On IoT Programming

Dmitry Namiot, Manfred Sneps-Sneppe

Abstract—**This paper provides an overview of challenges for Internet of Things programming. In this article we discuss system software models and solutions, rather than network related aspects. It continues our series of publications about M2M systems, existing and upcoming system software platforms for M2M applications. We discuss here such issues for IoT systems as latency, power limitation, reliability (unreliability), network topology related effects as well as data processing in IoT applications.**

Keywords—**IoT, communications, software standards, micro-service, middleware.**

I. Introduction

In this paper, we would like to discuss some challenges associated with Internet of Things (IoT) programming. In this article we discuss system software models and solutions, rather than network related aspects. This paper continues our series of publications about software aspects of M2M and IoT.

In our first paper [1] we discussed the problems with the unified standards with Machine to Machine communications (M2M). We concluded that the current development misses the larger point of how M2M services and products get created and deployed. In many cases, developers either have to use some predefined platform and be locked with its restriction or build a system completely from scratch. For M2M and Internet of Things products to be successful, interfaces must be simple. The complexity that lies underneath should be completely hidden. As seems to us, at the current stage the existing solutions very often just increase the complexity.

The complexity of existing approaches we've discussed also in our paper [2]. It raises the following question: 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. This simple step (refusal to support API) can seriously simplify the interaction with the devices. DPI's are much simpler, of course. And what is more important – they can create a unified API for all devices. The process of reading data can be similar for all devices. As usual, we can pass data interpretation (translation) to the end-user devices. And our "unified" reading procedure can simply return some JSON array.

So, as soon as all the "unified" standards become too complex, what is the solution? We are strong proponents of micro-services.

The micro-services approach is a relatively new term in software architecture patterns. The micro-service architecture is an approach to developing an application as a set of small independent services [3]. Each of the services is running in its own independent process. Services can communicate with some lightweight mechanisms (usually it is something around HTTP) [4]. Such services could be deployed absolutely independently. Also, the centralized management of these services is a completely separate service too. It may be written in different programming languages, use own data models, etc. We think that micro-services are the natural fit for M2M (IoT) development.

In accordance with this, in our opinion, considering the individual systems, such as Open IoT [5], for example, a description of their abilities cannot be the main purpose. The main point is the allocation of micro-services within them. And the second goal is, accordingly, the issues of their independent usage and deployment. Such an analysis with respect to M2M applications was presented in our paper [6].

IoT and M2M have remote device access in common. But they are not completely similar, of course. Some of authors draw the difference in the way IoT and M2M access to the remote devices. For example, traditional M2M solutions typically rely on point-to-point communications using embedded hardware modules and either cellular or wired networks. In contrast, IoT solutions rely on IP-based networks to interface device data to a cloud or middleware platform. It is probably now always true, because the cloud is not a mandatory stuff for the Internet of Things. Nothing prevents the application access to remote devices directly, or, more precisely, get data from remote devices without the cloud (and without the middleware, by the way). The typical examples are Bluetooth Low Energy tags, mentioned in [2]. Some authors point to UI (User Interfaces). Obviously, the UI is a mandatory part for IoT projects and could be missed in M2M. This definition evolves into a more radical statement: M2M is simply a part for IoT. For our programmers-oriented (data access oriented) review this latest definition is, probably, most suitable.

The rest of the paper is organized as follows. In Section II we discuss challenges for IoT programming. In Section III we discuss perspective programming models for IoT applications.

Article received Sep 20, 2014.

D.Namiot is senior researcher at Open Information Technologies Lab, Lomonosov Moscow State University. Email: dnamiot@gmail.com

M. Sneps-Sneppe is with ZNIIS. Email: sneps@mail.ru

II. On challenges for IoT programming

As the first challenge for the system development in IoT area, we should mention the power supply. Obviously, it is the first limitation. It directly affects the algorithms we can use in our systems. So, solutions (e.g., libraries) for implementing power-optimized calculations (algorithms) will prevail. The same is true for network protocols.

We should mention in this context such entity as Dynamic Power Management (DPM). The main idea behind this approach is to shut down devices when they don't need to be on-line on and to start them up when they need to transmit (receive) data. As per [8], Dynamic power management (DPM) is a design methodology for dynamically reconfiguring systems to provide the requested services and performance levels with a minimum number of active components or a minimum load on such components.

Normally, it is a typical task for the operating system (OS). E.g., a mobile operating system can prefer accelerometer over GPS for some tasks due to energy limitations, etc. But complex IoT may orchestrate several devices, and any individual operating system is simply unaware about the whole process. So, the whole system should be able to switch services on and off more intelligently than each individual device's OS.

But of course, DPM itself is not free and may cause such a problem as latency. The latency could be of course a congenital problem for IoT devices too. E.g. device may transmit data in discrete time cycles only. The typical example is the above mentioned BLE tags (iBeacons).

Another typical source of delays is very often the network topology optimized for IoT system. For example, mesh networks are immune to the failure of a few nodes [9]. But as a price for this we will have more hops (read – increased delay) in data delivery paths. Actually, the scalability for IoT networks is a big problem. The things could be more complicated if will admit the fact that many devices may simply transmit data without requests (e.g., do that by the timer). It could lead to the wasted bandwidth and increased delays in communications.

