The Importance of Being Thing
Or the Trivial Role of Powering Serious IoT Scenarios

Sumi Helal
School of Computing and Comm.
Lancaster University
InfoLab21, Lancaster LA1 4WA, UK
sumi.helal@ieee.org

Ahmed Khaled
Computer Science Dept.
Northeastern Illinois University
Chicago, Illinois 60625, USA
aekhaled@neiu.edu

Wyatt Lindquist
School of Computing and Comm.
Lancaster University
InfoLab21, Lancaster LA1 4WA, UK
w.lindquist@lancaster.ac.uk

***Abstract* – In this article, we call for a "Walk Before You Run" adjustment in the Internet-of-Things (IoT) research and development exercise. Without first settling the quest for what thing is or could be or do, we run the risk of presumptuous visions, or hypes, that can only fail the realities and limits of what is actually possible, leading to customers and consumers confusion as well as market hesitations. Specifically, without a carefully-designed Thing architecture in place, it will be very difficult to find the "magic" we are so addicted and accustomed to – programming! Programming the IoT, as we once programmed the mainframe, the workstation, the PC and the mobile devices, is the natural way to realize a fancy IoT scenario or an application. Without Thing architectures and their enablement of new programming models for IoT – we will continue to only envision fancy scenarios but unable to unleash the IoT full potential. This article raises these concerns and provides a view into the future by first looking back into our short history of pervasive computing. The article focuses on the domain of "Personal" IoT and will address key new requirements for such Thing architecture. Also, practicing what we preach, we present our ongoing efforts on the Atlas Thing Architecture showing how it supports a variety of thing notions, and how it enables novel models for programmability.**

Index Terms –Internet of Things, Thing Architectures, Pervasive Computing, IoT Programming Models, Mobile Apps as IoT Things.

I. Introduction

The success of the Internet of Things (IoT) will largely depend on how its main ingredient—the thing—is architected and prepared to match expectations and to fulfill the big heroic role that will magically make blue-sky visions a reality. Unfortunately, it seems we have not focused adequately or broadly enough on the architectural aspects of *things* in our pervasive computing journey. Explicit architectures addressing vision-enabling requirements need to be arrived at before we can harness the full potential of IoT. The highly abstracted notion of *things*, intentionally embedded within the IoT concept to keep a focus on the potential end-benefits, could perhaps explain the delay in paying attention to the architectural details of the *things* and the complexities inherent in their wide varieties and fragmented nature. Also, advances in low-power communications and light-weight networking protocols helped steer the initial focus to the internetworking aspects of IoT, leaving *things* themselves for later. Now seems to be the right time to focus on *things* themselves.

In this article, we present a brief history of *things* evolution and provide an outlook for the critical research and development needed to enhance and accelerate their preparedness to engage in and power the IoT. We use *things*, *devices,* and *smart objects* interchangeably in this article to refer to *things* in the IoT ecosystem.

II. Thing Evolution: Back to the Future

Things have evolved considerably in the past 20 years, even though under different names, shapes and forms. Instrumenting devices and everyday objects so that they become "digital" *things* utilizable in a pervasive system was the first step in this evolution. Devices and objects were "smartened" by connecting them to a variety of microcontroller boards (also known as sensor/actuator platforms) such as the TINI Internet interface [26], the Mote family [4], the Smart-It board [1] and the Atlas platform [2][3], among several others. Simply attaching tags (e.g., RFID or QR codes) was another approach to create digital *things* even though tagging required the cooperation of other companion *things*—the tag readers or the cameras. These early *devices* helped wireless sensor networks (WSN) [11][12] to evolve from theory to practice making it easy to use physical nodes to demonstrate application scenarios. The opposite was also true where WSN research helped accelerate the interconnectivity of these early digital *things* supporting peer-to-peer (ad hoc) and infrastructure communication modes.

Integrating *smart objects* effortlessly into systems was one main goal of these early pervasive computing developments. The goal was more about avoiding ad hoc system integration which was a non-replicable, rigid and inflexible approach to sustain. Operating systems and middleware were developed to enable the effortless integration of these early digital *things*. TinyOS was one of the first open source operating systems developed for *things* initially supporting the mote hardware platform. Gaia [27], Pico [28], Aura and Atlas [29] were among the early middleware developed to enable the promise of effortless integration [30].

Programming *things* flexibly into applications was a subsequent and complimentary goal towards effortless integration which required changing the actors' roles. Making *smart objects* programmable empowered ordinary software developers with no hardware or *thing* knowledge to

create pervasive and ubiquitous applications using then *de jour* programming models (object-oriented and service-oriented models). Below, we zoom in a little on the integration and programmability aspects in the journey of *thing* evolution [16][17][18].

A. *Effortless Thing Integration*

Human- and machine-readable descriptive languages such as IEEE 1451 [10], SensorML [9], ECHONET [8], IBM's Device Kit [25] and Atlas' DDL [13][14] eased *thing* integration problems by shifting quite a bit of the responsibility and role of integration from the traditional system integrator engineers, to the original equipment manufacturers (OEM). This approach was similar to, but much lighter in weight than, the universal plug and play (UPnP) standard. The main idea was for a *thing* OEM to provide a human- and machine-readable description for its *thing* that can be used by configuration tools or run-time middleware to generate an integration point out of the *thing* description, automatically.

