On Data Program Interfaces

D. Namiot[1] and M. Sneps-Sneppe[2]

[1]*Lomonosov Moscow State University*
[2]*Ventspils University College*
E-mail: {dnamiot; manfreds.sneps}@gmail.com

Received: 24 November 2014;
Accepted: 9 March 2015

Abstract

In this paper, we discuss the global unified standards for the Internet of Things software products and existing de-facto standards (practical approaches). Using specific examples of interaction with Bluetooth Low Energy tags we compared existing approaches to the development and the proposed global standards (FI-WARE). Can the proposed unified approach to the creation of services to cover all the possible use cases and scenarios for Internet of Things and Smart Cities services? The paper emphasizes the need to address the simplicity of the development and highlights the critical importance of such a thing as the time to market for new applications and services. By our opinion, time to market and the simplicity of the development are key parameters for any proposed software standard.

Keywords: Internet of Things, FI-WARE, API, DPI, Bluetooth.

1 Introduction

This paper presents an extended version of our presentation for ITU Kaleidoscope Conference [1]. In this article we want to compare unified programming standards and practices of development.

In our research, we will talk about the standards that affect software development. And even more precisely, we will discuss Internet of Things (IoT) software. Nowadays, we can see many attempts to create a unified

Journal of ICT, Vol. 2, 269–288.
doi: 10.13052/jicts2245-800X.234

interface (platform, API) for IoT [2, 3]. So, it is correct to talk about an adaptation (adoption) of proposed standards by the existing development community. Only the adoption by the community ensures their wide distribution and use. Otherwise (which is not uncommon), we are faced with a situation where standards exist in parallel and independently of the established practice. In recent history, we can recall examples of real opposition to the proposed standard from the existed approaches and practices. For example, we can point out here the confrontation of TCP/IP protocol stack and the ISO, Corba and Web services, IIOP and XML, and on the same Web Services versus REST, XML versus JSON, etc [4]. And it is not always the confrontation ended with the benefit (in favor) of an official standard. As seems to us, the standards proposed for the software development have their own specifics.

Using specific examples of interaction with Bluetooth Low Energy tags, we compared the existing approaches to the development and the proposed global standards (FI-WARE). We would like to show that in many cases the proposed approaches are over-engineered and, actually, complicates the development process.

The rest of the paper is organized as follows. Section 2 describes the philosophy behind the software standards. Section 3 presents Bluetooth Low Energy tags. Section 4 discusses de-facto standard programming approach for Bluetooth Low Energy tags. Section 5 describes FI-WARE project. Section 6 targets Smart City programming in FI-WARE. Section 7 presents a discussion.

2 On Software Standards

In general, the typical standards committee starts with an idea. The idea could be adopted as a work item and then committee takes it through the successive stages of standardization (i.e. the standards process) [5]. As a result, we can get the standard specification. It could be implemented in a product or a service (a standard implementation). The paper [5] highlights the following sources of implementation problems:

- The idea that underlies a standard may not be implementable (e.g. too comprehensive).
- The ideal of consensus decision-making may affect the standards process. Typically, it leads to too many options ("a camel is a horse designed by a committee"). It affects, of course, the implementability of standards.
- Different use of terminology in a standard specification may lead to problems of interpretation, implementation and interoperability.

- Modest user requirements and cost-constraints in the implementation process may lead to partial standard compliance and incompatible implementations.

In the telecommunications world, the most interesting example is, of course, the whole story behind the Parlay. We saw a whole family of APIs: Parlay/OSA, Parlay X, which can be described as a simplified version of Parlay/OSA, then JAIN. This constant redesigning and repositioning of standards leads to a loss of meaning as to that constitutes a standard [6]. Parlay is also a great example of incompatibility for standards implementation.

An interesting fact is that for software systems the standard implementation does not oppose to some already well-established technology. With programming standards, where emerging de facto procedures play the major role, the whole picture is more complicated. At the time of introduction (during initial phases) common standards for their respective approaches to development have yet to develop. So, we can say that standard committee and developers outside of it are in the same conditions. But nevertheless, a more pragmatic approach invariably won. Each standard (relating to software development) proposes an all or nothing approach. And, accordingly, it sought to cover all the imaginable aspects. It leads (especially, with the above mentioned ideal of consensus decision-making) to the highest possible level of generalization. This, of course, is both the strength and weakness of this approach. In general, the problems with such an approach are the same as with the introduction new enterprise architecture.

What are the typical key challenges for the enterprise for implementing a new architecture? On the first hand it is the alignment of the organization's capabilities and strategy with the environment in which it operates. The same is true for the world of software development. A new standard will play a significant role in realigning the organizational structure for the software companies. Obviously, it changes the stability. And too many changes in a short time frame can cause the resistance. But in the same time misalignment can create big problems for software companies.

