On M2M Software

Dmitry Namiot, Manfred Sneps-Sneppe

Abstract — **This paper provides an overview for existing and upcoming system software projects in M2M area. In this article we discuss system software models and solutions, rather than network related aspects. The primary goal is to provide an overview of existing models as well as discuss the possible extensions. Can we describe the common points for the different M2M software models? Are there some reused patterns? How M2M software models are going to attract developers? These are the main issues addressed in this article.**

***Keywords*— M2M, communications, Smart metering, middleware, FI-WARE.**

I. Introduction

Machine-to-Machine (M2M) is a category of Information and Computing Technology that combines communications, computer and power technologies that enable remote iterations with physical, chemical and biological systems and processes [1]. Simply, M2M traditionally refers to technologies that allow both wireless and wired systems to communicate with other devices of the same ability. M2M uses a device (such as a sensor or meter) to capture an event (such as temperature, inventory level, etc.), which is relayed through a network (wireless, wired or hybrid) to an application (software program), translates the captured event into meaningful information [2].

As per widely used classification scheme, the M2M system consists of three main domains: M2M Device, Network, and Application Domain and contains the following key elements [3]:

• M2M Device. A device capable of replying to requests for data contained within those devices or capable of transmitting data contained within those devices autonomously.

• M2M Area Network. These networks provide connectivity between M2M Devices and M2M Gateways. Examples of M2M Area Networks include: Personal Area Network technologies such as IEEE 802.15, ZigBee, Bluetooth; and local networks such as PLC, M-BUS, and Wireless M-BUS.

• M2M Gateway. It uses M2M capabilities to ensure that M2M Devices interwork and interconnect to the communications networks.

• M2M Communications Networks. These are the communications networks between M2M Gateways and M2M Applications (servers). They can be further broken down into Access, Transport and Core Networks. Examples include: xDSL, PLC, satellite, LTE, GERAN, UTRAN, W-LAN, and WiMAX.

• M2M Application (Server). This is the middleware layer where data goes through the various application services and is used by the specific business processing engines.

It is illustrated in Figure 1.

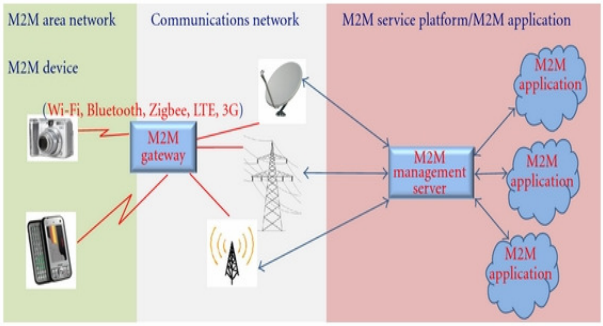


Figure 1. M2M Architecture [4]

It is, probably, the most elaborated software architecture for M2M area. As you can see, most of the elements target the network component. The last one only (application server) is, formally, the software. We would like to discuss it in this paper, as well as in the forthcoming articles. Do we always need such a component? What kind of tasks could be solved in this layer? Is it always a real software layer or just a virtual group of functions? In many cases, as we see, application server in M2M applications is the specification of functionality, rather than the separated code.

II. M2M System Applications

Figure 2 demonstrates the high level M2M architecture from ETSI [5]. The high level architecture for M2M includes a Device and Gateway Domain and a Network domain. Actually, that schema is suggested by ETSI, but quite general and can be used for describing other frameworks too.

The Device and Gateway Domain are composed of the following elements: M2M Device, M2M Area Network and M2M Gateway. The M2M Gateway acts as a proxy for the Network Domain towards the M2M Devices that are connected to it.

The Network Domain is composed of the following elements: Access Network, Core Network, M2M Service Capabilities and M2M applications.

Article received May 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 Institute of Mathematics and Computer Science, University of Latvia. Email: sneps@mail.ru

M2M Service Capabilities:

- Provide M2M functions that could be shared by different Applications.

- Expose functions through a set of open interfaces.

- Use Core Network functionalities.

- Simplify and optimize application development and deployment through hiding of network specificities.

