cognitive Management in IoT Daemon

Smart features such as automation, intelligence, and dynamicity, are the inherent characteristics of the IoT ecosystem. To support these features, we envisage a set of cognitive functions at the each layer of the DIAT architecture. These functions can be developed independently and integrated into the IoT daemon. In this paper, we describe some of the key cognitive functionalities of the IoT daemon.

A. Dynamic Service Creation

One of the major tasks of the SL is automatic creation of services according to the context and situation. We define a cognitive entity, called the *observer* that plays a key role in automated machine-to-machine (M2M) communication in order to provide a service intelligently. The functionalities and the workflow of an observer are spread across the CVOL and SL (Fig. 2). The observer continuously monitors objects and assesses their situation. Based on available knowledge and current situation, it may decide to initiate a set of service requests without any human trigger.

Ubiquitous and pervasive computing is an inherent feature of IoT. Thus, context-awareness is an essential requirement for IoT. To initiate and execute an automated service intelligently, situations of all objects (humans, devices) need to be assessed carefully. Hence, context-awareness is an integrated part of the observer, which continuously monitors objects and derives the contextual information. The observer stores the contextual information about each object in its associated vector. Objects can be broadly categorized into two groups—human and nonhuman objects. Two different types of vectors are used for these two sets of objects. A contextual information vector for a human object contains the following fields [refer to Fig. 3(a)].

1) Current Location: This field contains the current location of the human being. This is not raw data (like GPS data), but high-level information derived using low-level data. For example, the current location can indicate *atHome, atOffice, atSupermarket*, etc. Additionally, this field has a subfield, called *expected location*, which is derived based on the prescheduled job list and previous observations. If expected location information is available and if it differs from the current location, the user will be notified or a new service request will be initiated. For example, if the current location is *atHospital* and expected location is *atOffice* due to a scheduled meeting, a service request is generated to inform all the interested party for cancellation and rescheduling of the meeting.

2) Operating State: The operating state indicates what a person is currently involved in. Examples of operating state can be *inMeeting, isWorking, watchingTV, isSleeping*, and *inHealthEmergency*. This helps to initiate new service requests. For example, if a person is in a meeting, any incoming call is automatically diverted to autoresponse mechanism or another person and the actual callee is notified in due time. Similar to the location field, this field also has a subfield, called *expected operating state*. Any conflict between the subfield and the main field initiates a new service request.

3) Next Job Queue: This field describes upcoming jobs that are derived from the to-do list or calendar events of a person. This field contains two subfields—*notification time* and *complementary service*. The notification time is the approximate time when an event should happen. The SL initiates a new service request at that moment. Additionally, the execution of a job might initiate some other jobs. The *observer* tries to determine those jobs and queue them in the list. The complementary service field points those jobs in the queue.

4) Interruption: Any service request (human-initiated or auto-generated) might require some human intervention at various points during the execution. The *observer* sets an interruption flag, if some service requires human attention. Based on the urgency level, a person is either notified immediately or it is postponed. The operating state of the person is important to decide the urgency level. For example, if a person is in a meeting and the central heating system of his/her house is broken, the person can be informed later. But if there is a burglary, the person should be informed immediately.

Similarly, the contextual information vector for a nonhuman object contains the fields as shown in Fig. 3(b).

1) Attention Flags: This field contains various flags that indicate something is unusual with the object and attention is needed. The attention may not be necessarily from human beings. It may be in consultation (communication) with other objects. The level of urgency decides how quickly the attention needs to be provided. For example, an *observer* identifies a fire from the data produced by the fire detector VO and sets the attention flag to a value so that an immediate fire alarm service request can be initiated. On the contrary, if a room heater stops working, the attention flag is set to a relatively low-priority value such that a service request is scheduled at an appropriate time.

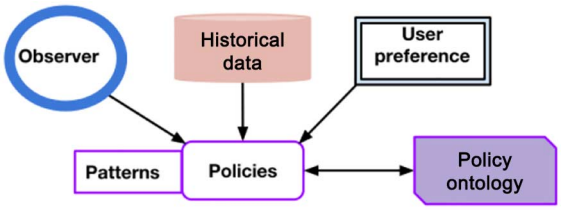

Fig. 4. Creation of policies in DIAT.

