Synchronization Abstractions and Separation of Concerns as Key Aspects to the Interoperability in IoT

Marcio Ferreira Moreno[1(✉)], Renato Cerqueira[1], and Sérgio Colcher[2]

[1] IBM Research, Av. Pasteur, 138 and 146 – Botafogo, Rio de Janeiro (RJ), Brazil
{mmoreno,rcerq}@br.ibm.com

[2] Department of Informatics – PUC-Rio, Rua Marques de São Vicente, 225 – Gávea, Rio de Janeiro (RJ) 22451-900, Brazil
colcher@inf.puc-rio.br

Abstract. In this paper we argue that synchronization abstractions could be used as the glue that tie together the interactions between 'things' in an IoT environment. We also support that this is analog to what is used in distributed multimedia applications. Using this argument, we propose in this paper that IoT solutions, protocols and applications should benefit from standardized multimedia tools like specification languages and corresponding middleware support platforms as a means for harmonization and interoperability. Additionally, we extend our recent contributions in favor of a separation of concerns in multimedia systems, in which synchronization support can operate independently of other features. More specifically, the main contribution of this paper is the discussions about how media synchronization challenges can enroll the Internet of Things research area, where distributed sensors and actuators are specified as media objects and can be related to usual hypermedia objects, all synchronized in time and space, in what we call the "Synchronism of Things".

Keywords: Internet of Things · Synchronism of things · Interoperability · Ginga · NCL

1 Introduction

IoT has emerged as a promise for a completely new ecosystem made for the interconnection of "things" that will open the doors for emerging and compelling applications like smart cities, smart grids, health and fitness wearable devices and agro-business sensor powered equipment that could revolutionize productivity. IoT can be thought as representing a new wave on Internet evolution technologies, that brings not only Machine-to-Machine (M2M) communications to the world of interconnected people and business processes, but also inspires new thoughts over the meaning of what is interconnection itself.

To reach the full potential of the IoT, however, it is not sufficient for things to just be connected to the Internet; they also need to be found, accessed, managed and potentially connected to other things. To enable this interaction, a degree of

N. Mitton et al. (Eds.): InterIoT 2016/SaSeIot 2016, LNICST 190, pp. 26–32, 2017.
DOI: 10.1007/978-3-319-52727-7_4

interoperability is necessary that goes beyond simple protocol interoperability as provided by the Internet [1].

Standard organizations and community forums have been working towards reference models, architectures and specific standards for different parts or levels within those models in order to bring some structure to the chaos, trying to lessen the gap between the different vertical domains and help industry not to jump into proprietary solutions. From the network point of view, protocol interoperability is the main focus, and organizations like IEEE and ITU-T have been working very hard trying to overcome the challenges of bringing together efficient protocols like the various low power networking protocols (ZigBee, ZWave, and Bluetooth), traditional networking protocols (Ethernet, WiFi) and new technologies (5G) [2]. The IETF community has also been involved in foundational IoT technologies such as IPv6 and the Constrained Application Protocol (CoAP) focusing on getting constrained devices and sensor networks connected to the Internet [3].

But, as Blackstock and Lea state [3], before addressing interoperability, there must be some agreement on what interoperability means, and about the degree of interoperability required, as well as on its implications for IoT system and application developers.

Actually, we could think that application level interoperability is also desirable. In fact, the Internet of Things Architecture project (IoT-A) is proposing an architectural reference model for IoT interoperability, along with key components that deals with application level issues like search, discovery, and interaction between things [3]. But there is also a multiplicity of competing application level protocols such as CoAP (Constrained Application Protocol), MQTT (Message Queue Telemetry Transport) and XMPP (Extensible Messaging and Presence Protocol) that have been proposed by various organizations to become the *de facto* standard to provide communication interoperability, each of which with unique characteristics that happens to be adequate for different types of IoT applications. However, as pointed out by Desai et al. [2], a scalable IoT architecture should be independent of messaging protocol standards, while also providing integration and translation between various popular messaging protocols. Moreover, while exchanging information by messages in an efficient way is an important requirement at this level of abstraction, the synchronization that should be obtained when things are engaged in these communication processes seems to be left aside.

In this paper we argue that synchronization abstractions should be treated as the glue that tie together all the interactions between things. We also support that this is not much different than what is used in distributed multimedia applications. Using this argument, we propose that IoT applications should benefit from standardized multimedia tools like specification languages and corresponding middleware support platforms as a means for acquiring interoperability.