In general, for many cases we have to consider IoT data as unreliable. It may lead to the additional data curation and error-correction procedures on the application level [10].

The data curation and data brokering stuff is very important for IoT applications by the another reason also. Actually, remote devices (sensors) in case of IoT can produce a huge amount of data. And it is very important to have the ability for data projection. We need to select the right amount of data for the particular task. And one of the biggest problem here is to find a right (and commonly used) tool just for data description. Raw data from sensors should have some meta-data associated with them. Otherwise, there is no way to develop adaptive algorithm. As soon as the mapping for data is unknown, we cannot automatically detect the dependencies for example. And this information is critical for many algorithms.

Figure 1 illustrates the basic data model behind FI-WARE project [11]

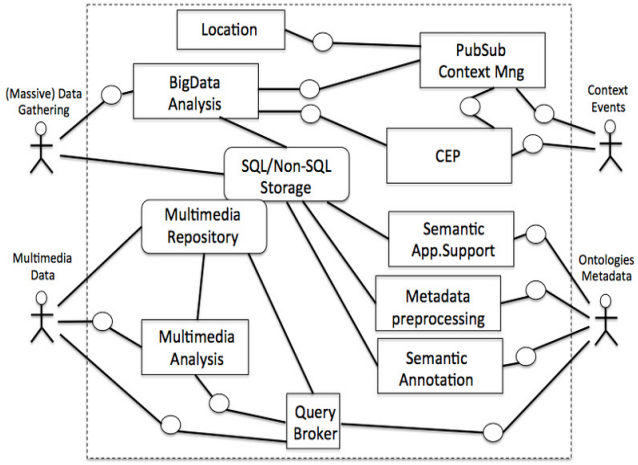

Figure 1. FI-WARE data model

Obviously, remote devices (sensors) may generate a big amount of data. So Big Data approach is a natural fit for IoT. But in case of a huge amount of distributed data developers need a way for real time processing some sub-sets. Think, for example about processing sensors data for some limited retail space. So, there is a huge demand for some kind of toolchains. Current IoT architectures are device or network oriented. However, the key value proposition of IoT is from the interaction of these "Things" with humans and society. So, for getting the benefits some form of stream processing for IoT data is practically mandatory.

It the terms of context-aware computing ("ubiquitous computing") IoT makes the software context much larger. So, the developed applications should have some mechanisms for dealing with this fast changed data.

III. On programming models

Lets us see some programming models that could be suitable for IoT.

Reactive programming (functional reactive programming - FRP) [12] is a paradigm for programming hybrid systems (systems containing a combination of both continuous and discrete components) in a high-level, declarative way. The key ideas in FRP are its notions of continuous, time-varying values, and time-ordered sequences of discrete events. The most important concept underlying functional reactive programming is that of a signal: a continuous, time-varying value. That is, a value of type Signal is a function mapping suitable value of time to a value of a given type.

Conceptually, then a signal's value at some time t is just a value for this functional mapping. Being able to define and manipulate continuous values in a programming language provides great expressive power. Figure 2 describes the reactive traits.

Figure 2. The reactive traits [13]

The next interesting concept is Abstract Task Graph [14]. The Abstract Task Graph (ATaG) is a data driven programming model for end-to-end application development of networked sensor systems. An ATaG program is a system-level, architecture-independent specification of the application functionality. ATaG model maps the network graph to an application graph.

ATaG provides a methodology for architecture-independent development of networked sensing applications. Architecture independence here is the ability to specify application behavior for a generic and parameterized network architecture. The same application may be automatically adopted for the different network deployments. Application will work as nodes fail or are added to the system. Furthermore, it allows development of the application to proceed prior to decisions being made about the final configuration of the nodes and network. Figure 3 describes ATaG program for environment monitoring [14].

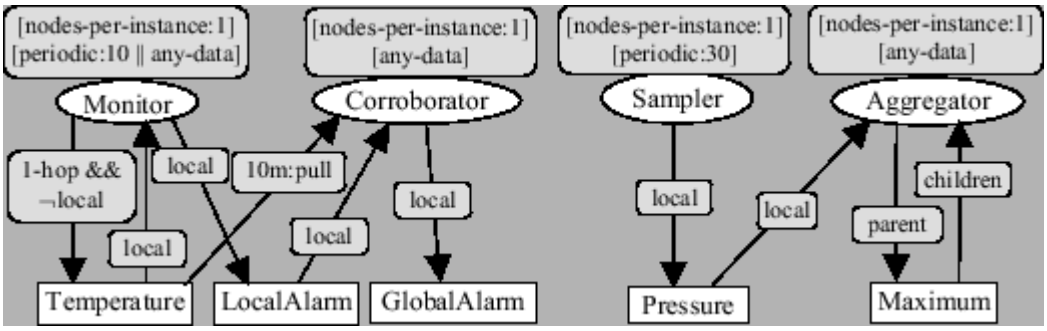

Figure 3. An ATaG program for environment monitoring [14].