2) Working Neighbor Group: This field creates a group among similar objects within a small area. The definition of an area can vary in different scenarios. In case of climate controller in an office domain, the area is defined by the office premises (building). On the other hand, the area for a traffic sensor can spread across couple of blocks. The group members can communicate with each other for coordination and optimize their performance. An Observer can detect an anomaly in one object by consulting with other group members and set the attention flag.

3) Collocation Neighbor Group: Similar to the working neighbor group, this field creates and monitors a group among the neighboring objects. These objects need not be of the same type, but they are grouped based on their geographical collocation.

B. Dynamic Service Modeling

In the previous section, we described observer—a key entity to monitor objects and initiate a service request based on the context and situation. Execution of these services in real time is a challenge in dynamic environments. As the CVOs are the service executors, to support dynamic service execution, we propose "policy-based models" for dynamic CVO creation in near real time. Policies are an effective way to adapt to the dynamic environment and changing user requirements. Policies handle macro level specifications of service request and are dynamically created based on the inputs from observer, historical information, and user requirements (Fig. 4).

In this work, policies are modeled using semantic technologies namely, Ontology to describe in a machine-interpretable and interoperable manner. Policies represent a structured way to determine the CVOs required to accomplish a service request. A generic policy tuple consists of policy id, modality, trigger, subject, target, behavior, constraint, role, desires, intentions, and assignment. *Modality* defines the authorization and obligation of the VO's. *Trigger* defines the state of the VO's or the time events. *Subject and target* define the roles of VO's in the system. *Behavior* defines the overall goal of the policy that is depicted by the CVO. *Constraint* specifies condition under which the policy is applicable. *Role* indicates the set of roles required to realize the goals. *Desire* is the set of subgoals obtained upon on service decomposition. *Intention* specifies the execution manner of subgoals; plans are updated dynamically based on availability of VO's and knowledge gained over time. *Assignment* is a dynamic mapping mechanism that assigns a role to a particular VO whose behavior matches to the goal of the policy. Further details on policy ontology is described in [20].

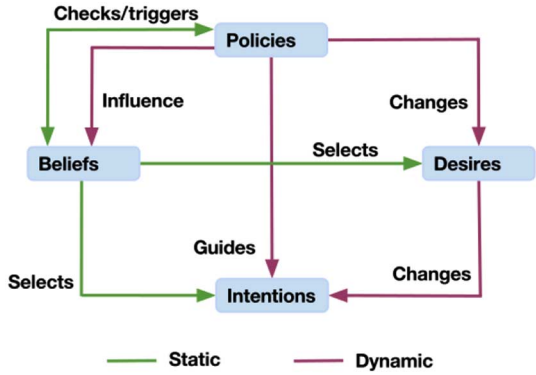

Fig. 5. BDIP model.

Policies are categorized into three groups in accordance with the three levels of the DIAT architecture.

1) *High-level policy (SL level)*: Service level policy determines the high-level abstract from the service request. This policy handles the macrolevel specification described by the service request.
2) *Concrete policy (CVO level)*: The CVO level policy determines the subgoals and roles of the VOs using inputs from observers and policy ontology. Concrete policies define how to create a CVO to accomplish subgoals in order to complete the service request.
3) *Low-level policy (VO level)*: VO level policy contains the implementation details in terms of execution plan. VO policies determine what is the role of each VOs in the system.

In this work, we extend belief–desire–intention model [26], to determine the policies for dynamic service execution. BDI model is extensively used in intelligent agents to model complex and dynamic systems. To enable dynamic service execution and CVO creation, we develop belief–desire–intention–policy (BDIP) model. The major entities in BDIP model are as follows.

1) *Belief*: What the IoT daemon believes about a particular service request. This refers to the initial belief of what the CVO is supposed to do to accomplish the service request.
2) *Desire*: What are the goals that need to be achieved by the CVO. This describes what the CVO wants to achieve based on the initial beliefs derived.
3) *Intention*: What is the plan or sequence of action that need to be done to achieve the goal.
4) *Policy*: It guides the intentions, i.e., how to execute the plan of action and the roles of each CVO and VO.