Actually the developer's APIs are here. M2M applications run the service logic and use M2M Service Capabilities accessible via an open interface.

The goals for M2M middleware are obvious. M2M middleware helps us with heterogeneity of M2M applications. Heterogeneity of service protocols inhibits the interoperation among smart objects using different service protocols and/or API's. We assume that service protocols and API's are known in advance. M2M API provides the abstraction layer necessary to implement the interactions between devices uniformly. The work "abstraction" here point exactly to the above-mentioned virtual application server. It is just a convenient form which lets us describe the functionality.

The M2M API provides the means for the device to expose its capabilities and the services it may offer, so that remote machines may utilize them. Consequently, such an API is necessary to enable proactive and transparent communication of devices, in order to invoke actions in M2M devices and receive the relating responses as well as the simplified management of resources [6].

ETSI TR 102 691 [7] is, probably, the most elaborated document in ETSI's suite. It describes the following required areas to M2M applications:

Management - specifies requirements related to the management modes (malfunction detection, configuration, accounting, etc.).

Functional requirements for M2M services - describes functionalities-related requirements for M2M (data collection & reporting, remote control operations, etc.).

Security - covers the requirements for M2M device authentication, data integrity, privacy, etc.

Naming, numbering and addressing - provides the requirements relating to naming, numbering and addressing schemes specific to M2M.

As something significant in this part we can highlight probably the list of potential new requirements to M2M systems (devices) listed here.

- A M2M device should be able to register its capability information (e.g. access technology, its serial number, its accessible address, allowed user list, etc.) to the M2M System.

- M2M devices and M2M gateways should be able to perform access control that checks the access right of the end-user.

- M2M devices should be able to communicate either directly or via M2M gateway.

- M2M devices should be alternatively able to perform the access control of M2M devices.

- M2M devices and M2M gateways should be able to manage the scheduling of multiple accesses that multiple remote parties (i.e. end-users, M2M devices or M2M applications in M2M network, etc.) try to access one M2M device or one M2M gateway simultaneously.

As seems to us, this statement is very important: "register its capability information". We think, that there is definitely a demand for some analogues of SNMP management, where capabilities could be defined in the abstract terms (like MIB – management information blocks) [8]. Technically, SNMP could run on top of the 6LoWPAN layer, but it would be inefficient for the low power nodes that are used in M2M networks. There are many papers, devoted to ontology usage in M2M (and especially IoT) applications [9][10], but as seems to us it is too far from practice yet.

ETSI devotes the special direction to Automotive Applications [11]. As per ETSI, M2M automotive applications encompass M2M use cases involving the automotive or transportation industries where the involved M2M communication modules may be embedded into a car or transportation equipment, for whatever purpose. This implies common requirements such as mobility management and environmental hardware constraints, despite the extended variety of applications addressed (insurance or road pricing, emergency assistance, fleet management, electric car charging management, traffic optimization, etc.). The new requirements listed here are:

• the capability of M2M Devices to receive, store, and execute scheduled measurements;

• the ability of Devices to poll and check for the occurrence of events;

• the capability of Devices to autonomously establish a connection directly to a mobile telecommunication network;

• the capability for Devices to be able to maintain M2M communications while moving at high velocity and over a wide geographic area;

• the ability of devices to be able to be contacted ("called") directly by a mobile telecommunication network

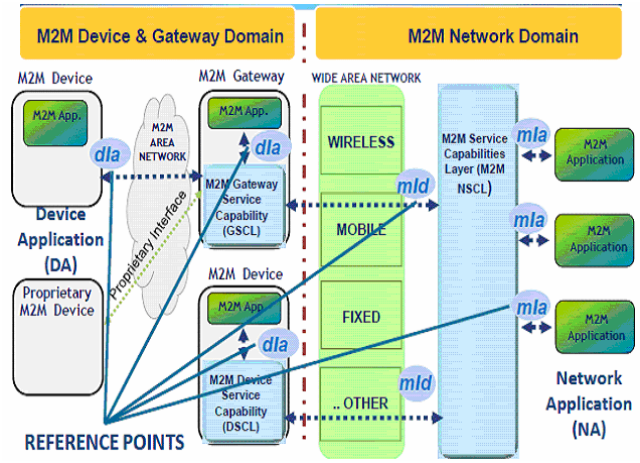
• the inclusion of position-determination capability.