The key issue in a multimedia system is the support for temporal and spatial synchronization among media assets, in its broad sense. In this work, we extend our recent contributions in favor of a separation of concerns in multimedia systems, in which synchronization support can operate independently of other features. More specifically, the main contribution of this paper is the discussions about how media synchronization challenges can enroll the Internet of Things research area, where distributed sensors and actuators are specified as media objects and can be related to usual hypermedia objects, all synchronized in time and space, in what we call the "Synchronism of Things" (SoT).

2 Separation of Concerns: Isolating the Synchronization Support

The evolution of multimedia applications has continuously created new challenges for systems that support media synchronization. Besides the development of new communication technologies and the advances in computational resources, high-quality media objects and new compression techniques led to novel requirements for media synchronization. Multi-sensorial media (mulsemedia) presentations has also introduced different requirements [4], where media objects that state traditional visual content types (i.e. text, images, and video) can be related with media objects that target other human senses (e.g. olfactics, haptics etc.).

As introduced in Sect. 1, in the SoT perspective, the new requirements go beyond the use of multi-sensorial media to enhance user Quality of Experience, as usual in mulsemedia. Indeed, the idea is to introduce how media synchronization challenges enrolls the Internet of Things research area, where different distributed sensors and actuators are specified as media objects and related to usual objects, all synchronized in time and space.

The separation of concerns in this paper extrapolates our recent contributions [5–7], arguing in favor of architectures of multimedia systems being divided in two modules at least: one to support the synchronization of media assets, controlling the logic of the execution; other in charge of transporting and handling the backend mechanisms of media assets. In this paper, the former is called Execution Manager and the latter Multithing Backend.

In the separation of concerns, the Execution Manager can operate independently of the underlying backend features, making it possible for developers or end-users to add and update features (for instance, having new logics for sensors or actuators added as plug-ins) dynamically without needing to make changes to the host Manager (host). Open application programming interface allows third parties to create plug-ins that interact with the host application. A stable plug-in API allows both third-party plug-ins to continue functioning as the original version changes and to increase the lifetime of obsolete applications.

These concepts are essential for coexistence and interoperability of different IoT solutions and protocols, and were applied in our previous work. More specifically, in the specification language named NCL and in its middleware Ginga [7]. Indeed, NCL has a strict separation between application content and application structure. NCL does not define any media itself. Instead, it defines the glue that relates media objects in time and space. NCL documents (NCL application specifications) only refer to media content. Any media object has a set of properties and some content. Content can be the logic of sensor/actuator, samples of video, audio, images and text, or any code chunk in some specification language. Properties are usually related to the content, like the positioning of a mechanical arm, the format of data being sensed, and others.

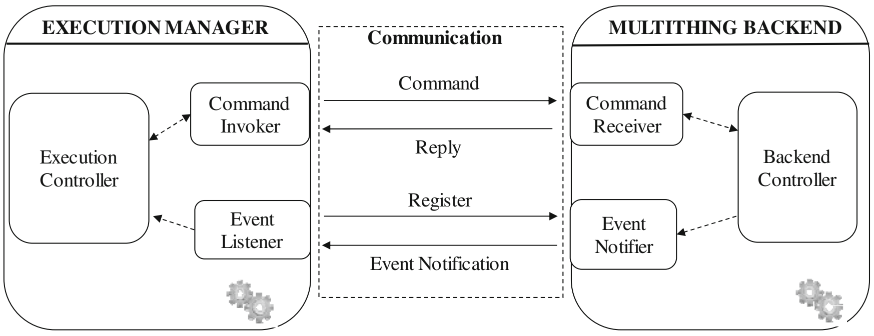

Fig. 1. Separation of concerns in multimedia synchronism support.

Figure 1 illustrates the proposed separation of concerns. The Execution Manager is responsible for reading the document specification and for building an execution plan, using a sub-module named Execution Controller. The Multithing Backend transport, decodes, processes, and handles media content, using a sub-module called Backend Controller.

Nevertheless, this separation of concerns goes beyond modularization. Those two main modules are deeply separated from each other, by means of process isolation. For this reason, they also include sub-modules that provide inter-process communication. The Execution Manager defines the Command Invoker and the Event Listener. The Multithing Backend defines the Command Receiver and the Event Notifier.

The modules communicate either (i) to handle the execution of commands or (ii) to handle the occurrence of events (e.g. temporal events, user-generated events). The design of the first type of communication resembles the Distributed Command design pattern, while the second type resembles the Distributed Observer design pattern [5, 8].