Fig. 5 shows the interactions among the entities in BDIP model. To accomplish a service in static environment, we first determine the initial belief, based on the current service request and historical requests. Based on the initial belief, the BDIP model updates the desires of the CVO and determines the sequence of plan to complete the service. Observer monitors the VOs and creates necessary service requests in order to adapt to the dynamic environment. The proposed BDIP model can support dynamic service execution and CVO creation process with the help of policies. Upon creation of new service requests, the policies influence the initial belief of the CVO to accomplish

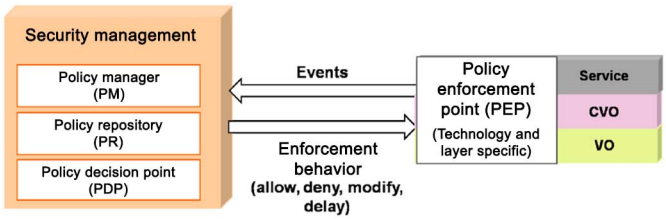

Fig. 6. Security management.

the new service requests. Policies also change the desires of the CVO and intention of the service requests based on the new service requirements and the dynamic environment. Fig. 5 shows sequence of interactions in the BDIP model based on static and dynamic service requests. Thus, our proposed BDIP model can adapt to the dynamic environment and user requirements making our architecture more scalable and robust.

C. SecKit: SM Module

The IoT daemon, as mentioned earlier, defines components for management of security and privacy in the SM module. This cross-layer function is one of the key cognitive features in our DIAT architecture. The SM module consists of three components, which are part of the SecKit tool: 1) policy manager (PM); 2) policy repository (PR); and 3) policy decision point (PDP) (Fig. 6). The PM component is responsible for retrieving the rules of the security policy stored in the PR and deploying them in the PDP. The PDP instantiates the policy rules and subscribes to the events in the policy enforcement points (PEPs) deployed at different layers of the DIAT architecture. The signaling of the events is represented in Fig. 1 by the arrows between the SM and the three layers in DIAT: Service, CVO, and VO. The SM module also provides an interface for receiving policies from other domains running the DIAT architecture.

The PEP component is layer and technology specific, and is responsible for intercepting specific activities at the different layers and notifying events to the PDP. On the other hand, the PDP, which is technology independent, receives the events and evaluates the deployed policy rules. If some specific policy rule is triggered, the resulting enforcement behavior, is send to the PEP. In our policy language, we support enforcement behaviors to *allow*, *deny*, *modify*, or *delay* an activity. In addition to the enforcement behaviors, the PDP may also execute additional activities, e.g., to notify users or to execute compensation actions if needed. In this way, the SM addresses the interoperability and heterogeneity challenge in DIAT since the PDP remain the same while the PEP is customized with new technologies/devices without significant changes.

The security policy rules are specified using templates that can be parametrized and reused. These templates follow an event condition action (ECA) structure, when the trigger event (E) is observed and the condition (C) evaluates to true, the action (A) is executed. The event part of a rule template is called the *trigger event*, and is an event pattern matching operator that can also be used in the condition part. The adoption of an event-based paradigm can support a distributed IoT architecture. Events represent the execution of actions and interactions among VOs, CVOs, and services in DIAT.

Our performance results presented in [25], [27], and [28] indicate that we can address the scalability issue efficiently with SecKit. The PDP engine is capable of handling more than 200 thousand events in less than a second [25]. One of our technology specific enforcement mechanisms for VOs implemented at the Message Queuing Telemetry Transport (MQTT) broker does not introduce more than 10 ms of additional delay on the average to deliver messages [27].

In SecKit, activities are represented by events. These events are associated with the modalities: *start*, *ongoing*, and *completed*. To support enforcement of usage control policies including authorization, we also model *tentative* and *actual* events. A tentative event is generated when an activity is ready in the respective DIAT layer but has not yet started, giving the opportunity for the execution of the enforcement behaviors. The event format includes the details about the activity being executed such as identity and roles of the entities performing the activity and information attributes of the activity itself.

The action part of an enforcement template includes an enforcement and an execution part. The enforcement part refers to the trigger event of the rule template (E), and may allow or deny the execution of the respective activity. If the activity is allowed, it is also possible to specify an optional modification or delay of the activity execution, e.g., anonymizing activity data before the activity takes place. The execution part of an enforcement template may trigger the execution of additional activities, e.g., notifications or logging of information. The condition part of a rule template consists of event patterns, context patterns, role patterns, and trust patterns. The different patterns can be recursively composed using propositional, temporal, and cardinality operators allowing the specification of complex conditions [28].