Different integration points were developed, including language-specific APIs, sandbox services (e.g., Open Services Gateway Initiative (OSGi) service bundles [7]) and web-based RESTful services. The descriptive approach proved to be productive and continues to live on to date. Google's recent *Android Thing* development (an IoT operating system based on Android) features a similar declarative core service called *Weave* which utilizes community-developed declarative schemas offering functional and semantic descriptions of devices and *things*.

B. *Programmable Things*

Programming models also evolved as systematic *thing* integration was eased and facilitated by middleware and tool developments. Several models of overlapping types emerged, but we focus here on two key models: context-based and service-oriented. The former provided for a first successful approach to developing pervasive applications programmatically. A context was defined as an explicit specific state over a collection of *things*, or an expression that yields the state of a collection of *devices* when evaluated. Developing context-driven applications amounted to modeling, acquiring and reasoning about/reacting to such contexts. Context-driven programming models were a great success, even though they were limited in their expressive power, ability to model complex applications and capacity to alter or "actuate" contexts.

Service-oriented architecture (SOA) [22][23] models were also successful and provided for a more expressive alternative, equally powerful in both context sensing and context actuation. In this model, devices are integrated directly into services, forcing service composition as the model for programmability of pervasive applications. The service-oriented device architecture (SODA) [24] was a successful model developed jointly by IBM and the Atlas project in which *devices* were integrated into service bundles within OSGi, which is a Java sandbox supporting service activation, discovery, invocation and life cycle management. *Things* are basic services in this model, whereas applications are composed of basic and other composite services. Creating a SODA service for a *thing* had the effect of "*integrate once, program everywhere*." SODA successfully made it to some commercial products in the form of personal health devices that follow the Continua Alliance reference architecture [34], which was initially based on SODA.

SOA has undoubtedly been a powerful programming model for pervasive computing, but as it was put to practical use in smart space applications (e.g., smart homes), it become gradually evident that it was perhaps too powerful to be safe. Conflicting or even harmful combinations of basic or composite services could yield unsafe applications with the potential of causing harm to the space user, physical damage to the *devices* themselves, or both. The unconstrained actuations that could result from direct service invocations were lacking numerous safeguards. For example, there was nothing to restrain a buggy application code that erroneously invokes an automatic door opener 50,000 times per second in an infinite loop. The strike mechanism of such a door opener would jam up and its circuit eventually damaged. SODA also overpromised and supplied "fake convenience" to the programmers by providing the same kind of interface often used for services hosted on powerful servers, to any*thing*, including unreliable pinhead devices. Presenting ants as elephants was a major overpromise with safety, reliability and availability concerns. Initial promising solutions to address these concerns have recently been proposed including virtual *things* [35] in which multiple identical or equivalent *things* are utilized into a single virtual thing with superior safety, reliability, and availability.

The SOA approach also proved to be very productive and continues to live on to date. In fact, lighter weight versions of SODA exist today such as RESTful web services, often referred to simply as device API's. The ARM Mbed ecosystem, for instance, allows embedded application code, developed and loaded into an ARM Mbed board running Mbed OS, to be externalized and "API'ed" as a "cloud device service" which is an HTTP-REST service hosted in the Mbed cloud [31][32].

Service architectures like Service Oriented Architecture (SOA), Resource Oriented Architecture (ROA) [6] and Event Driven Architecture (EDA) [5] are considered as frameworks to program the smart space through the explicit trigger of the available services provided by the *devices* and their appropriate APIs. The social IoT (SIoT), on the other side, is considered as a different way of organizing the services provided by the *things* and discover opportunities for smart space programmability. The recently proposed ideas on social IoT are to logically link the *things* according to their identification attributes (e.g., *things* collocated in same smart space, *things* from same vendor) and to exploit the service-level relationships that logically and functionally tie the offered services to the smart space (e.g., the weather forecast readings can drive the home thermostat, different coffee makers offer similar services that compete with each other).

SIoT can empower developers to program a much wider class of meaningful IoT applications [19][20][21].

III. Raising the Bar – New Thing Requirements

The "anesthetic" effect of abstracting *things* away from the *Internet of Things* has been stimulating unbounded imaginations of exciting IoT scenarios and unprecedented applications. Consequently, such "state of mind" has rapidly been raising big expectations and founding a tricky entanglement between ambitious technological future and hype. While past goals of programmability and friction-free integration remain to be critical and needy of further advancements, the new scenarios seem to be driving new requirements and goals for *smart objects* to meet. We demonstrate some of these new requirements in two sample scenarios in the domain of Personal IoT (e.g., smart homes).

A. The Coffeemaker and the Smartphone Scenario

It is 2019, and the days of coffeemakers with built-in time displays and small "set" buttons are over. Such coffeemakers are considered backward; coffeemakers today buddy up with other thing-mates such as smartphones (or smart watches) to deploy and offer the user a better, more usable interface to their now invisible time functions. Coffeemakers may not even have any built-in time functions, in which case, in addition to interfacing through a smartphone, the coffeemaker would utilize the smartphone time services.