By our opinion, such a division is one of the weakest points in the whole ETSI approach. From the developer's point of view, it would be better to have a small unified schema for all aspects. With this direction, ETSI approach potentially leads to the huge set of different APIs. We saw already the similar approach in Parlay [12] for example. It complicates the adoption of new development tools or even makes it impossible.

Indeed, in the Open API from ETSI we can see the big influence of Parlay specification. The goals are obvious, and they are probably the same as for any unified API. One of the main challenges in order to support the easy development of M2M services and applications will be to make M2M network protocols "transparent" to applications. Providing standard interfaces to service and application providers in a network independent way will allow service portability.

At the same time an application could provide services via different M2M networks using different technologies as long as the same API is supported and used. This way an API shields applications from the underlying technologies, and reduces the efforts involved in service development. Services may be replicated and ported between different execution environments and hardware platforms.

This approach also lets services and technology platforms to evolve independently. A standard open M2M API with network support will ensure service interoperability and allow ubiquitous end-to-end service provisioning.



The Open API relates to several interfaces of M2M architecture (Figure 2). For example:

Figure 2. Interfaces from ETSI [11]

- the interface between the platform and external service providers running their services remotely,

- the interface between the platform and the customer applying the features offered by the platform,

- a set of interfaces supporting additional functionality (installation support, access to remote databases, remote operation and management of the platform), etc.

Figure 2 shows the following interfaces: mIa – machine to application interface, mId – Machine to Device interface, dIa – device to application interface. ETSI proposes that every device needs a service capability based a REST server platform.

Table 1 summarizes Open API categories (with some our remarks).

Main API sections of Services Capabilities Level are:

- Subscription and Notification (e.g. Publish/Subscribe).
- Grouping and Transactions.
- Application Interaction: Read, Do, Observe.
- Compensation (micro-payment).
- Sessions.

Table 1. ETSI Open API Categories.

ETSI Open API categories	API contents	Comments
Grouping	A group here is defined as a common set of attributes (data elements) shared between member elements. In practice it is about the definition of addressable and exchangeable data sets.	Just note, as it is important for our future suggestions, there are no persistence mechanisms for groups.
Transactions	Service capability features and their service primitives optionally include a transaction ID in order to allow relevant service capabilities to be part of a transaction. Just for the deploying transactions and presenting some sequences of operations as atomic.	In the terms of transactions management Open API presents the classical 2-phase commit model. By the way, we should note here that this model practically does not work on the large-scale web applications. We think it is very important because without scalability we cannot think about "billions of connected devices".
Application Interaction	The application interaction part is added in order to support development of simple M2M applications with only minor application specific data definitions: readings, observations and commands.	Application interactions build on the generic messaging and transaction functionality and offer capabilities considered sufficient for most simple application domains.
Messaging	The Message service capability feature offers message delivery with no message duplication. Messages may be unconfirmed, confirmed or transaction controlled.	The message modes supported are single Object messaging, Object group messaging, and any object messaging; (it can also be Selective object messaging). Think about this as Message Broker.
Event notification	The notification service capability feature is more generic	It is a generic form. So, for example, geo fencing

and presence	than handling only presence. It could give notifications on an object entering or leaving a specific group, reaching a certain location area, sensor readings outside a predefined band, an alarm, etc.	should fall into this category too. The subscriber subscribes for events happening at the Target at a Registrar. The Registrar and the Target might be the same object. This configuration offers a publish/subscribe mechanism with no central point of failure.
Compensation	Fair and flexible compensation schemes between cooperating and competing parties are required to correlate resource consumption and cost, e.g. in order to avoid anomalous resource consumption and blocking of incentives for investments. The defined capability feature for micro-payment additionally allows charging for consumed network resources.	It is very similar, by the way, to Parlay offering for Charging API.
Sessions	In the context of OpenAPI a session shall be understood to represent the state of active communication between Connected Objects.	OpenAPI is REST based, so, the endpoints should be presented as some URI's capable to accept (in this implementation) the basic commands GET, POST, PUT, DELETE (See an example below).