Our approach for security and privacy supports the specification of security policies to address many different requirements in an IoT system. Authorization and obligation policies can be specified to implement anonymization of data, context-based-authorization and data protection, consent policies, integrity policies, nonrepudiation, and trust management. We support the specification of nested policy rules to overlap combining algorithms that can be configured to decide the priority. For example, the first applicable rule should be chosen in case conflicting rules are triggered.

V. A Multiapplication Use-Case

To validate our architecture, we describe a composite use-case in this section. The example consists of various IoT applications such as smart home, smart transportation, and smart health-care, as shown in the Fig. 7. The proposed architecture overcomes the barrier of (so-called) separate IoT applications and achieves a global *IoT* via interoperability. We assume an IoT daemon runs on every object, belonging to various application domains, and they communicate among themselves to carry out a task.

In the example shown in Fig. 7, a smart fridge hosts a simple IoT daemon, which has only the VOL. The IoT daemon

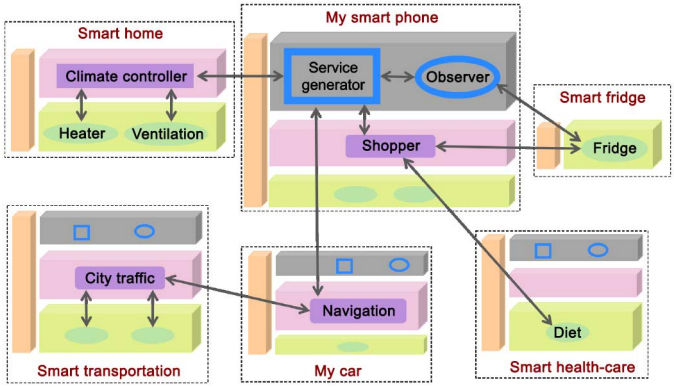

Fig. 7. Use-case showing the unification of various applications based on the distributed IoT architecture.

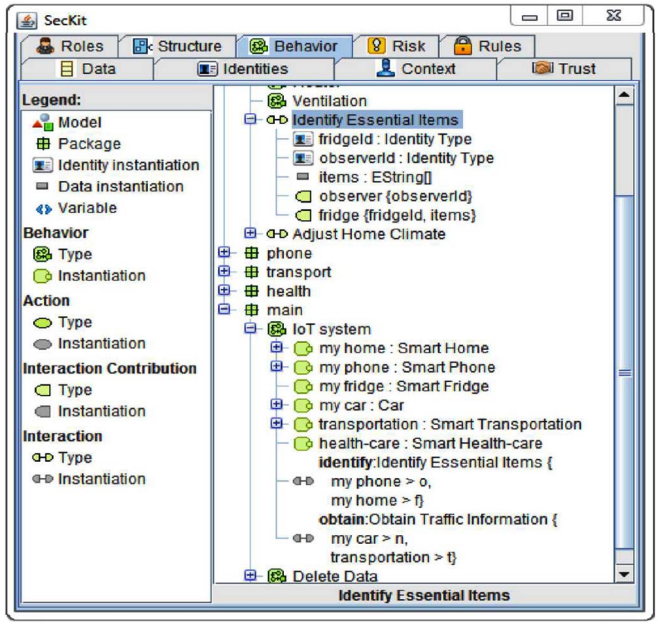

Fig. 8. Behavior design model GUI.

hosts a *fridge* VO, and it can inform a legitimate user (e.g., the owner) about its content. An *observer* of the fridge is hosted in a more capable pairing device, e.g., a PC, and a smart phone. The *observer* can identify essential items that are not available in the fridge. Let us assume that the *observer* is in the smart-phone of the fridge owner and it may add a task in the "next job queue" if some (essential) items are missing from the fridge. When the owner leaves his/her office in the evening, the *observer* identifies this by analyzing the contextual information vector associated with the person (current location and operating state). It also fetches the scheduled tasks from the job queue and creates a service request through the service generator. The service request forms a CVO, called *shopper*, using the BDIP model by engaging two VOs—the *fridge* VO and the *diet chart* VO. The *diet chart* VO is maintained at the "smart health care" domain. It is created based on a doctor's and/or a dietitian's advice. The *shopper* CVO also finds a convenient place (supermarket) to buy the items.