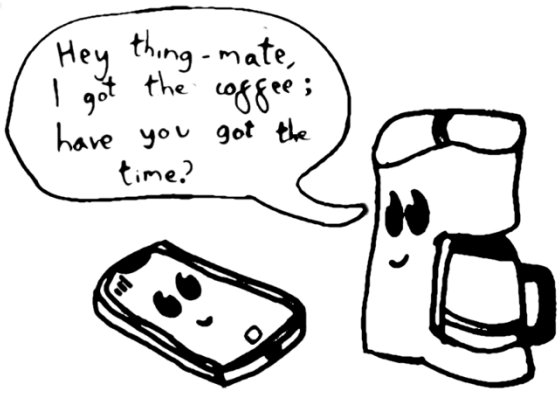

Fig. 1: The foreseen interaction between a coffeemaker and smartphone thing.

The above is a simple, flowing wording one would use to describe an exciting blue-sky IoT scenario and even draw a cute little cartoon for it (Fig.1). Below, however, is a different language – one with a more challenging and demanding tone burdened with technical details of a possible set of requirements that must be met for the two *things* on hand to engage in fulfilling the scenario and making it a reality.

Obviously, both *things* must have a minimal hardware platform allowing them to store data, process code, and communicate. Smartphones are powerful hardware platforms, but now coffeemakers must include a miniaturized, low-cost platform or a system-on-a-chip (SoC). A minimal software layer will need to run atop the hardware (thing middleware, or thing OS) which implements many of the requirements we will discuss here. *Things* should also be able to sense each other's presence perhaps through low-energy proximity beaconing (e.g., utilizing Bluetooth low energy (BLE)), and should be able to "chat" or tweet to learn about what each has to offer or capable of. Also, they should be able to chat to learn about each other's interest. For instance: the coffeemaker's interest in a *thing* with interface-hosting capability. Let us zoom in and analyze these requirements for a minute before we present additional ones. The following briefly described requirements are observed.

- *Thing* hardware capabilities could vary widely ranging from minimal to powerful platforms such as smartphones, edge computers. Yet, all *things* should be able to engage and interact with each other. This defines the requirement for minimal hardware.
- *Things* should be able to chat regardless of their make, specific hardware platform, the communication medium available to them, or the language they use to communicate. This may further require *things* to speak multiple languages (e.g., an HTTP-REST-speaking thing may need to somehow understand MQTT-speaking thing messages). This may also require *things* to use "mediator *things*" such as multi-interface edge devices or even the cloud to make communication possible despite incompatible communication media. This set of requirements may seem challenging and it is. However, given the broadness and the wide varieties of *things*, the importance of these requirements cannot be overstated, or we risk an ecosystem of fragmented, non-cooperating *things*, hampering IoT progress.
- *A thing*, giving a descriptive summary of its capabilities and services through its OEM, should be able to dynamically convert such description into actual executable services and build interfaces (e.g., APIs) for others to engage and trigger the offered services.
- *Things* should be able to chat their capabilities and services both conceptually (so other *things* can reason if such capabilities are of interest), and programmatically (by providing a precise description on how to request the service, e.g., an API). *Things* may also chat concepts they are interested in but in a "natural language" without requiring or following any stringent protocols or using specific ontologies. Such protocols or ontologies will have to be both standardized and widely accepted, which are difficult goals that are slow to attain. Hence, the requirement here is to make *smart objects* chat and reason as naturally (humanly) as possible. This is no doubt a difficult requirement but attempting to defragment or unify all IoT *things* through a standardized set of concepts and ontologies is far more difficult. Having a smaller, focused, and community-driven consensus on specialized concepts will be very helpful though in meeting this requirement.

More would be required to enable such a simple coffeemaker-smartphone scenario. Sensing each other's presence, tweeting capabilities and interest and exchanging API's are not enough. Somehow, all such preparations need to be made actionable leading to "meaningful interactions" – a challenging and a missing piece of the IoT puzzle. Which *thing* will put together and autonomously initiate a meaningful interaction? Is a meaningful interaction the same thing as an IoT application? What is the exact definition and composition of such interactions? Should not such interactions be allowed to span more than two *things*? What if multiple interactions are initiated simultaneously? Could there be conflicting interactions? And how can such conflicts be handled? These are some of the questions that need to be addressed very carefully and adequately.

In our scenario, a meaningful interaction may be created and initiated by the coffeemaker requesting the smartphone to accept to host its coffeemaker interface. It may consist of a sequence (a recipe) of API calls along with assets such as HTML5 interface code and other "attachments". It may even include mobile native code (e.g., Android services) that somehow can be dynamically linked to the preloaded thing middleware on the smartphone. Such native code could enable coffeemakers to utilize the smartphone not only as an interface-hosting thing, but also as a time service provider. We explain the requirements of meaningful interactions further in the next scenario.