The example below illustrates the typical request-response cycle:

URI: http://{nodeId}/a/do

Method: POST

Request

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<appint-do-request xmlns="http://eurescom.eu/p1957/openm2m">

<requestor>9378f697-773e-4c8b-8c89-27d45ecc70c7</requestor>

<commands>

<command>command1</command>

<command>command2</command>

</commands>

<responders>9870f7b6-bc47-47df-b670-2227ac5aaa2d</responders>

<transaction-id>AEDF7D2C67BB4C7DB7615856868057C3</transaction-id></appint-do-request>

Response

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<appint-do-response xmlns="http://eurescom.eu/p1957/openm2m">

<requestor>9378f697-773e-4c8b-8c89-27d45ecc70c7</requestor>

<timestamp>2010-04-30T14:12:34.796+02:00</timestamp>

<responders>9870f7b6-bc47-47df-b670-2227ac5aaa2d</responders>

<result>200</result>

</appint-do-response>

Let us describe the proposed standards from the modern web development point of view. We think it is correct, because Open API declares REST support right for the web development. In other words, support for web developers as the first class citizens is one of the obvious goals for ETSI. The history of this approach is described in [13].

What could be suggested in this connection? On the first hand, it is JSON vs. XML. It looks like JSON is the prevailed format for data exchange in the modern web development. The second position is asynchronous communication. Keeping in mind the growing role of JavaScript, the ideal interface should provide a callback-based model for communications. The application should post requests to the device and define some callback function that will accept JSON data upon request completion. So, as per our vision, the deployment of the server-side solution should include the following steps:

- define the contact point (define callback URL via x-etsi-contactURI header)
- perform the request
- proceed callbacks (HTTP requests) via a callback URL

But it means that we will need to prepare a CGI script for the each callback processing. From the other side, why shall we ignore the client side processing? The good candidates for client side processing were Web Intents. Web Intents enable rich integration between web applications. Increasingly, services available on the web have a need to pass rich data back and forth as they do their jobs. Web Intents facilitate this interchange while maintaining the kind of loose coupling and open architecture that has proven so advantageous for the web. They reside purely client-side, mediated through the User Agent, allowing the user a great degree of control over the security and privacy of the exchanged data [14].

Any Intent is a user-initiated action delegated to be performed by a service. It consists of an "action" string which tells the service what kind of activity the user expects to be performed (e.g. "share" or "edit"), a "type" string

which specifies the data payload the service should expect, and the data payload itself. So, we can replace callbacks (URLs) in the Open API with JavaScript actions.

Web Intents are extensible by design. Neither the list of actions, nor the list of media types is fixed. That is why intents can play an important role for semantic web too [15].

Intents play the very important role in Android Architecture. Intents bind individual components to each other at runtime (you can think of them as the messengers that request an action from other components), whether the component belongs to your application or another.

Going into M2M applications, it means that our potential devices will be able to present more integrated data for the measurement visualization for example. The final goal of any M2M based application is to get (collect) measurements and perform some calculations (make some decisions) on the collected dataset. We can go either via low level APIs or use (at least for the majority of use cases) some integrated solutions. The advantages are obvious. We can seriously decrease the time for development.

We can re-phrase an original idea of Web Intents. M2M data logging application should be aware of a user's preferred editing Web application, rather than enforcing the specific one that the data logging happens to be integrated with. Web Intents put the user in control of service integrations and makes the developers life simple. It is based on the well-known concept of callbacks. Each callback (response) returns JSON (not XML!) formatted data. As per suggested M2M API we should perform several individual requests, parse XML responses for the each of them and only after that do some visualization. Additionally, web intents based approach is asynchronous by its nature, so developers do not need to organize their own asynchronous schemes.

Also Web Intents approach lets us bypass sandbox restrictions. In other words, developers can raise requests right from the end-user devices, rather than always call the server. The server-side only solution becomes bottleneck very fast. And vice-versa, client side based request lets developers deploy new services very quickly. For example, right from mobile web browser. Why do not use the powerful browsers in the modern smart-phones? At the end of the day Parlay spec were born in the time of WAP and dumb phones. Why do we ignore HTML5 browsers and JavaScript support in the modern phones?