From the contextual information vector, the *observer* also identifies that the person is driving currently. So the *service generator* informs the *navigation* CVO, residing in the car's IoT daemon, to fetch the route information to the nearest (or on the way to home) store. Then, the *navigation CVO* gathers the information about the shortest route and an empty parking place from the *city traffic* CVO of the "smart transportation system." The *city traffic* CVO collects these information after consulting with the appropriate traffic sensors deployed across the city (through their corresponding VOs). Finally, the store is selected based on traffic condition, parking information, and distance. Additionally, the *service generator* estimates the time to reach home. It conveys this information to the "smart home" climate control application. The *climate controller* CVO then engages the *heater* and the *ventilation* VOs to make the indoor climate as comfortable as possible according to the preference of the owner. The BDIP model guides the whole process starting from the service request initiation. It makes decision about the roles of each VOs and models the CVOs accordingly if certain changes occur during the execution of the service.

From the use-case as shown in Fig. 7, it is clear that the VOs, CVOs, and service logics are not hosted centrally. Rather, they are hosted locally by the local IoT daemon based on direct association. This makes the architecture scalable. Moreover, use-case shows a local CVO can combine data from the local VOs ("climate controller" CVO in smart home) or a remote CVO can also combine the data collected from VOs hosted by different IoT daemon ("shopper" CVO in my smart phone). This depicts that the proposed architecture supports distributed operation among various IoT entities.

Fig. 8 shows the design of our use-case behavior model using the SecKit graphical user interface (GUI), which is aimed for IoT application designers with knowledge of the modeling language. In the model, behaviors of a CVO, contained VOs, interactions, data, and identity instantiations are specified. In this figure, we have highlighted the *identify essential items* interaction type, which has two types of contributions: *observer* and *fridge*. The observer provides his identity information (*observerId*) and receives from the fridge the list of essential items (*items*) along with its identity information (*fridgeId*). This interaction is instantiated in the *IoT system* behavior, where the behavior instantiations in *my home* and *my phone* interact with their respective roles. The behavior design model supports the design of Services, CVOs, and VOs using a common metamodel, and is used as a reference for the specification of the security policy rules. The behavior design models depict how the executable artifacts deployed by the administrators of each application domain interact with each other. They are not used for runtime coordination.

The definition of the appropriate policies and the governance of the policy is an important aspect of the overall framework. The proposed approach is that the specific entities with knowledge of the regulatory framework and the domains will create a set of predefined policies for generic profiles of users, which could be accustomed by users with appropriate capabilities. The approach is similar to security configurations in personal computers where the user has the option to adopt a recommended and already defined security configuration or to create a new one. The difference is that policies may be more sophisticated

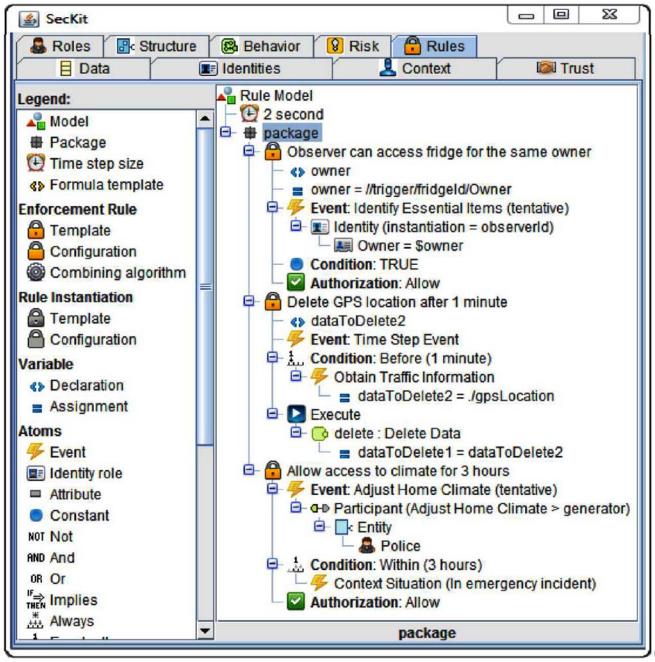

Fig. 9. Policy design model GUI.

than the security configurations of a personal computer and they may be more complex depending on the dynamically changing context from where the user operates. As a consequence, it is important to define governance processes for the life-cycle of policies (installation, deployment, secure distribution, end of life) and related roles, which are specified below.