B. The 2018 & 2022 World Cup Scenario

In 2018, soccer fans around the world had many *devices*, all smart, all connected, but no Cigar (no IoT)! From smartphones with World Cup apps, to smart TV's to connected Digital Video Recorder (DVR), all in their homes, and even accessible remotely. Browsing the semi-final games' schedule on a mobile app revealed the historic *France-Belgium* game that a fan is unable to watch due to an urgent business meeting with an important client. The fan is forced to put down the smartphone, move to the living room, search for the basket of remotes, sits down to remember and figure out how to program the darn DVR to record the game – what a chore. In 2022, the cup watching experience could be very different—enhanced by IoT. A fan browsing the games' schedule will be able to program a recording of the game directly from the app. Here, the app developer did not make assumptions about the availability of a DVR thing in the user's "smart space" and did not write any code to enable such a recording. The DVR thing and the World Cup app chatted and exchanged interest and capability lists, which resulted in the DVR thing initiating a meaningful interaction with the mobile app offering a recording service. The fan is presented with a toast or a pop-up screen (an asset, part of the meaningful interaction) asking if she wants to record the game on its DVR. On answering "yes", she is asked if she wishes to have recording as a permanent feature of the app she is using, which she happily accepts.

This scenario underlines a couple of extra important requirements. First, mobile apps should be fully supported as first class *things* in future IoT. They are very opportune and extremely useful *things* to have around as they provide actual opportunities for the user to interact with his IoT through their host devices (the smart phone, watch, etc.) There are millions of these mobile app *things* covering all imaginable application areas. It may not be an overstatement they are or will become the "fertilizer" *things* in the personal Internet of Things!

Such a requirement is however significant and requires new ways for apps to spontaneously adapt to changes in their environment. When new devices exchange capabilities with an app, the app must alter its behaviour and functionality, accordingly, deviating from the standard logic intended by its original developers. These alterations could range from how application data is managed and stored, to how UI elements are presented to the user.

Such altering of the app's behaviour presents a reasonable challenge. Even if the interested *thing-mate* can provide all of the logic needed by the app, to control it for instance such as in the DVR thing example, the thing app must know when and where in the app to execute it. It must also know how to execute it. The thing middleware needs flexibility to not only recognize the capabilities of another thing, but also to accept and integrate logic provided by them. Rather than exchanging data or receiving commands, an app acting as a thing must understand how to integrate the API of another *thing* and make it available to the core functionality of the app.

In the above scenario, the developer of the World Cup app did not explicitly implement the ability to interact with a DVR recorder, nor any of the other myriad devices that could utilize information the app provides. Instead, the developer may merely describe the capabilities that can be offered to a smart space, in this case television program IDs and descriptions. When the capabilities are utilized by an interested thing-mate, the app must change its interface to act as a gateway for the user to control its new functionality. This presents another challenge; with minimal consideration from the original developer, the app must allow the user to control when and how its data (whether represented as a list, button, action, or other) is utilized by a *thing*, through an appropriate UI. Trust and security issues are not addressed here to maintain focus on the new functional requirements.

How to adequately express this capability for interaction (from the mobile app side, especially) must be carefully considered. Pressure is put on the app developer: if these capabilities are not able to be adequately represented between *things*, such a scenario cannot occur even if both devices are capable. Expecting app developers to implement and properly utilize such a system may seem to be a tall order but could be facilitated by adding support and tools to their app development pipeline. For example, an IDE plugin may consider cases when the developer is writing code that produces data or an action, asking them if such code could be utilized by a smart space and how its functionality could be described (such as keywords). The plugin could then encode this into a form used by the middleware, reducing the

overhead required of the developer. This information could also be of use when the developer is designing the user interface, for example, with the plugin recommending where to leave space/logic for UI resulting from meaningful interactions with interested *things*. Such enhancement would require the developer to focus less of the intricacies of the thing middleware, and instead consider how their app may be utilized in an IoT ecosystem.

Now to the second requirement of this World Cup scenario. The increased responsibility of the interested thing to dictate new behaviour to an app also emphasizes the requirement of attachments—resources available externally to a thing. It is unlikely the *devices* interacting with an app maintain copies of all meaningful interaction assets (such as unique copies for Android and iOS, or localization variants) for all the potential interactions the thing intends to support. This is especially significant for more lightweight *things* which may not have the resources to store application assets such as HTML/mobile interface code. Rather, it may be more manageable to architect *things* in a way that allows additional elements to be "attached" just-in-time to a thing for direct use, but stored, hosted, updated, and managed in the cloud or a powerful edge.

The Atlas Thing Architecture, briefly discussed in section V below, attempts to support this scenario [38], utilizing its social relationships, tweeting capabilities and on-board device descriptions. The overall functioning of this scenario is represented in Fig.2. Here, the DVR OEM encodes the device's identity and capabilities, along with a list of keywords describing potential interests (such as receiving a channel to record), in a descriptive schema (the IoT-DDL, briefly discussed in section V below). The mobile app, on the other hand, describes no services in its developer-written schema, only providing keywords of its capabilities (such as providing a TV channel). When powered up, these devices begin to broadcast their identities and search for potential relationships. The semantic similarity between the DVR's interests and the mobile app's capabilities results in the DVR sending its API to the app, and the app notifying the user of a new relationship. The app can then expose a new web view (defined by the DVR) to allow the app user to access the features of the DVR.