Capacity varies across development teams. So, the need for resource commitments is another reason for the resistance. Obviously, the resource commitments affect the implementation plan in any software company (team of developers). Other factors we can mention here are diversity of software companies, complexity of new offering as well as the need for coordination during the implementation phase.

But from our point of view, the main (determining) parameter is the answer to the main question of interest to developers. This issue is the time it takes

to build services (applications) using a novel approach. Time is a key factor in software development [7]. Can we save a time with new approach? The biggest problem with Parlay was the fact that actually this standard increased the time for development (time to market for new services).

In light of the above, we want to compare two approaches to the development of services for the same type of sensors. One of these approaches is the proposed standard FI-WARE and the second one is the de facto folding (forming) approach to development: Bluetooth Low Energy (BLE). One of the approaches is pushed by the international community, where the second is initiated by one vendor. Actually, this use case is a typical for the software development. The most successful outcome for such kind of projects is when the private software development (software product) is becoming a standard, supported by the community.

We choose BLE related development as the typical example of sensors connected projects. From the one side, this development should be effectively supported by the proposed standard. But in the same time, we have a powerful vendor (Apple) with the own development. And this vendor has got enough power to turn any of the own projects into standard entity.

In general, we can describe two approaches for the software standards. Firstly, we can talk about some unified (generic) API. Any generic approach will cover a wide area of elements (devices and their models in case of IoT) and lets programmers use the same methods across the whole project. And the second approach is a specialized (vertical) API targets one particular class of entities (devices in case of IoT). Usually, the direct proposals from hardware vendors are more compact. Although of course, we cannot expect the broad applicability here.

3 Bluetooth Low Energy

Bluetooth low energy (BLE) as a technology started as a project in the Nokia Research Centre. Later it was adopted by the Bluetooth Special Interest Group [8]. The aim of this technology is to enable power sensitive devices to be permanently connected to the Internet. BLE sensor devices are typically required to operate for many years without needing a new battery [9]. As an ecosystem of other devices, BLE may also be known as Bluetooth v4.0 and is part of the public Bluetooth specification [10]. A device that operates Bluetooth v4.0 may not necessarily implement other versions of Bluetooth, in such cases it is known as a single mode device. Bluetooth classes are:

Class 1 (100 mW, 100 m range)

Class 2 (2.5 mW, 10 m range)

Class 3 (1 mW, 1 m range)

There are other low power technologies that could be deployed. For example, ANT which operates in the 2.4 GHz spectrum, ZigBee, Zigbee Reduced Function Device (RFD), etc [11].

Apple has been embedding Bluetooth Low Energy in its devices since iPhone 4s. Since iOS7 release, Apple has released iBeacon API. It is programming interface to low energy sensors from Apple (iBeacons).

In 2013, over 250 million Bluetooth Smart accessories are expected to ship. By 2016, this is expected to reach over a billion [12]. At this moment BLE has been supported by 100% of Apple iOS devices since iPhone 4S (including the iPod Touch, iPads, and all phones), and is roughly supported by 30% of new shipped Android phone models. So, these Bluetooth-enabled devices will interact with iOS and Android devices. Beacons can take any form factor and can be placed anywhere. From a developer perspective, they simply advertise data in peripheral mode by broadcasting some unique identifier. Actually, each tag broadcasts 3 numbers: own unique ID and two digits (so called minor and major). Minor and major could be configured in application in order to distinguish different areas. Application developers then use these numbers to understand the location of a particular device and connect an application (a mobile user) to a service or to content in the cloud.

Existing use cases include retailers and venue owners (see Figure 1).

Actually, the same picture is correct for Gimbal beacons [13] too, for example. As we can see, application to sensors communication link is a read-only channel. It is a very important remark. In practice, most of IoT applications are reading data only. It means, that the word API borrowed from telecom is very often means over-engineering here. Actually, most of the devices (sensors) do not accept data and cannot execute commands. They can only transmit data. They can do that by the own initiative (push) or do it as a response to any request (poll). This means also that the security constraints can be seriously reconsidered. Usually, by the obvious reasons, read only is much more safe, than read/write. Especially, if we keep in mind the fact, that present data for the reading is the main function for the most of our devices (sensors). They are not made for data hiding.