1) The policy engineer is responsible for defining the policies for different class of users' profiles and for a specific domain. For example, in the domain of an intelligent transport system (ITS), a specific policy can be defined for generic drivers, another policy for law enforcers, and another policy for commercial drivers (e.g., truck drivers). The number and types of profiles can be defined by the policy engineer.
2) The policy administrator is responsible for the lifecycle of the policy and the definition of the processes for initial installation of the policies. He is also responsible for the distribution of new policies and their removal when the policies are obsolete.
3) The policy certification engineer is responsible for the certification of the policies for a target environment and devices (e.g., boundary conditions and constraints). Policies may be recertified if they are modified after the initial certification.
4) Policy user is the user of the policy framework, who can choose to adopt the recommended policy for his/her profile or customize it. The policy engineer may decide that only a well defined part of the policy can be tailored and not the core part, or he may decide that policy must be recertified to be used.

Fig. 9 shows three examples of the security policy templates. We have specified for our use-case in order to illustrate the expressiveness of our policy language. These policies illustrate an access control rule, an obligation with temporal constraints, a context-based access control rule [29], [30], and parametrization using variables. Our policy language is very flexible and allows the combination of these different elements, e.g., to specify temporal, cardinality, and context-based authorizations and obligations. In this example, we adopt a white-listing approach, meaning only explicitly allowed activities can be executed and all other activities are denied. Using our policy language, we can also specify black-listing or a combination of both using nested policy rules.

The first rule template in Fig. 9 specifies that the interaction of type *identify essential items* is only allowed to be executed if the observer and the fridge behaviors have the same value for the *owner* identity attribute. The second rule template specifies that 1 min after the traffic information is obtained, the global positioning satellite (GPS) location information exchanged should be deleted. It is possible to support privacy regulations by adopting specific policies in the IoT objects for the usage of stored data, for filtering outgoing data from IoT objects or to implement the "right to be forgotten." In addition, SecKit can support changes in the context due to exceptional circumstances like an emergency crisis. The third rule template specifies that 3 h after the start of an emergency incident situation, the police (associated to a specific role in SecKit) should be allowed access to the *adjust home climate* interaction.

In order to be useful in concrete IoT scenarios, PEPs must be extended with technology specific runtime monitoring components. We have implemented a PEP to support monitoring and enforcement of policies for an MQTT broker, which is a middleware technology that can be used in a concrete DIAT deployment to support the multidomain communication among Services, CVOs, and VOs. The implementation of the PEP in a middleware component has performance benefits since the resource constrained IoT nodes do not need to perform signaling of events and execution of enforcement actions.

VI. Conclusion

The foundations for a generic IoT architecture have been discussed in this paper. Based on these foundations, we have proposed a distributed layered architecture for the IoT, called DIAT. DIAT is a simple, scalable architecture for the IoT. It accommodates heterogeneous objects and provide support for interoperability. In our proposed IoT architecture, we also implement security and privacy aspects using usage control policies. Features such as automation, intelligence, dynamicity, and zero configuration are integral part of DIAT. Several cognitive functions such as dynamic service creation, modeling and execution are incorporated in the proposed architecture. We have shown that DIAT satisfies the key characteristics and goals of an IoT architecture through the study of a realistic use-case. The basic concepts of a futuristic architecture is being shaped in an on-going EU project, called iCore [31].

Acknowledgment

The research leading to these results has been performed within the iCore project. The authors also thank partners from iCore who gave valuable suggestions to improve this paper.

REFERENCES

[1] M. Weiser, "The computer for the Twenty-First Century," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 3, no. 3, pp. 3–11, Jul. 1999.

[2] L. Atzori, A. Iera, and G. Morabito, "SIoT: Giving a social structure to the Internet of Things," *IEEE Commun. Lett.*, vol. 15, no. 11, pp. 1193–1195, Nov. 2011.

[3] G. M. Lee and J. Y. Kim, "The Internet of Things: A problem statement," in *Proc. Int. Conf. Inf. Commun. Technol. Convergence (ICTC)*, 2010, pp. 517–518.

[4] N. I. C. Wangi, R. V. Prasad, M. Jacobsson, and I. Niemegeers, "Address autoconfiguration in wireless ad hoc networks: Protocols and techniques," *IEEE Wireless Commun.*, vol. 15, no. 1, pp. 70–80, Feb. 2008.