As the next model we would like to discuss the Computational REST [15]. In this model the traditional content resources are replaced with computational resources. The key moments behind the Computational REST are:

- Computations and their expressions are explicitly named.

- Services may be exposed through a variety of URLs which offer perspectives on the same computation.

- Interfaces may offer complementary supervisory functionality such as debugging or management.

- Functions may be added to or removed from the binding environment over time or their semantics may change.

- Computations may be stateful and stateless.

- Potentially autonomous computations exchange and maintain state.

- A rich set of stateful relationships exist among a set of distinct URLs.

- The computation is transparent and can be inspected, routed, and cached.

- The migration of the computation to be physically closer to the data store is supported thereby reducing the impact of network latency.

In this context we should mention also an interesting model CoReWeb [16]. It presents a web of linked computational resources.

And at the end, we will describe Flow-Based Programming (FBR) [17] and the Actor Model [18]. Both models are based on components where the messages are the only entities which can affect processes. FBR is actually very close to the extensions of M2M API proposed in our paper [19]. Also Actors are very close to the basic primitives for micro-services [3]:

We can mention the following primitives need for micro-services architecture [15]:

1) Request/Response calls with arbitrary structured data

2) Asynchronous events should be flowing in real-time in both directions

3) Requests and responses can flow in any direction,

4) Requests and responses and can be arbitrarily nested. The typical example is a self-registering worker model

5) A message serialization format should be pluggable. So, developers may use, for example, JSON, XML, etc.

REFERENCES

[1] Namiot, D., & Sneps-Sneppe, M. (2014). On M2M Software. International Journal of Open Information Technologies, 2(6), 29-36.

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

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

[4] Uckelmann, Dieter, Mark Harrison, and Florian Michahelles. "An architectural approach towards the future internet of things." Architecting the internet of things. Springer Berlin Heidelberg, 2011. 1-24.

[5] Kim, J., & Lee, J. W. (2014, March). OpenIoT: An open service framework for the Internet of Things. In Internet of Things (WF-IoT), 2014 IEEE World Forum on (pp. 89-93). IEEE.

[6] Namiot, D., & Sneps-Sneppe, M. (2014). On M2M Software Platforms. International Journal of Open Information Technologies, 2(8), 29-33.

[7] Alam, M., Nielsen, R. H., & Prasad, N. R. (2013, July). The evolution of M2M into IoT. In Communications and Networking (BlackSeaCom), 2013 First International Black Sea Conference on (pp. 112-115). IEEE.

[8] Benini, L., Bogliolo, A., & De Micheli, G. (2000). A survey of design techniques for system-level dynamic power management. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 8(3), 299-316.

[9] Akyildiz, I. F., Wang, X., & Wang, W. (2005). Wireless mesh networks: a survey. Computer networks, 47(4), 445-487.

[10] Tyagi, Sapna, Ashraf Darwish, and Mohammad Yahiya Khan. "Managing Computing Infrastructure for IoT Data." Advances in Internet of Things 2014 (2014).

[11] Elmangoush, A., Al-Hezmi, A., & Magedanz, T. (2013, December). Towards Standard M2M APIs for Cloud-based Telco Service Platforms. In Proceedings of International Conference on Advances in Mobile Computing & Multimedia (p. 143). ACM.

[12] Hudak, P., Courtney, A., Nilsson, H., & Peterson, J. (2003). Arrows, robots, and functional reactive programming. In Advanced Functional Programming (pp. 159-187). Springer Berlin Heidelberg.

[13] The reactive manifesto http://www.reactivemanifesto.org/ Retrieved: Sep, 2014

[14] Bakshi, A., Prasanna, V. K., Reich, J., & Larner, D. (2005, June). The abstract task graph: a methodology for architecture-independent programming of networked sensor systems. In Proceedings of the 2005 workshop on End-to-end, sense-and-respond systems, applications and services (pp. 19-24). USENIX Association.

[15] Erenkrantz, J. R. (2009). Computational REST: A New Model for Decentralized, Internet-Scale Applications DISSERTATION (Doctoral dissertation, University of California, Irvine).

[16] Monnin, A., Delaforge, N., & Gandon, F. (2012, June). CoReWeb: From linked documentary resources to linked computational resources. In Proceedings of the WWW2012 Conference Workshop PhiloWeb 2012:“Web and Philosophy, Why and What For.

[17] Morrison, J. P. (1994). Flow-based programming. In Proc. 1st International Workshop on Software Engineering for Parallel and Distributed Systems (pp. 25-29).

[18] Esposito, A., & Loia, V. (2000). Integrating concurrency control and distributed data into workflow frameworks: an actor model perspective. In Systems, Man, and Cybernetics, 2000 IEEE International Conference on (Vol. 3, pp. 2110-2114). IEEE.

[19] Sneps-Sneppe, M., & Namiot, D. (2012, April). About M2M standards. M2M and Open API. In ICDT 2012, The Seventh International Conference on Digital Telecommunications (pp. 111-117).