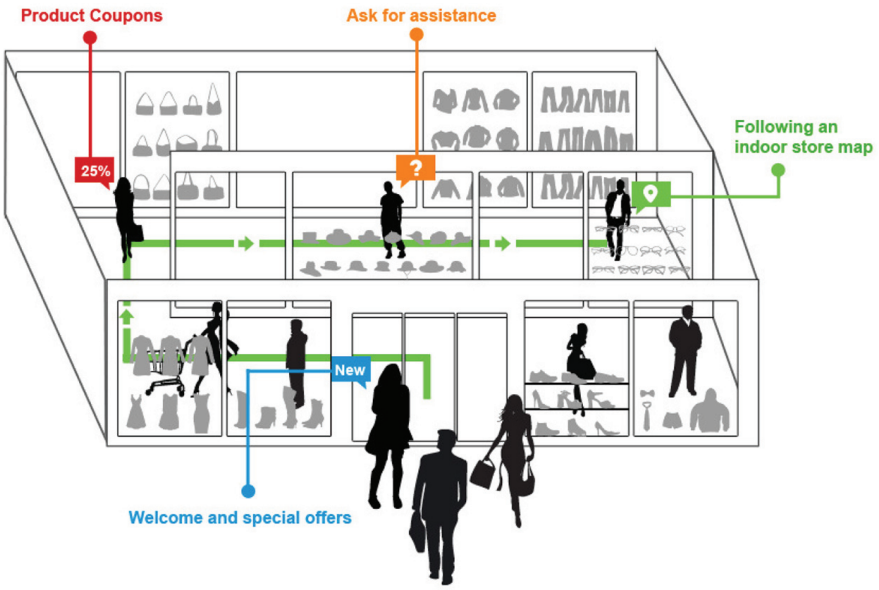


Figure 1 iBeacons use cases [14]

So, we can treat most of the devices as tags. We have presented iBeacon here just due to fact that the functionality described in the specific embodiments is wide enough by itself.

And the model, selected for this family of tags can be used in many applications. Therefore, it is important to standardize access to such facilities (tags). This standard should affect applications (services) that work with tags. Standardization will ensure the porting of applications and easy replacement of hardware devices.

And it is a perfect example for standard versus de facto approach in software development. On the one hand, we have FI-WARE standards that should cover sensors, on the other hand, we have developers around the existing mobile operating systems.

Bluetooth Low Energy uses Generic Attribute Profile (GATT) for service discovery [15]. One device looks for others and sends ID Packets. One or more other devices listen (these devices are discoverable). When acceptable packet is received, searching device may continue with this device or continue looking for other devices. When a desired device is found, connection process begins. There is always master-slave communication. One master controls communications and can support up to 7 active slaves. This system does not

support slave-to-slave communication. So, only one device always initiates the connection. Other device has to be willing to accept a connection. Once listening device hears the ID packet with its address, it replies.

GATT provides a framework for developing profiles. A profile is composed of one or more services. A service is composed of characteristics or references to other services. Each characteristic contains a value and may contain optional information about the value [16]. Note, that it is a classical client-server model. Also, we should note that response's description is a good fit for JSON (one of the favorite formats for the modern programming).

Mobile phone supports the central role and BLE device supports the peripheral role. Once the phone and the BLE device established a connection, they start transferring GATT metadata to one another.

4 BLE Programming

The description below targets Android 4.3 (API Level 18). The *BluetoothAdapter* is required for any and all Bluetooth activity [17]. The *BluetoothAdapter* represents the device's own Bluetooth adapter. There is one Bluetooth adapter for the entire system.

```
// Initializes Bluetooth adapter.

final BluetoothManager bluetoothManager =
        (BluetoothManager)
getSystemService(Context.BLUETOOTH_SERVICE);

mBluetoothAdapter = bluetoothManager.getAdapter();
```

In order to find BLE devices, we can use the *startLeScan()* method. This method takes a *BluetoothAdapter.LeScanCallback* as a parameter. In this callback we can deal with scan results.

```
mBluetoothAdapter.startLeScan(mLeScanCallback);
```

Here is an implementation for callback as per Google manual:

```
private LeDeviceListAdapter mLeDeviceListAdapter;

. . .

// Device scan callback.
```

```
private BluetoothAdapter.LeScanCallback mLeScanCallback =
      new BluetoothAdapter.LeScanCallback() {
  @Override
  public void onLeScan(final BluetoothDevice device, int rssi,
        byte[] scanRecord) {
      runOnUiThread(new Runnable() {
        @Override
        public void run() {
           mLeDeviceListAdapter.addDevice(device);
        mLeDeviceListAdapter.notifyDataSetChanged();
        }
     });
   }

};
```

As soon as the device is updated, we can connect to a GATT server on the BLE device: *connectGatt()* method. This method takes three parameters: a Context object, *autoConnect* (a boolean indicating whether to automatically connect to the BLE device as soon as it becomes available), and a reference to a new callback.