[5] BUTLER [Online]. Available: http://www.iot-butler.eu/

[6] COMPOSE [Online]. Available: http://www.compose-project.eu

[7] FIND [Online]. Available: http://www.nets-find.net/

[8] FIRE [Online]. Available: http://www.ict-fire.eu/

[9] IoT-A [Online]. Available: http://www.iot-a.eu/

[10] J. Hoebeke *et al.*, "Personal network federations," in *Proc. 15th IST Mobile Summit*, Myconos, Greece, 2006, p. 6.

[11] A. P. Castellani *et al.*, "Architecture and protocols for the Internet of Things: A case study," in *Proc. 8th IEEE Int. Conf. Pervasive Comput. Commun. Workshops (PERCOM)*, 2010, pp. 678–683.

[12] S. Beier, T. Grandison, K. Kailing, and R. Rantzau, "Discovery services-enabling RFID traceability in EPCglobal networks," in *Proc. Int. Conf. Manage. Data (COMAD)*, 2006, pp. 214–217.

[13] I. Gronbaek, "Architecture for the Internet of Things (IoT): API and interconnect," in *Proc. 2nd Int. Conf. Sensor Technol. Appl. (SENSORCOMM'08)*, 2008, pp. 802–807.

[14] G. Dai and Y. Wang, "Design on architecture of Internet of Things," in *Advances in Computer Science and Information Engineering*. New York, NY, USA: Springer, 2012, pp. 1–7.

[15] L. Tan and N. Wang, "Future Internet: The Internet of Things," in *Proc. 3rd Int. Conf. Adv. Comput. Theory Eng. (ICACTE)*, 2010, vol. 5, pp. V5-376–V5-380.

[16] H. Ning and Z. Wang, "Future Internet of Things architecture: Like mankind neural system or social organization framework?" *IEEE Commun. Lett.*, vol. 15, no. 4, pp. 461–463, Apr. 2011.

[17] M. Kovatsch, S. Mayer, and B. Ostermaier, "Moving application logic from the firmware to the cloud: Towards the thin server architecture for the Internet of Things," in *Proc. 6th Int. Conf. Innovative Mobile Internet Serv. Ubiquitous Comput. (IMIS)*, 2012, pp. 751–756.

[18] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, "Interacting with the SOA-based Internet of Things: Discovery, query, selection, and on-demand provisioning of web services," *IEEE Trans. Serv. Comput.*, vol. 3, no. 3, pp. 223–235, Sep. 2010.

[19] P. Spiess *et al.*, "SOA-based integration of the Internet of Things in enterprise services," in *Proc. IEEE Int. Conf. Web Serv. (ICWS'09)*, 2009, pp. 968–975.

[20] A. U. Nambi, C. Sarkar, R. V. Prasad, and A. Rahim, "A unified semantic knowledge base for IoT," in *Proc. IEEE World Forum Internet Things (WF-IoT)*, Mar. 2014, pp. 575–580.

[21] C. Sarkar, V. S. Rao, R. V. Prasad, A. Rahim, and I. Niemegeers, "A distributed model for approximate service provisioning in Internet of Things," in *Proc. Int. Workshop Self-Aware Internet Things*, 2012, pp. 31–36.

[22] R. V. Prasad, C. Sarkar, V. S. Rao, A. R. Biswas, and I. Niemegeers, "Opportunistic service provisioning in the future Internet using cognitive service approximation," *Proc. 28th World Wireless Res. Forum Meeting (WWRF)*, Athens, Greece, 2012.

[23] Rissanen, "Extensible access control markup language v3.0," *XACML*, 2010.

[24] S. Gusmeroli, S. Piccione, and D. Rotondi, "A capability-based security approach to manage access control in the Internet of Things," *Math. Comput. Modell.*, vol. 58, no. 5, p. 1189, 2013.

[25] R. Neisse *et al.*, "A model-based security toolkit for the Internet of Things," *Proc. 9th Int. Conf. Avail. Rel. Secur. (ARES)*, 2014, pp. 78–87.

[26] A. S. Rao *et al.*, "BDI agents: From theory to practice," in *Proc. Int. Conf. Multiagent Syst. (ICMAS)*, 1995, vol. 95, pp. 312–319.

[27] R. Neisse, G. Steri, and G. Baldini, "Enforcement of security policy rules for the Internet of Things," in *Proc. 10th IEEE Conf. Wireless Mobile Comput. Netw. Commun. (WiMob)*, 2014, pp. 165–172.