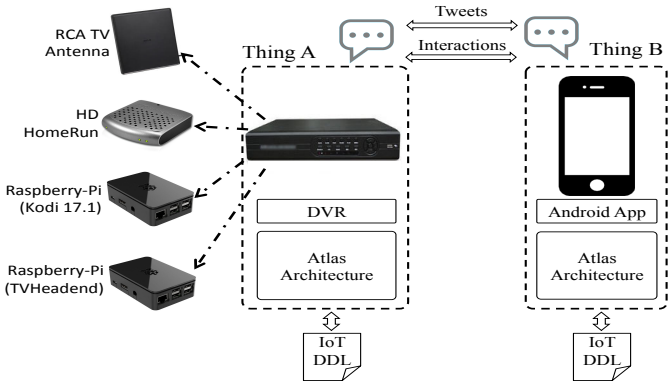

Fig. 2: The World Cup scenario, as viewed by the Atlas Thing Architecture.

IV. The Rise of Thing Architectures

Several architectures and what is widely becoming known as IoT platforms have emerged recently in support of *things* in the IoT. As would be expected, they all encompassed embedded computing elements based on traditional embedded Linux or emerging embedded operating systems specifically designed for the IoT. However, they all broke out of the confines of traditional inward embedding of computing and smartening to additionally provide an outward path of the same. In doing so, embedded elements became part of a broader fabric in an ecosystem spanning other *things*, edge computers and the cloud. This is a significant development marking the evolution from just smart *things* to Internet of *Things*, or the evolution of embedded computing to an "*ascended-embedded*" form of computing. We present only a few examples of such developments due to space limitations.

One of the most fast-evolving developments is ARM Mbed—a licensable thing architecture specifically designed for the IoT with an unmistaken ascended-embedding design philosophy. It is based on the 32-bit ARM Cortex-M family of micro-controllers, which makes it open only to this processor architecture. Mbed OS is a new thing operating system offering integrated set of lightweight yet powerful software components, including security, communication and device management features. Applications developed for (and using) Mbed OS can be embedded into any compliant thing hardware. Additionally, the same applications (or parts of) can be "ascended" and exposed as cloud device services in the cloud. A cloud device service is a light-weight web service (HTTP-REST) which is linked to the actual application running on an ARM Mbed hardware platform. Right here is where the ARM Mbed delivers – making *devices* part of the Internet, or creating an Internet of Things, not just an Internet of content or powerful web services. Mbed has a large community of developers today and is hoped to productively align the energy of several stakeholders including IoT solution developers, hardware partners and cloud providers to create a critical mass IoT industry. Requiring ARM Cortex-M processor hardware would certainly remain as a limitation even though the positive impact of the Mbed ecosystem on advancing IoT is highly recognized [31][32].

Another development is the SmartThings platform. Through development and acquisitions, Samsung was one of the first to establish an IoT ecosystem consisting of *things* (actually brand-named SmartThings), SmartThings Hub (an edge computer) and the Samsung Cloud. Several SmartThings are offered by Samsung itself in the personal IoT (consumer) market. To promote the creation of more *things* by third party in this ecosystem, Samsung has pushed the ARTIK [33] hardware platform as an embedded/ascended platform that effortlessly connects with SmartThing Hubs and the Samsung Cloud. Several ARTIK architectures are provided to cater to very small, medium or complex *smart objects*. The programming model is simple in which an application is composed of restricted, Java-like code (known

as Groovy script) that executes on the SmartThing hub and/or the Samsung cloud. The SmartThings follow SOA in which HTTP-REST APIs are generated and hosted in the Hub and/or the cloud (known as device handlers). Such an API is automatically linked and connected to the actual SmartThing (no effort needed by the developer). Groovy scripts simply access the API's for device information or control. A less open but very similar IoT ecosystem is Wink, spanning Wink *things*, Wink hub and the Wink cloud. HTTP-REST is also used to externalize Wink *things* on the Wink hub.

The Amazon AWS IoT ecosystem is another emerging thing architecture. Here, emphasize is placed on the cloud with significant support to allowing *devices* to access and interact with AWS services. An SDK is provided for third party to bring up any device into this ecosystem. The SDK acts as a middleware that connects the device to an AWS IoT Gateway, a software stack that runs on standard edge devices which provides support for lightweight and secure communication with the AWS cloud. Applications are cloud-hosted in the form of rules run by a rule-engine. *Things* are utilized by the applications through cloud-based web service proxies that are synced with the physical *things* through MQTT messages exchanged between the proxies and the device gateways. Similar emerging ecosystems are IBM's BlueMix and GE's Predix.

While recent developments are impressive by all measures, they each had to make their own assumptions or set some limitations on openness and interoperable participation. That is, as we start, we seem to be building silos of fragmented ecosystems. For instance, only SmartThings can be used within the Samsung ecosystem. Additionally, only Zigbee and Z-Wave technology are compatible with the same. While not universal and interoperable, current efforts are very important as they provide key early lessons to learn from. These ecosystem silos also make for an excellent platform to test and sift through emerging ideas, standards and open elements which guides adoption and eventually consensus and interoperability.