Also, as it is shown above, this approach automatically introduces JSON versus XML communications. Again, JSON (and especially JSONP) is a preferred format for web development and should be welcomed by programmers.

III. FI_WARE PROJECT

The most interesting from the developer's point of view is FI-WARE project [16]. FI-WARE will deliver a novel service infrastructure, building upon elements (called Generic Enablers) which offer reusable and commonly shared functions, making it easier to develop Future Internet Applications in multiple sectors – building a true foundation for the Future Internet.

The project will develop public and royalty-free Open Specifications of Generic Enablers, together with a reference implementation of them available for testing. This way, it is aimed to develop working specifications that influence Future Internet standards. FI-WARE is the cornerstone of the Future Internet Public Private Partnership (PPP) Program, a joint action by the European Industry and the European Commission.

The FI-PPP follows an industry-driven, user-oriented approach that combines R&D on network and communication technologies, devices, software, service and media technologies; and their experimentation and validation in real application contexts. The platform technologies will be used and validated by many actors, in particular by small- and medium-sized companies and public administrations. FI-WARE architecture is shown in Figure 3.

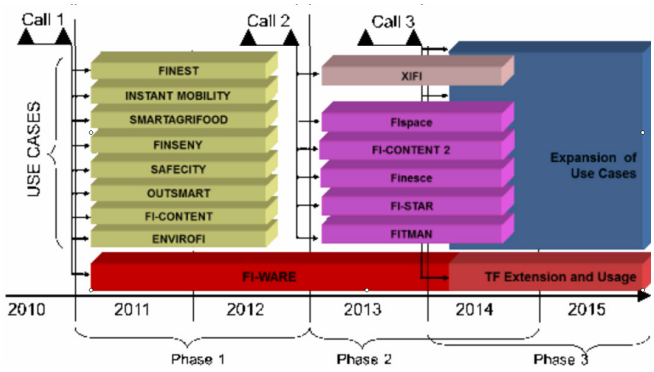


Figure 3. FI-PPP Programme Architecture

There are more than 60 FI-WARE Generic Enamblers (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. 17 Specific Enablers relate to the OUTSMART project (Smart City project) but only a few are implemented by now. Let us name the most interesting between them:

- CKAN – an open data platform software, where data are securely extracted from a SCADA/production system.
- Service Information Repository – aims at providing the possibility to search/retrieve and store the information about services, is used in Santander and Birmingham services.

A lot of work should be done on these Specific Enablers for use as software standards for Smart City projects [17].

The high-level architecture illustrated in Figure 4 is structured according to the key business roles and their relationships within the overall service delivery framework and existing IT landscapes. The applications and service delivery framework comprises the internal key business roles:

Broker - supports exposing services from diverse providers into new markets, provides a monetization infrastructure.

Hoster – allows representing the different cloud hosting providers.

Aggregator - supports domain specialists and third-parties in aggregating services and apps for new and unforeseen opportunities and needs.

Gateway – supports Providers and Aggregators in selecting a choice of solutions that may provide interoperability, as a service, for their applications.

Channel Maker – provides support for creating outlets through which services are consumed: Web sites/portals, social networks, mobile channels and work centers, through which application/services are accessed.

Consumer – completes the service supply chain.

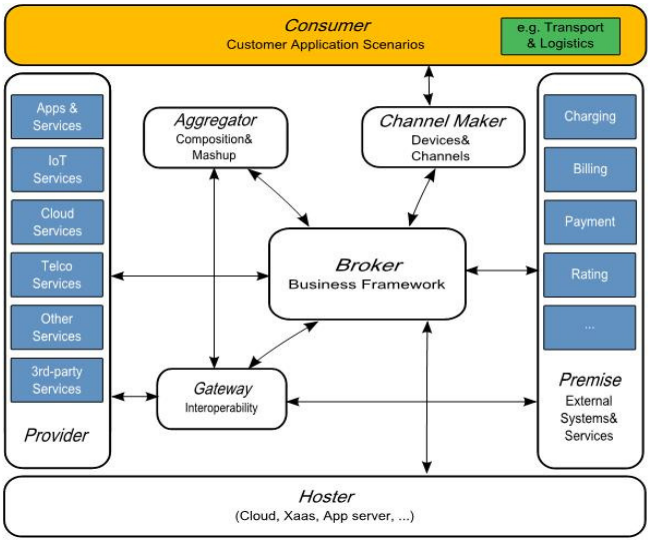


Figure 4. FI-WARE high level architecture

The Reference Architecture of the FI-WARE platform [18] is structured along a number of technical chapters, namely:

- Cloud Hosting,
- Data/Context Management,
- Internet of Things (IoT) Services Enablement,
- Applications/Services Ecosystem and Delivery Framework,
- Security and
- Interface to Networks and Devices (I2ND).

IV. FI-WARE DATA MODEL

As per official document, FI-WARE will enable smarter, more customized/personalized and context-aware applications and services by the means of a set of assets able to gather, exchange, process and analyze massive data in a fast and efficient way (Figure 5).

Data in FI-WARE refers to information that is produced, generated, collected or observed that may be relevant for processing, carrying out further analysis and knowledge extraction. A basic concept in FI-WARE is that data elements are not bound to a specific format representation. Actually the whole data model in FI-WARE has been described by the concept of NoSql [19] systems in mind. Data items could be named and presents themselves by just named collection of triples: <name, type, value>.

What is important, that, optionally, data elements could have meta-data (descriptions) associated with them. Meta-data elements could be described via collections of triples <name, type, value> too. The data-model described in FI-WARE could be actually perfectly supported by distributed key-value systems [20].

The context in FI-WARE is represented through context elements. A context element extends the concept of data element by associating an *EntityId* and *EntityType* to it, uniquely identifying the entity (which in turn may map to a group of entities) in the FI-WARE system to which the context element information refers. In addition, there may be some attributes as well as meta-data associated to attributes that we may define as mandatory for context elements as compared to data elements [21].

It is very important that FI-WARE actually uses the same model for data and meta-data. It means that from the developer's point of view, it should be possible to use the same model for persistence and search for both data and meta-data.

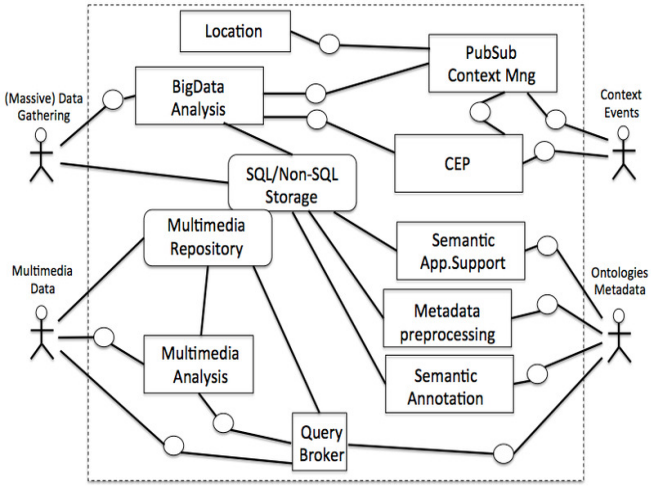


Figure 5. FI-WARE data model

An event in FI-WARE is an act of creating a new element. It could be either data event (create data elements) or context event (creates a context element). As an example, a sensor device measures some value and periodically creates and sends a new context element. The creation and sending of the context element is an event. Because each event has got either data or context elements linked to, the whole system can see events via linked data. It makes the whole system much more uniform (homogeneous) comparing with M2M approach described below.

At this moment we have a wide choice for real-time analytical systems based on key-value stores. For example, we can mention Google Percolator [22] or Twitter Storm [23]. It is exactly the approach needed for processing data in FI-WARE model.

For event publishing FI-WARE roadmap suggests

ContextML [24] and SPARQL [25]. *ContextML* is a light-weight XML based context representation schema in which context information is categorized into scopes and related to different types of entities (e.g. user, device). The schema is also applied for encoding management messages in order to allow for a flexible framework supporting gradual plug & play extendibility and mobility. *ContextML* is tailored to be used for REST-based communication between the framework components.