[28] R. Neisse and J. Doerr, "Model-based specification and refinement of usage control policies," *Proc. 11th Annu. Int. Conf. Privacy Secur. Trust (PST)*, 2013, pp. 169–176.

[29] R. Neisse, P. D. Costa, M. Wegdam, and M. van Sinderen, "An information model and architecture for context-aware management domains," in *Proc. IEEE Workshop Policies Distrib. Syst. Netw. (POLICY)*, 2008, pp. 162–169.

[30] R. Neisse, "Trust and privacy management support for context-aware service platforms," Ph.D. dissertation, Centre Telemat. Inf. Technol. Univ. Twente, Enschede, The Netherlands, 2012.

[31] iCore: Empowering IoT Through Cognitive Technologies [Online]. Available: http://www.iot-icore.eu/

Chayan Sarkar (GSM'12) received the B.E. degree from Jadavpur University, Kolkata, India, in 2009, and the M.Tech. degree from the Indian Institute of Technology Bombay, India, in 2011, both in computer science and engineering, and is currently working toward the Ph.D. degree at Delft University of Technology, Delft, The Netherlands.

He is currently with the Embedded Software Group, Delft University of Technology. His research interests include wireless sensor network, smartphone sensing, and the Internet of Things.

Akshay Uttama Nambi S. N. (GSM'14) received the B.E. degree in computer science and engineering from Visvesvaraya Technological University, Belgaum, India, and is currently working toward the Ph.D. degree at Delft University of Technology, Delft, The Netherlands.

He is currently with the Embedded Software Group, Delft University of Technology. Prior to this, he was a Research Assistant with the Ecole Polytechnique Federale de Lausanne (EPFL), Lausanne, Switzerland, and the Indian Institute of Science (IISc), Bangalore, India. His research interests include cyber physical systems, smart grids, energy harvesting systems, and the Internet of Things.

R. Venkatesha Prasad (M'06–SM'13) received the B.E. degree in electronics and communication engineering and M.Tech. degree in industrial electronics from the University of Mysore, Mysore, India, in 1991 and 1994, respectively, and the Ph.D. degree from the Indian Institute of Science, Bangalore, India, in 2003.

He has been a Consultant with the ERNET Laboratory, Indian Institute of Science (IISc). From 1999 to 2003, he was also a Consultant with CEDT, IISc, involved with VoIP application developments. In 2003, he was a Team Leader with Esqube Communication Solutions Pvt Ltd., Bangalore, India, where he was involved in the development of various real-time networking applications. From 2005 to 2012, he was a Senior Researcher, and since 2012, he has been an Assistant Professor with the Delft University of Technology, Delft, The Netherlands.

Dr. Prasad is a Senior Member of the ACM.

Abdur Rahim (S'11–GSM'11–M'13) received the Master's degree in electronics from the University of Gavel, Gavle, Sweden, and the Ph.D. degree in electrical and electronics from the Technical University Dresden, Dresden, Germany.

He possesses over ten years of experience working/managing European Union (EU) projects, namely, PULSERSII, EUWB, C2Power, iCore, i-locate, etc. He is currently a Project Manager of the EU FP7 IP project on Internet of Things (IoT) and Technical Manager of Intelligent Knowledge as a Service (iKaaS), IoT, Big Data, and Cloud integration projects. He is working as collaborate with Create-Net, Trento, Italy, where he served as a Technical Leader of the Smart IoT Group, focusing on IoT research.

Ricardo Neisse received the Ph.D. degree in computer science from the University of Twente, Enschede, The Netherlands, in 2012.

Since 2013, he has been a Scientific/Technical Project Officer with the European Commission Joint Research Center (JRC), Ispra, Italy. His research interests include security engineering for the Internet of Things and enterprise systems.

Gianmarco Baldini (M'09) received the Electronic Engineering degree (with a specialization in wireless communications) from the University of Rome "La Sapienza," Rome, Italy, in 1993.

He was a Senior Technical Architect and System Engineering Manager with Ericsson, Lucent Technologies, Hughes Network Systems, and Selex Communications before joining the Joint Research Centre of the European Commission, Ispra, Italy, in 2007, as a Scientific Officer. He has coauthored more than 40 papers on wireless communications and security topics. His research interests include communication services and security for intelligent transport systems (ITS) and Internet of Things.