The Command Invoker submodule is responsible for sending commands to the Multithing Backend. These commands can either request the execution of an action or query the value of a variable (that is, a media object or system property).

The Multithing Backend receives commands through the Command Receiver. This submodule parses the requests to check whether they are valid and forwards them to the Backend Controller. The Command Receiver may send a reply message, notifying whether the Multithing Backend was able to meet the request or not.

The Event Listener is responsible for receiving event notifications. First, it registers itself as a listener (observer) to be notified of events during the presentation. Upon receiving an event notification, this submodule converts the notification into the data structure expected by the Presentation Controller and delivers the event.

Finally, the Event Notifier is in charge of notifying the registered observers about events. The event types notified depend on the implementation. To reduce the message traffic, this module can implement a filtering approach, in which the observers inform upon registration the types of events they can handle. The Event Notifier would then send notifications only when there is a match between the event type and the filters of an observer.

3 The Synchronism of Things with an NCL Application

In NCL every <media> element can have <area> and <property> child elements. An <area> element defines a subset of information units of some media-object content. Thus, an <area> element can define an interval of time, a subset of data in a sensor or a text string. Since media objects can also contain code chunks, an <area> element can also delineate a code chunk; for example, a function of the application, coded in the media object content.

The <property> element defines the name attribute, which indicates the name of a property or of a property group of its parent media object, and the value attribute, an optional attribute that defines an initial value for the name property. Property can define where and how content resulting from the media object processing is executed or presented. In media objects with imperative code content, the <property> element can also refer to a specific code chunk through its name attribute, in which case the value attribute has input parameters to be passed to the code chunk.

Figure 2 depicts a very simple NCL application illustrating how media objects encapsulate concepts of new media types and how the NCL execution engine is in charge of orchestrate the execution of documents with these different media assets that can be of different IoT standards and protocols. NCL defines the <ncl> root element and its child <head> and <body> elements, following the terminology adopted by W3C standards to structure documents. The <body> element includes <port>, <media>, and <link> child elements. The <port> elements externalize interfaces of child media-objects of a composition (the <body> element in this case). When an NCL application is started without specifying a <port>, the execution of every media object associated to every <port> element is started. There are three <media> elements in the example. All of them use remote node.js[1] entry points to communicate with remote content, which must be processed by a plug-in whose location is www.ginga.ncl.org.br/plugins. The plug-in use the IoT communication infrastructure of IBM (e.g. IoT Foundation, Watson IoT Platform, and REST & Real-time APIs)[2] to exchange data between sensors and actuators. Finally, three <link> elements establish spatial and temporal relationships among the media objects.

The Execution Manager of Ginga starts the application with the presentation of the NCL object “oil”: a player that uses a sensor designed to extract oil parameters during a distillation process. If the sensor detects the end of the process, the player notifies the end of the “oil” component execution to its parent controller (the Execution Manager of Ginga). At this moment, the condition of <link id = “l2”> is satisfied, starting the “highlight” object. The player of this object is a cognitive computing system designed to extract and register the highlights of the distillation process.

During the “oil” object presentation, if the sensor detects changes in the “viscosity” parameter, the node.js player (plug-in) notifies the parent Execution Manager the start-attribution-event occurrence. As a consequence, the condition of the <link id = “l1”> is satisfied, setting the “visbreaker” property of the “distillation” media

[1] https://nodejs.org/.

[2] https://internetofthings.ibmcloud.com.

```
<?xml version="1.0" encoding="UTF-8"?>
<nclid="SoT" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">

  <head>
  ...
  </head>

  <body>
<portid="entryPoint" component="oil"/>

  <mediaid="oil"src="nodejs/oilsensor" descriptor="oDesc"
player="www.ginga.ncl.org.br/plugins">
<propertyvalue="viscosity"/>
</media>

  <mediaid="highlights"src="nodejs/highlights" descriptor="hDesc"
player="www.ginga.ncl.org.br/plugins"/>

  <mediaid="distillation"src="nodejs/oilactuator" descriptor="dDesc"
player="www.ginga.ncl.org.br/plugins">
<propertyname="visbreaker"/>
</media>

  <linkid="l1"xconnector="onEndAttributionSet">
<bindrole="onEndAttribution" component="oil" interface="viscosity"/>
<bindrole="set" component="distillation" interface="visbreaker"/>
<bindParamname="var" value="$getValue"/>
</bind>
<bindrole="getValue" component="oil" interface="viscosity"/>
</link>

  <linkid="l2"xconnector="onEndStart">
<bindrole="onEnd component="oil"/>
<bindrole="start" component="highlights"/>
</link>

  </body>
</ncl>
```

Fig. 2. NCL example: a domain-specific language with synchronization abstractions to promote interoperability between IoT sensors and actuators.

object to the current value of "viscosity". When the "distillation" plug-in receives the "viscosity" value, it executes the node.js code, the "visbreaker" method, passing the value as a parameter. As a consequence, the actuator coupled with the "distillation" player executes the corresponding operation.