Once our Android application has connected to a GATT server and discovered services, it can read and write attributes. So, if we skip the Android callback-based architecture, it is very straightforward: discover the device, make a connection and read attributes.

The application can ask to be notified when a particular characteristic changes on the device. And of course, we can stop and start scan.

Some vendors can provide a high level API for access to low energy tags (Samsung, HTC, Estimote [18]). There is *BeaconManeger* object (in iOS) that plays a role of proxy and hides details of the connection:

```
(void)beaconManager:(ESTBeaconManager *)manager
  didDetermineState:(CLRegionState)state
```

```
        forRegion:(ESTBeaconRegion *)region
{
   if(state == CLRegionStateInside)
   {
     [self setProductImage];
   }
   else
   {
     [self setDiscountImage];
   }
}
```

The programming pattern in both cases is the same. We need to find (address) our device (devices) and define a callback method for accepting data. We do not need even to make periodical requests. In this case, tags pushed data by themselves.

5 FI-WARE Approach

FI-WARE approach [19] is our example of unified API. The high-level goal of the FI-WARE project is to build the Core Platform of the Future Internet. FI-WARE is based upon elements (called Generic Enablers) which offer reusable and commonly shared functions serving a multiplicity of Usage Areas across various sectors [20]. Generic Enablers (GEs) are considered as software modules that offer various functionalitiesalong with protocols and interfaces for operation and communication. The Core Platform to be provided by the FI-WARE project is based on GEs linked to the following main FI-WARE Technical Chapters:

Cloud Hosting – the fundamental layer which provides the computation, storage and network resources, upon which services are provisioned and managed.

Data/Context Management – the facilities for effective accessing, processing, and analyzing massive volume of data, transforming them into valuable knowledge available to applications.

Applications/Services Ecosystem and Delivery Framework – the infrastructure to create, publish, manage and consume FI services across their life cycle, addressing all technical and business aspects.

Internet of Things (IoT) Services Enablement – the bridge whereby FI services interface and leverage the ubiquity of heterogeneous, resource-constrained devices in the Internet of Things.

Interface to Networks and Devices (I2ND) – open interfaces to networks and devices, providing the connectivity needs of services delivered across the platform.

Security – the mechanisms which ensure that the delivery and usage of the services is trustworthy and meets security and privacy requirements.

Let us see what is offered by IoT and I2ND enablers. From a physical architecture standpoint, IoT GEs have been spread in two different domains: Gateway and Backend.

The deployment of the architecture of the IoT Service Enablement chapter is typically distributed across a large number of Devices, several Gateways and the Backend (Figure 2).

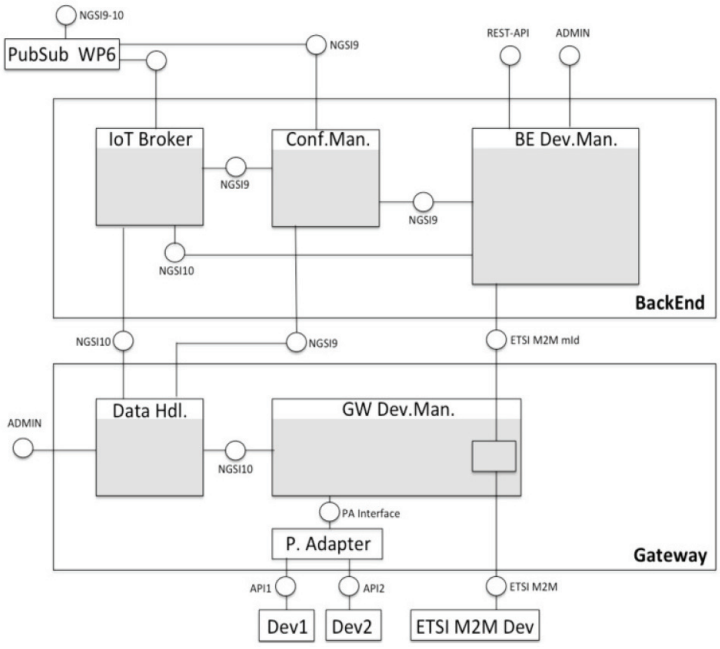


Figure 2 IoT service enablement

FI-WARE IoT Gateway is a hardware device that hosts a number of features of one or several Gateway Generic Enablers of the IoT Service Enablement. It is optional element and could be eliminated (as per FI-WARE spec). FI-WARE IoT Backend is a setting in the cloud that hosts a number of features of one or several Generic Enablers of the IoT Service Enablement. In the FI-WARE IoT model, at least one IoT Backend is mandatory, which will be connected to all IoT end devices either via IoT Gateway(s) and/or straight interfaces [19].