RDF is a directed, labeled graph data format for representing information in the Web. And SPARQL specification defines the syntax and semantics of the query language for RDF. SPARQL can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. SPARQL also supports extensible value testing and constraining queries by source RDF graph. The results of SPARQL queries can be results sets or RDF graphs.

The obvious candidates here are standards activities from The Open Geospatial Consortium (OGC) that focus on sensors and sensor networks comprise [26]. On the first hand it is Observations & Measurements Schema (O&M) as well as Sensor Model Language (*SensorML*), Transducer Model Language (*TransducerML* or *TML*), Sensor Observations Service (SOS), Sensor Planning Service (SPS) and Sensor Alert Service (SAS).

For example O&M supports data sampling as this:

```
<gml:description>
  Observation test instance: fruit mass
</gml:description>
 <gml:name>Observation test 1</gml:name>
  <om:phenomenonTime>
 <gml:TimeInstant    gml:id="ot1t">
<gml:timePosition>
2005-01-11T16:22:25.00
</gml:timePosition>
  </gml:TimeInstant>
 </om:phenomenonTime>
  <om:parameter>
<om:NamedValue>
<om:name
xlink:href="http://sweet.jpl.nasa.gov/ontology/property.owl
#Temperature"/>
 <om:value xsi:type="gml:MeasureType" uom="Cel">
</om:value>
</om:NamedValue>
</om:parameter>
```

But keeping in mind the modern trend in web development – shall we keep that as XML, or it is a time to replace it with an appropriate JSON?

FI-WARE proposes also an interesting approach for Applications/Services Ecosystem and Delivery Framework. It is based on the heavy usage on USDL [27]. Universal Service-Semantics Description Language (USDL) can be used by service developers to specify the formal semantics of web-services. Thus, if WSDL can be regarded as a language for formally specifying the syntax of web services, USDL can be regarded as a language for formally specifying their semantics. USDL is as formal service documentation that will allow sophisticated conceptual modeling and searching of available web-services, automated composition, and other forms of automated service integration. For example, the WSDL syntax and USDL semantics of web services can be published in a directory which applications can access to automatically discover services. We target some data models in our paper [28].

V. Conclusion

We think 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. The main problems are not devices. The main question is service. As seems to us, at the current stage the existing solutions very often just increase the complexity. There are too many telecom-related issues and too few data processing issues. The true developers-oriented stack for M2M is yet to be created.

References

[1] Brazell, J. B., Donoho, L., Dexheimer, J., Hanneman, R., & Langdon, G. (2013). M2M: the wireless revolution.

[2] Schneps-Schneppe, M., & Namiot, D. (2013). Machine-to-Machine Communications: the view from Russia. International Journal of Open Information Technologies, 1(1), 1-5.

[3] Wu, G., Talwar, S., Johnsson, K., Himayat, N., & Johnson, K. D. (2011). M2M: From mobile to embedded internet. Communications Magazine, IEEE, 49(4), 36-43.

[4] Tan, Siok Kheng, Mahesh Sooriyabandara, and Zhong Fan. "M2M communications in the smart grid: Applications, standards, enabling technologies, and research challenges." International Journal of Digital Multimedia Broadcasting 2011 (2011).

[5] "ETSI Machine-to-Machine Communications info and drafts" http://docbox.etsi.org/M2M/Open/ Retrieved: May, 2014.

[6] M. Sneps-Sneppe, D.Namiot "About M2M standards and their possible extensions" Future Internet Communications (BCFIC), 2012 2nd Baltic Congress on, 25-27 April 2012 pp. 187-193 DOI: 10.1109/BCFIC.2012.6218001

[7] Galetić, V., Bojić, I., Kušek, M., Jezic, G., Desic, S., & Huljenic, D. (2011, May). Basic principles of Machine-to-Machine communication and its impact on telecommunications industry. In MIPRO, 2011 Proceedings of the 34th International Convention (pp. 380-385). IEEE.

[8] Starsinic, M. (2010, May). System architecture challenges in the home M2M network. In Applications and Technology Conference (LISAT), 2010 Long Island Systems (pp. 1-7). IEEE.