4 Final Remarks and Future Directions

Standard organizations and community forums have been working towards reference models, architectures and specific standards for different parts or levels within those models in order to bring some structure to the chaos, trying to lessen the gap between the different vertical domains and help industry not to jump into proprietary solutions. The main contribution of this paper is to present synchronization abstractions in form of a specification language and the separation of concerns to isolate key-issues in architectures as a solution to the coexistence and interoperability challenge in IoT. The separation of concerns combined with the definition of an API for plug-ins allows the system evolution without the need of modifying its synchronization support. More important, the model specifies calls that enables handling synchronization issues. Among other benefits, Ginga plug-in API enables the presentation of new types of media, allowing for applications that can synchronize sensors, actuators, and the usual media objects, in time and space. The drawback of this approach is to have all IoT mechanisms centralized in the plug-ins. To address this issue, we intent to study how NCL and Ginga functionalities are related to the ones present in Node-RED[3]. Node-RED is an authoring tool to specify data flows relating IoT devices, APIs and online services. Another future direction we aim to pursue is the use of knowledge engineering and cognitive computing agents in the description of NCL applications. We argue that this could bring the description of NCL-IoT applications to another level, allowing, for instance, the use of semantics on sensors and actuators as well as the use of cognitive computing analysis over sensed data. Our recent works [9–11] consist in a first step in this direction.

References

1. Blackstock, M., Lea, R.: IoT interoperability: a hub-based approach. In: 2014 International Conference on the Internet of Things (IoT), Cambridge, MA, USA, pp. 79–84. doi:10.1109/IOT.2014.7030119
2. Desai, P., Sheth, A., Anantharam, P.: Semantic gateway as a service architecture for IoT interoperability. In: 2015 IEEE International Conference on Mobile Services (MS), New York City, NY, USA, pp. 313–319. doi:10.1109/MobServ.2015.51
3. Blackstock, M., Lea, R.: Toward interoperability in a web of things. In: Mattern, F., Santini, S., Canny, J.F., Langheinrich, M., Rekimoto, J. (eds.) The 2013 ACM Conference, Zurich, Switzerland, pp. 1565–1574. doi:10.1145/2494091.2497591
4. Yuan, Z., Bi, T., Muntean, G.-M., Ghinea, G.: Perceived synchronization of mulsemedia services. IEEE Trans. Multimed. **17**(7), 957–966 (2015)
5. Moreno, M.F., Santos, R., Lima, G., Soares, L.F.G.: Deepening the separation of concerns in the implementation of multimedia systems. In: ACM SAC (2016)
6. Soares, L.F.G., Moreno, M.F., Marinho, R.S.: Ginga-NCL architecture for plug-ins. Softw. Pract. Exp. **43**(4), 449–463 (2013). doi:10.1002/spe.2144
7. Moreno, M.F., Soares, L., Cerqueira, R.: A component-based architecture for Ginga. In: 13th International Conference on Software Engineering Research and Practice, pp. 76–82
8. Soares, L.F.G.S., Moreno, M.F., Guedes, A.L.V.: Controlling the focus and input events in multimedia applications. In: ACM SAC 2015, Salamanca, Spain, 13–17 April 2015
9. Moreno, M.F., Brandao, R.R.M., Cerqueira, R.: Extending hypermedia conceptual models to support hyperknowledge specifications. In: IEEE ISM, San Jose, CA, USA (2016, in press)
10. Moreno, M.F., Brandao, R.R.M., Cerqueira, R.: Towards a conceptual model for cognitive-intensive practices. In: IEEE ISM, San Jose, CA, USA (2016, in press)
11. Moreno, M.F., Brandao, R.R.M., Cerqueira, R.: NCM 3.1: a conceptual model for hyperknowledge document engineering. In: ACM DocEng, Vienna, Austria (2016)

[3] http://nodered.org/.