V. Reporting on our own Initial Efforts Towards Thing Architectures

In this section, we present some of our own efforts and attempts to contribute to advancing thing architectures and IoT programming models. We briefly describe the IoT Device Description Language (IoT-DDL) which is a descriptive language that seeks to achieve seamless integration between *things* and also affect some level of homogenization with the IoT ecosystem. To utilize such descriptive language, a lightweight architecture we call the Atlas Thing Architecture is designed and implemented. The architecture provides new layers and services to existing embedded operating systems and introduces novel capabilities a thing must have to be easily configured and managed, and to seamlessly engage with other *things* in hopefully powerful IoT scenarios. We then briefly present an overview of an inter-thing relationships programming framework that describes how services offered by *things* can be combined to build applications explicitly or opportunistically through a new set of service-level relationships.

A. IoT-Device Description Language (IoT-DDL)

The IoT-DDL is a machine- and human-readable descriptive schema used to describe, through a set of attributes and parameters, a thing in the smart space [15]. The thing description should be part of the thing itself to facilitate a thing's smooth migration from one smart space to another and to enable thing-to-thing direct ad-hoc interactions. The IoT-DDL is based on and extends the original Atlas DDL [13][14] to describe the thing in terms of the thing's identity, resources (e.g., network capabilities), inner software and hardware-based components (e.g., sensors and actuators), and the services such thing offers to the surrounding smart space. The OEM can also describe the cloud-based expansions attached to the thing to provide further representations (e.g., virtualization) and extra services (e.g., database) that are considered too heavyweight to be hosted on the thing.

The IoT-DDL also describes the knowledge (social bonds and relationships) injected or acquired by the thing, as well as the different interactions that engage the thing with other entities in the ecosystem. Such configuration scheme is then uploaded to the thing to enable the thing to self-discover its own power and engage with the surrounding IoT ecosystem. A thing in a smart space may engage with thing mates, users (e.g., end-user, developer), computing edges and cloud platforms through a set of action- and information-based interactions. Action-based interactions include management commands, lifetime updates and configurations from authorized parties, as well as the applications that target the thing's capabilities and services. Information-based interactions enable a thing to announce its identity, capabilities and APIs to others. A thing may tweet what it is, what it does and what it knows to the other thing mates. A thing can also share how socially it is related and linked with its thing mates. These social binds can be in terms of some identification attributes (e.g., *things'* vendor, space characteristics) and some relationships that show how the offered services can be logically and functionally tied (e.g., the proper display of an indoor TV requires the window blinds to close, thus the window blinds support the indoor TV). Such social network of logically connected *things* can guide the building of meaningful applications.

B. Atlas Thing Architecture

The current IoT platforms and architectures [36][37] link the access of *things* to a central point (e.g., cloud platforms or edge) where space users can access resources and collect data. This type of architecture highly facilitates cloud-based application development. On the other hand, direct communication between *things* is hardly supported, and the IoT clouding is mostly vendor-specific. Such vendor-restricted connections narrow down the capabilities of

connecting other types of *things* from different vendors seamlessly to the smart space. Such a restricted paradigm ignores the distributed nature of IoT, which requires *things* to communicate with other *things* as well as with cloud platforms and edge for the seamless integration, device management, and engagement with others in ad-hoc IoT scenarios and apps. The Atlas Thing Architecture [15] is a set of software operating layers that utilizes the specifications of the IoT-DDL to provide novel capabilities a thing requires to engage in IoT scenarios and applications.

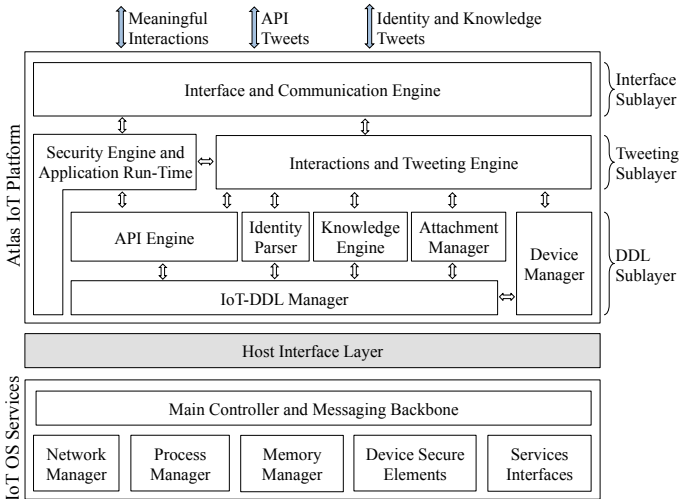

Fig. 3: Atlas Thing Architecture.

Fig.3 depicts the Atlas Thing Architecture, which is based on and extends the original Atlas architecture [2][3]. It takes advantage of the thing's OS services (e.g., network module, I/O ports and physical interfaces, and its process manager) to provide new functionalities for the thing to: 1) self-discover its characteristics, resources, and capabilities through the uploaded IoT-DDL along with the dynamic generation of its own services and the formularization of the appropriate APIs; 2) open an ad-hoc channel with a device management server for provisioning, management, and configuration purposes, and 3) enable the secure interactions with thing mates and engagement in IoT applications and scenarios. The Atlas Thing Architecture also enables the communication interoperability between the *smart objects* that speak different communication languages through Atlas protocol translator thing attachment [15][39]. The Atlas Thing Architecture takes advantage of device management standards and object modeling standards to enable thing management and configuration with minimal human intervention, along with IoT communication standards to empower secure ad-hoc interactions between the thing and thing mates. The architecture also extends the micro services concept to enable the dynamic generation of the services by the thing and the formulation of the corresponding APIs for service-oriented ad-hoc meaningful interactions to take place between the *things*.