[9] Jeon, P. B., Kim, J., Lee, S., Lee, C., & Baik, D. K. (2011, August). Semantic negotiation-based service framework in an M2M environment. In Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology-Volume 02 (pp. 337-340). IEEE Computer Society.

[10] Gronbek, I., & Biswas, P. K. (2009, October). Ontology-based abstractions for M2M virtual nodes and topologies. In Ultra Modern Telecommunications & Workshops, 2009. ICUMT'09. International Conference on (pp. 1-8). IEEE.

[11] ETSI TR 102 898 V0.4.0, "M2M Communications; Use cases of Automotive Applications in M2M capable networks".

[12] Moerdijk, A. J., & Klostermann, L. (2003). Opening the networks with Parlay/OSA: standards and aspects behind the APIs. Network, IEEE, 17(3), 58-64.

[13] Y. Daradkeh, D. Namiot, M. Sneps-Sneppe "M2M Standards: Possible Extensions for Open API from ETSI". European Journal of Scientific Research, vol. 72, N. 4, pp. 628-637

[14] http://www.w3.org/TR/2012/WD-web-intents-20120626/ Retrieved: May, 2014

[15] R. Verborgh, et al "Functional descriptions as the bridge between hypermedia APIs and the Semantic Web". WS-REST '12 Proceedings of the Third International Workshop on RESTful Design, pp. 33-40

[16] Usländer, T., Berre, A. J., Granell, C., Havlik, D., Lorenzo, J., Sabeur, Z., & Modafferi, S. (2013). The future internet enablement of the environment information space. In Environmental Software Systems. Fostering Information Sharing (pp. 109-120). Springer Berlin Heidelberg.

[17] Namiot, Dmitry, and Manfred Schneps-Schneppe. "Smart Cities Software from the developer's point of view." arXiv preprint arXiv:1303.7115 (2013).

[18] Robles, T., González-Miranda, S., Alcarria, R., & Morales, A. (2012). Web browser HTML5 enabled for FI services. In Ubiquitous Computing and Ambient Intelligence (pp. 181-184). Springer Berlin Heidelberg.

[19] J. Pokorny "NoSQL databases: a step to database scalability in web environment". Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services, ACM, New York, USA, 2011, pp. 278-283

[20] A. Lakshman, P. Malik "Cassandra: a decentralized structured storage system" ACM SIGOPS Operating Systems Review archive, Volume 44. Issue 2, April 2010, pp. 35-40.

[21] http://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/Data/Context_Management_Architecture Retrieved: May, 2014

[22] D. Peng, and F. Dabek "Large-scale incremental processing using distributed transactions and notifications". In OSDI '10, 2010, pp. 251–264.

[23] Yang, W., Liu, X., Zhang, L., & Yang, L. T. (2013, July). Big Data Real-Time Processing Based on Storm. In Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on (pp. 1784-1787). IEEE.

[24] M.Knappmeyer, et al "ContextML: A light-weight context representation and context management schema". Wireless Pervasive Computing (ISWPC), 5th IEEE International Symposium on Date of Conference: 5-7 May 2010, pp. 367 – 372.

[25] Quilitz, B., & Leser, U. (2008). Querying distributed RDF data sources with SPARQL. In The Semantic Web: Research and Applications (pp. 524-538). Springer Berlin Heidelberg.

[26] M. Botts, et al "OGC Sensor Web Enablement: Overview and High Level Architecture". In: GeoSensor Networks. Lecture Notes in Computer Science, Volume 4540/2008, pp.175-190, DOI: 10.1007/978-3-540-79996-2_10

[27] Kona, S., Bansal, A., Gupta, G., & Hite, T. D. (2006, June). Web service discovery and composition using USDL. In E-Commerce Technology, 2006. The 8th IEEE International Conference on and Enterprise Computing, E-Commerce, and E-Services, The 3rd IEEE International Conference on (pp. 65-65). IEEE.

[28] Schneps-Schneppe, M., Maximenko, A., Namiot, D., & Malov, D. (2012, October). Wired Smart Home: energy metering, security, and emergency issues. In Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2012 4th International Congress on (pp. 405-410). IEEE. DOI: 10.1109/ICUMT.2012.6459700