As we see, the unified approach makes data sharing (IoT Backend) mandatory. It could be one of the problems for the above-mentioned applications for access to BLE tags. For example, sharing data to cloud environment immediately adds security requirements. Also, it adds the complexity (read – increases price) for the technical installation. Obviously, it is much easier to install independent tags, rather than provide any cloud connectivity for each of them. Cloud based solution could be simple more costly. So, it is an absolute correct question: do we really need it for all imaginable scenarios? The typical use case for BLE tags is their inspection from smartphones (other personal devices). Smartphone (metering device) itself plays a role of gateway. Mandatory data sharing just adds the complexity.

6 Enablers for Sensing

According to [21], there are more than 60 FI-WARE Generic Enablers (GE) as common building blocks across Use Case projects, and more than 100 Specific Enablers as dedicated building blocks coming from the Use Case projects so as to support their proof of concept and build prototypes. Figure 3 shows GE for IoT:

Of course, this GE is supposed to work with many devices, so it is presented as much abstract as it is possible. For any particular preselected set of end-devices (sensors) the picture could be much simpler, of course.

The IoT Gateway and IoT Backend communicate with each other through an open standardized communication interface. The IoT gateway is responsible also for a protocol adaptation and control service. The protocol adaptation service will handle communication with different devices, while the control service will contain intelligent control logic that will deal with differences between the various implementations of the Gateway and access technologies, as shown in Figure 4.

Because it is a generic API, it provides (defines) all imaginable functions:

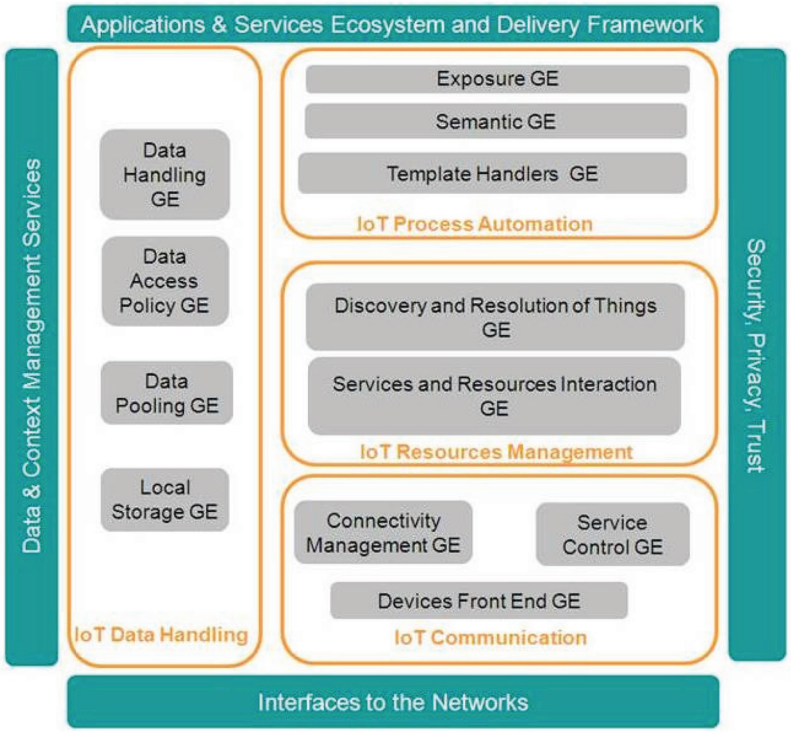


Figure 3 IoT GE

Communication protocols abstraction: a mechanism to enable unified communications between the IoT resources and the IoT backend.

Communication service capabilities: identification of and communication to IoT resources (identification of an IoT resource includes the mapping of physical network addresses to the IoT resource identifier).

Managed connectivity: definition of the interfaces towards the networks for the management of the connectivity.

Discontinuous connectivity: management of the IoT resources that are not always-on.

Support of the IoT mobility for the IoT resources that may physically move, or may change the access network.

Session management: management of the communication sessions to support mechanisms to handle reliability associated with network connections.

Traffic flow management: development of the mechanisms to deal with abnormal and occasional traffic conditions (e.g. overload traffic conditions).