C. *Inter-thing relationships programming framework*

The inter-thing relationships programming framework [21] utilizes both Atlas Thing Architecture and the thing IoT-DDL description language to build a distributed programming ecosystem for the social IoT. The framework broadens the social bonds (thing-level relationships) between *things* according to their identification attributes (e.g., vendor, *things* collocated in same space) and utilizes a new set of relationships between the offered services (e.g. Garmin as a GPS device competes with the GPS service offered by a smartphone, A DVR that records a TV channel extends the functionalities of a smart TV that can display channels) that we believe can empower developers to program a much wider class of meaningful applications.

These relationships logically and functionally tie these services to empower the developers to build domain-related applications. The framework introduces service (abstraction of the function offered by a thing), relationship (abstraction of how different services are linked together) and recipe (abstraction of how different services and relationships build up a segment of an app) as the primitives for the Atlas IoT application. The framework also defines Filter, Match, and Evaluate as three operators that functionally define how the primitives are wired. The proposed framework also facilitates the description of the IoT application through a set of semantic rules, that evaluate the correctness of the established application by the developer and guide the execution.

The relationships defined in the framework can be: 1) utilized by vendors in the *things'* IoT-DDLs, and 2) utilized by developers while building IoT apps, and 3) dynamically inferred from the exchanged knowledge (relationships and social bonds) between the *things*. The discovery and inference of links between un-related services suggest the existence of new engagement opportunities to the application developers.

VI. The Future of Things

Things are a new breed of computers that we recently started to more fully understand, in terms of importance and the huge role they will play in shaping our future. They are a thrilling development in the making, currently being shaped by multiple communities each addressing different sets of its demanding requirements. The VLSI community, for instance, is targeting extremely low energy and extremely low-cost designs that can deliver a complete system-on-a-chip including computation, security, and wireless communication. Even software giants such as Microsoft are delving into developing IoT hardware (the Azure Sphere [40]) given the extreme importance of *things*, and the importance of putting requisite software on them. Another community is machine learning, which is working hard on developing scalable big data, cloud-based algorithms for analyzing *thing* data and embedding constrained learning directly on the *things* themselves. Security is another critical area of development. By exploring the use of hardware

secure elements and blockchains, this community too will eventually influence *thing* architectures greatly. Understanding key *thing* requirements and developing solutions for meeting them is currently a feasible challenge. Managing the complexity of meeting all such requirements combined and in small packages could prove trickier than we have ever thought.

REFERENCES

[1] L. Holmquist, F. Mattern, B. Schiele, P. Alahuhta, M. Beigl and H. Gellersen (2001). Smart-Its Friends: A Technique for Users to Easily Establish Connections between Smart artefacts. In international conference on Ubiquitous Computing (pp.116-122). Springer, Berlin, Heidelberg, 2001.

[2] King, J., Bose, R., Yang, H. I., Pickles, S., & Helal, A. (2006, November). Atlas: A service-oriented sensor platform: Hardware and middleware to enable programmable pervasive spaces. In local computer networks, proceedings 2006 31st IEEE conference on (pp. 630-638).

[3] J. King, R. Bose, H. Yang, S. Pickles and A. Helal (2006). Atlas – A Service-Oriented Sensor Platform. Proceedings of the first IEEE International Workshop on Practical Issues

[4] Johnson, M., Healy, M., van de Ven, P., Hayes, M. J., Nelson, J., Newe, T., & Lewis, E. (2009, October). A comparative review of wireless sensor network mote technologies. In SENSORS, 2009 IEEE (pp. 1439-1442).

[5] Michelson, B. M. (2006). Event-driven architecture overview. Patricia Seybold Group, 2(12), 10-1571.

[6] Guinard, D., Trifa, V., & Wilde, E. (2010, November). A resource oriented architecture for the Web of Things. In IoT(pp. 1-8).

[7] Open Services Gateway initiative Alliance. https://www.osgi.org/

[8] Chen, C., & Helal, S. (2008). Sifting through the jungle of sensor standards. IEEE Pervasive Computing, 7(4).

[9] Aloisio, G., Conte, D., Elefante, C., Epicoco, I., Marra, G. P., Mastrantonio, G., & Quarta, G. (2006, June). SensorML for Grid Sensor Networks. In GCA (pp. 147-152).

[10] Lee, K. (2000). IEEE 1451: A standard in support of smart transducer networking. In Proceedings of the 17th IEEE Instrumentation and Measurement Technology Conference [Cat. No. 00CH37066] (Vol. 2, pp. 525-528). IEEE.

[11] Yick, J., Mukherjee, B., & Ghosal, D. (2008). Wireless sensor network survey. Computer networks, 52(12), 2292-2330.