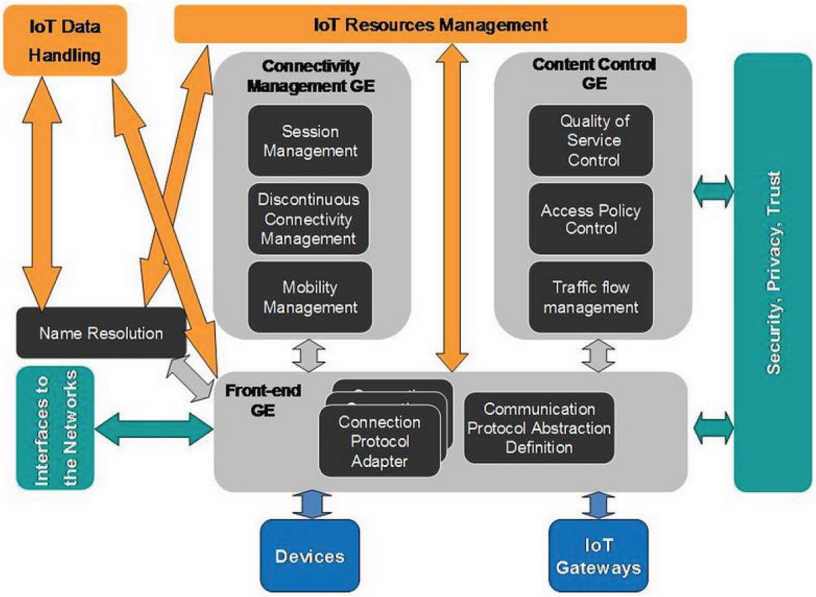


Figure 4 IoT communications [22]

It is interesting, that Fi-WARE documents mention so called "Data as a Service" component. FI-WARE assumes that often the data needs to be stored and later accessed by applications and / or by the IoT Services for its proper operation and management of IoT resources. So, "Data as a Service" is, firstly, some storage in their vision. This storage should follow to some structured approach and has to support consistency during time-to-time synchronization or sporadic events. And "Data as a Service" is, technically, some API for this storage.

We think, that "Data as a Service" should be a main function. At least, it is so for the vast majority of IoT systems. Almost all sensor-based systems fall into this category.

7 Discussion

In paper [23] we analyzed the typical IoT applications for wireless tags (e.g., iBeacons). We can present the top level definition as this:

- There is a set of sensors we need to poll periodically for getting new measurements

- There is a set of sensors we need to accept data from (push data – sensors initiated communications)
- The business process could be presented as a set of productions (rules). Each of the rules depends on some available data and, probably, on some global variables (states).
- The data availability always assumes the presence of data for any finite set of timestamps. In other words, the application makes conclusions (actions) depending on some window of measurements.

So, the top level architecture is very simple. The application finds a tag (device), obtains a descriptor and presents (defines) a callback function for getting data. It looks like a classical model for AJAX deployment: get data via some asynchronous call and proceed them in the predefined callback. It means that this approach will be supported sooner or later in some JavaScript environment.

The best candidate for top level presentation is Web Intents [24]. It is practically the same proxy for asynchronous calls accessible from JavaScript. Note, that the same approach works for many sensors. For example, our own SpotEx approach [25] that collects information from network nodes could be presented in the similar manner. So, as seems to us the following question is correct: do we really need Application Program Interfaces (API) always, or our goal could be described as Data Program Interfaces (DPI)?

We can describe DPI as an interface at the edge of an IoT device that exposes and consumes data. IoT devices very often do not support commands (instructions). Many of sensors just provide some data and nothing more. The above-mentioned "Data as a Service" approach is a good example. We could have data layer (probably, with the persistence support) and client-side processing (JavaScript).

Actually, we can mention several IoT-related projects with the similar conception. There is a Webinos project [26] with a very similar model. It is a European Union project, which aims at developing software components for the future Internet, in the form of Web Runtime Extensions. We can mention also OpenRemote project [27]. Another example is a MQTT protocol from IBM [28].

Actually, publish/subscribe systems are good examples of data centric communications. In the publish/subscribe communication model components which are interested in obtaining data register their interest. For our BLE example, they scan and discover tags. In publish/subscribe systems it is called subscription. Components, which want to share some information, do so

by publishing their data. There is also a broker - the entity which ensures that the data gets from the publishers to the broker. This actor coordinates subscriptions. In our case this broker is personalized. It is an in-application proxy. There are three principal types of pub/sub systems: topic-based, type-based and content-based [29]. By topic-based systems, the list of topics is usually known in advance. E.g., in our case we know the tags and their data. In type-based systems, a subscriber states the type of data it is interested in (e.g., temperature data). It means that we need to maintain some directory of tags with metadata. In content-based systems the subscriber describes the content of messages it wants to receive. Subscriber describes types of data for receiving and condition for the delivery (e.g., the temperature is below a certain threshold). Publish/subscribe it is a good example for DPI too.

We can mention here a wide set of papers devoted to the Web of Sensors with linked data and HTTP based REST protocol [30]. But the basic point for such models could be a serious limitation. The models assume that the HTTP server is deployed on each sensor node, making it an independent and autonomous Web device.

Keeping in mind the above-mentioned examples with BLE tags, FI-WARE GE for this task should be personalized by some way and placed right on the mobile phone. For the many exiting BLE deployments backend is the mobile phone. And there is no cloud involved in this operation.

In general, the main question in our discussion could be provided in this form: can we state that excessive generalization (unification) in software standards could be a biggest source of the problems?

And it is a very significant question to the standard committee. Actually, this is a very important (if not a most important) point. It belongs to any standard devoted to the software development. Shall the standard follow to the "all or nothing" model and covers all the areas of life cycle? Actually, we are talking about all the imaginable areas at the moment of the standard's development. Note, that the above-mentioned scenario with direct access from mobile phone to tags (sensors) is quite common. And we should make its implementation by the most convenient way for mobile developers. They will be responsible for the putting new services in place.

The alternative approach is to target individual technical areas and let developers assemble applications like mashups from standard components. The key moment here is exactly time to market. Mashups allow developers seriously decrease time to market for new services. In the modern software architecture world, we can mention also micro-services approach [31].

We should mention in this context Data-as-a-Service (DaaS) approach [32]. DaaS (in its technical aspects) Data as a Service (DaaS) is an information provision and distribution model in which data files (including text, images, sounds, and videos) are made available to customers over a network. The key moment (again, we do not discuss the business model) is the separation for data and proceedings. The key delivery element is a data chunk rather than some API with the predefined model for data processing. Benefits of DaaS include outsourcing of the presentation layer and reducing overall cost of data maintenance and delivery.

Let us provide some analogue from telecom market. Shall we provide standards for SMS (SMPP) and MMS only or provide a standard framework for all messaging based services? Sure, we can have some benefits from the unified framework, but at the same time we may lose the flexibility. Guarantee, that such a standard model will cover all conceivable services is very low.

The key point is the simplicity and finally, time to market for new applications. Note, that telecom projects could be heavily affected by standard problems due to high diversity in the devices and use case models. Not taking into account the interests of developers to create standards, we risk facing a parallel existence of the standard model and the actually utilized the existing approaches.

References

[1] Namiot, D., & Sneps-Sneppe, M. (2014, June). On software standards for smartcities: API or DPI. In ITU Kaleidoscope Academic Conference: Living in a converged world-Impossible without standards?, Proceedings of the 2014 (pp. 169–174). IEEE.

[2] Trappeniers, L., Feki, M. A., Kawsar, F., & Boussard, M. (2013). The internet of things: the next technological revolution. Computer, 46(2), 0024–25.

[3] Gubbi, Jayavardhana, et al. "Internet of Things (IoT): A vision, architectural elements, and future directions." Future Generation Computer Systems 29.7 (2013): 1645–1660.

[4] Sneps-Sneppe, M., and Namiot, D. (2012, April). About M2M standards and their possible extensions. In Future Internet Communications (BCFIC), 2012 2nd Baltic Congress on (pp. 187–193). IEEE. DOI: 10.1109/BCFIC.2012.6218001.

[5] Egyedi, Tineke M. "Standard-compliant, but incompatible?!." Computer Standards & Interfaces 29.6 (2007): 605–613.

[6] Hawkins, R., and Ballon, P. (2007). When standards become business models: reinterpreting "failure" in the standardization paradigm. Info, 9(5), 20–30.
[7] Schneps-Schneppe, M., Namiot, D., and Ustinov A. "A Telco Enabled Social Networking and Knowledge Sharing." International Journal of Open Information Technologies 1.6 (2013): 1–4.
[8] Persson P., Jung Y. Nokia sensor: from research to product //Proceedings of the 2005 conference on Designing for User eXperience. – AIGA: American Institute Of Graphic Arts, 2005. – p. 53.
[9] Smith P. Comparisons between Low Power Wireless Technologies //US Patent CS-213199-AN. – 2011.
[10] Bluetooth S. I. G. Specification of the Bluetooth System-Version 4.0 //2010. http://www.bluetooth.com. – 2010.
[11] Otte R. Low-Power Wireless Infrared Communications. – Springer-Verlag, 2010.
[12] Namiot, D., and M. Sneps-Sneppe. "On the analysis of statistics of mobile visitors." Automatic Control and Computer Sciences 48.3 (2014): 150–158.
[13] Pongpaichet, S., Singh, V. K., Jain, R., & Pentland, A. S. (2013, October). Situation Fencing: Making Geo-Fencing Personal and Dynamic. In Proceedings of the 1st ACM international workshop on Personal data meets distributed multimedia (pp. 3–10). ACM.
[14] Nitro Mobile http://blog.nitromobilesolutions.com/category/technology/ibeacons/ Retrieved: Nov, 2014–11–18
[15] Silva, S., et al. (2012). A Bluetooth Approach to Diabetes Sensing on Ambient Assisted Living systems. Procedia Computer Science, 14, 181–188
[16] Rate, B., Rate, E. D., LE, L. E., Protocol, A., & Profile, G. A. Introduction to Bluetooth Technology.
[17] Android BLE http://developer.android.com/guide/topics/connectivity/bluetooth-le.html Retrieved Nov, 2014.
[18] Nahavandipoor, Vandad. iOS 8 Swift Programming Cookbook: Solutions & Examples for iOS Apps. "O'Reilly Media, Inc.", 2014.
[19] Usländer, Thomas, et al. "The future internet enablement of the environment information space." Environmental Software Systems. Fostering Information Sharing. Springer Berlin Heidelberg, 2013. 109–120.
[20] Moltchanov, B., & Rocha, O. R. (2014, April). Generic enablers concept and two implementations for European Future Internet test-bed.