[12] Raghavendra, C. S., Sivalingam, K. M., & Znati, T. (Eds.). (2006). Wireless sensor networks. Springer.

[13] Chen, C., & Helal, A. (2009, July). Device integration in SODA using the device description language. In Applications and the Internet, 2009. SAINT'09. Ninth Annual International Symposium on (pp. 100-106). IEEE.

[14] Mobile and Pervasive Computing Laboratory University of Florida, Device Description Language Specification (Version 1.2), Nov. 2008.

[15] Khaled, A. E., Helal, A. S., Lindquist, W., & Lee, C. (2018). IoT-DDL—Device Description Language for the "T" in IoT. IEEE Access, 6 (1), 24048-24063.

[16] Helal, S., & Tarkoma, S. (2015). Smart Spaces [Guest editors' introduction]. IEEE Pervasive Computing, 14(2), 22-23.

[17] Helal, S. (2005). Programming pervasive spaces. IEEE Pervasive Computing, 4(1), 84-87.

[18] Nastic, S., Sehic, S., Truong, H. L., & Dustdar, S. (2013). Patricia-a novel programing model for iot applications on cloud platforms. In In 6th IEEE International Conference on Service-Oriented Computing and Applications (SOCA), 53-60

[19] Atzori, L., Iera, A., Morabito, G., & Nitti, M. (2012). The social internet of things (siot)–when social networks meet the internet of things: Concept, architecture and network characterization. Computer networks, 56(16), 3594-3608.

[20] Kranz, M., Roalter, L., & Michahelles, F. (2010, May). Things that twitter: social networks and the internet of things. In What can the Internet of Things do for the Citizen (CIoT) Workshop at The Eighth International Conference on Pervasive Computing (Pervasive 2010) (pp. 1-10).

[21] Khaled, A. E., & Helal, S. (2018, February). A framework for inter-thing relationships for programming the social IoT. In IEEE 4th World Forum on Internet of Things (WF-IoT), 670-675.

[22] Erl, T. (1900). Service-oriented architecture. Pearson Education Incorporated.

[23] MacKenzie, C. M., Laskey, K., McCabe, F., Brown, P. F., Metz, R., & Hamilton, B. A. (2006). Reference model for service oriented architecture 1.0. OASIS standard, 12, 18.

[24] de Deugd, S., Carroll, R., Kelly, K., Millett, B., & Ricker, J. (2006). SODA: Service oriented device architecture. IEEE Pervasive Computing, 5(3), 94-96.

[25] IBM "How IBM Sees the Internet of Things," from ibm.com/internet-of-things, 2015.

[26] Loomis, D., & Cargill, T. (2001). The TINI specification and developer's guide. Addison-Wesley.

[27] Román, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R. H., & Nahrstedt, K. (2002). Gaia: a middleware platform for active spaces. ACM SIGMOBILE Mobile Computing and Communications Review, 6(4), 65-67.

[28] Schoofs, T., Jenn, E., Leriche, S., Nilsen, K., Gauthier, L., & Richard-Foy, M. (2009, September). Use of PERC Pico in the AIDA avionics platform. In Proceedings of the 7th International Workshop on Java Technologies for Real-Time and Embedded Systems (pp. 169-178). ACM.

[29] J. King, R. Bose, H. Yang, S. Pickles and A. Helal (2006). Atlas – A Service-Oriented Sensor Platform. Proceedings of the first IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006). Tampa, Florida, 2006.

[30] Guo, B., Zhang, D., Wang, Z., Yu, Z., & Zhou, X. (2013). Opportunistic IoT: Exploring the harmonious interaction between human and the internet of things. Journal of Network and Computer Applications, 36(6), 1531-1539.

[31] ARM Connected Community (2016). From https://community.arm.com/

[32] ARM white paper "An Introduction to the ARM mbedTM IoT Device Platform," October 2014.

[33] Wootton, C. (2016). Samsung ARTIK Reference: The Definitive Developers Guide. Apress.

[34] The Continua Health Alliance. From www.continuaalliance.org

[35] Nitti, M., Pilloni, V., Colistra, G., & Atzori, L. (2016). The Virtual Object as a Major Element of the Internet of Things: A Survey. IEEE Communications Surveys & Tutorials, 18(2), 1228-1240.

[36] Li, S., Da Xu, L., & Zhao, S. (2015). The internet of things: a survey. Information Systems Frontiers, 17(2), 243-259.

[37] Khan, R., Khan, S. U., Zaheer, R., & Khan, S. (2012, December). Future internet: the internet of things architecture, possible applications and key challenges. In Frontiers of Information Technology (FIT), 2012 10th International Conference on (pp. 257-260). IEEE.

[38] Helal, S., Khaled, A. E., & Gutta, V. (2017). Atlas thing architecture-Enabling mobile apps as things in the IoT: 23rd Annual International Conference on Mobile Computing and Networking, MobiCom 2017.

[39] Khaled, A. E., & Helal, S. (2019). Interoperable communication framework for bridging RESTful and topic-based communication in IoT. Future Generation Computer Systems, 92, 628-643.

[40] Microsoft Internet of Things' Azure Sphere. https://azure.microsoft.com/en-gb/services/azure-sphere/