In Computing, Management and Telecommunications (ComManTel), 2014 International Conference on (pp. 304–308). IEEE.

[21] Future Internet Public Private Partnership. Outcomes, achievements, and outlook. Phase 1, Final Report, European Commission, Sept 2013.

[22] FI-WARE Wiki https://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_Internet_of_Things_(IoT)_Services_Enablement Retrieved: Nov, 2014

[23] Namiot, D., & Sneps-Sneppe, M. (2014). On Micro-services Architecture. International Journal of Open Information Technologies, 2(9), 24–27.

[24] Sneps-Sneppe, M., and Namiot, D. "M2M Applications and Open API: What Could Be Next?." Internet of Things, Smart Spaces, and Next Generation Networking. Springer Berlin Heidelberg, 2012. pp. 429–439.

[25] Namiot, Dmitry, and Manfred Sneps-Sneppe. "Geofence and Network Proximity." Internet of Things, Smart Spaces, and Next Generation Networking. Springer Berlin Heidelberg, 2013. pp. 117–127. DOI: 10.1007/978-3-642-40316-3_11

[26] Fuhrhop, C., Lyle, J., and Faily, S. (2012, April). The webinos project. In Proceedings of the 21st international conference companion on World Wide Web (pp. 259–262). ACM.

[27] Arsénio, Artur, et al. "Internet of Intelligent Things: Bringing Artificial Intelligence into Things and Communication Networks." Inter-cooperative Collective Intelligence: Techniques and Applications. Springer Berlin Heidelberg, 2014. 1–37.

[28] Hunkeler, Urs, Hong Linh Truong, and Andy Stanford-Clark. "MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks." Communication Systems Software and Middleware and Workshops, 2008. COMSWARE 2008. 3rd International Conference on. IEEE, 2008.

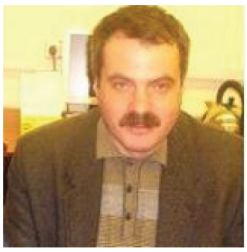
[29] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kernmarrec, The many faces of publish/subscribe, ACM Computing Surveys, vol. 35, no. 2, pp. 114–131, June 2003.

[30] Guinard, D., Trifa, V., & Wilde, E. (2010, November). A resource oriented architecture for the web of things. In Internet of Things (IOT), 2010 (pp. 1–8). IEEE.

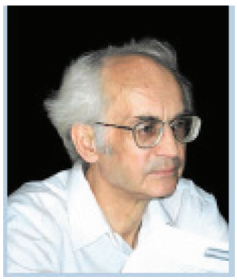
[31] Namiot, D., & Sneps-Sneppe, M. (2014). On Micro-services Architecture. International Journal of Open Information Technologies, 2(9), 24–27.

[32] Bahrami, M., & Singhal, M. (2015). The Role of Cloud Computing Architecture in Big Data. In Information Granularity, Big Data, and Computational Intelligence (pp. 275–295). Springer International Publishing.

Biographies



D. Namiot is a senior researcher at the Faculty of Computational Mathematics and Cybernetics Lomonosov Moscow State University. Dr. Dmitry Namiot received a Ph.D. degree in Computer Science of the Lomonosov Moscow State University for his work in the area of artificial intelligence. His research activity is in the field of mobile computing, context-aware programming, Smart Cities applications and open interfaces for telecom applications. He is author or co-author of over 90 journals and conferences articles and 4 books. Dr. Namiot won several awards for mobile development on 3GSM World contests, as well as several international Java Developers challenges awards. He is the editor of International Journal of Open Information Technologies (INJOIT).



M. Sneps-Sneppe is Leading Researcher at the Ventspils University College, Latvia. His research focus is on telecommunications software development. His hobby relates to Russian history. He is (co-)author of 11 books and many papers.