22nd IMEKO TC4 International Symposium & 20th International Workshop on ADC Modelling and Testing
SUPPORTING WORLD DEVELOPMENT THROUGH ELECTRICAL&ELECTRONIC MEASUREMENTS
IASI, ROMANIA, September 14-15, 2017

Bringing executable choreographies to the IoT

Cosmin Ursache[1], Catalin Damian[2], Sinica Alboaie[3], Lenuta Alboaie[4], El Mehdi Stouti[5]

[1,3,4]*Alexandru Ioan Cuza University of Iasi, Faculty of Computer Science,Romania,*
cosmin.ursache86@gmail.com, salboaie@gmail.com, adria@info.uaic.ro
[2]*Technical University "Gheorghe Asachi" of Iasi, Faculty of Electrical Engineering, Romania,*
cdamian@tuiasi.ro
[5]*Essaadi University of Tetouan, Morocco, Romania, mehdy.info@gmail.com*

***Abstract* – The Internet of Things (IoT) is growing fast as popularity, demand, availability and in the same time too slow on some regards [1]. Many companies that have interest in IoT are trying hard to secure a good part of the market by forming different associations or by designing "closed" standards under the blanket of interoperability and collaboration. No matter what level of the IoT we analyse, there are already directions that guide how things should be made or how they should work.**
Fortunately for smaller companies driven by communities there are still good opportunities to collaborate between them and target a more important challenge: build sustained technology that can adapt and evolve in the rhythm that is need it.

Keywords – IoT, ZigBee, Swarm, NodeJs

I. INTRODUCTION. HETEROGENEITY, PRIVACY AND INTEGRATION IN IOT

The Internet of Things at current state is represented by a paradigm defined by many visions that sometimes are totally different as perspective and implementation [2]. There are many standards and visions for each layer of IoT: at hardware level, communication level and even at software architecture level [3]. For example, the communication layer is divided in two major directions regarding the protocol: Zigbee and Z-wave.

Home use IoT products market can be a good example of how different technical and business visions can make a difference in what companies can do and what final consumer can have. Zigbee and Z-wave projects are trying to become the baseline and standards for how the devices from an IoT network can and should communicate wireless between them.

Nevertheless there are open niches supported by open source and communities driven technologies built on top of Linux distros run on hardware that sometimes can do more diversified tasks. Few examples of this direction are represented by Android and Tizen OS that are open source Linux distributions build for a specific purpose. Android is supported by the Open Handset Alliance and Tizenis supported by the Linux Foundation.Tizen has also micro versions capable to run on wearable or small IoT devices. This means that is ways of developing new software, features or even improves on the existing platforms.

Beyond the need of integration and creation of complex processes between IoT devices belonging to a single organization or a single person, various IoT systems will require some levels of integration between them. An important aspect of IoT systems communicating across multiple organizations or individuals is security and privacy issues. In [4] was presented how different types of executable choreographies can provide supplementary data protection levels over classical communication architectures.

In this paper we present how the SwarmESB can handle the heterogeneity of current and future hardware used in home or industrial for IoT applications and integrate them into a unitary ecosystem. SwarmESB is a research project that is built with non-proprietary technologies and is driven by a strong community focused on privacy at all levels of IoT ecosystem and not only. In section 2 we present in more details the executable choreographies. In chapter 3 we created a short survey of the current IoT devices and protocols that highlights the heterogeneity of the market.

II. SWARM BASED EXECUTABLE CHOREOGRAPHIES

In this paper, by executable choreographies we understand the types of executable choreographies promoted through the SwarmESB open source project [5].

Modern intelligent systems integrate technologies, organizations and individuals to complete complex process controlled computing systems. From a technical point of view, the integration perspective is very important for intelligent systems.In the case of a high number of integration points, the process is solved by dedicated solutions types:ESB (Enterprise Service Bus), [6, 7], MOM (Message-Oriented Middleware), [8], EIP

(Enterprise Integration Patterns) based systems, [9, 10], or by service orchestration using programming languages or code adapted to business process modeling, [11-13].All of these methods are generally sufficient to integrate single organizationsystems. On the other hand, integration between multiple organizations should be addressed using choreographies, because any centralized solution is risky from the point of view of security and personal data protection. The composition of systems using orchestration tends to generate centralized systems.

Although many authors in the literature perceive choreographies as a limited mechanism to the formal description of contracts among a number of organizations [14, 15], academic research suggests the concept of executable choreographies [4, 5, 17-19].

It is thus suggested that each participating choreography organization to transcribe the descriptions of the choreographies into executable code.Thus, choreography becomes not only a formal description of a contract between organizations, but a description of an operative process in an executable form. The same sequence (choreography) will be run by several organizations, so the need to translate choreography into other programming languages disappears.

When speaking about executable choreographies, this paper refers to a platform inspired by his own research work to implement executable choreographies, namely SwarmESB. From this perspective, executable choreography can be considered a sequence of steps in a business process (or flow) that is run by multiple organizations in a decentralized manner (without the need for a central coordinator). During execution, communication between organizations is accomplished by asynchronous messages, so that the sequence of operative processes is formed only from a set of steps that combines a declarative style for the executable phases with a code that provides a programmable approach model.

Our previous works have proposed three types of executable choreographies. The detailed description of each type of choreography and the modalities of implementation and formalization will be the subject of future articles and reports. However, to ease the understanding of this paper, we will briefly present the essential characteristics of these three types of choreographies. These types of executable choreographies are general concepts that do not relate to specific software implementation.

A. Executable Choreographies Types

Verifiable choreographies: In our work we consider all executable choreographies to be verifiable at the same time. To check the privacy and fairness properties of executable choreographies it is need to develop methods and software tools to perform the verification. Verifiable choreographies aim only to highlight organizations and the type of private data that is consumed.

Encrypted choreographies: are based on the existence of control encryption keys, identification and authentication mechanisms that allow creating choreographies that legally secure encrypted data exchange between organizations. Compared to verifiable choreographies, encrypted ones provide a set of tools and self-implementation of choreographies that minimize private information sharing.Implementation ofencrypted choreography is based on existing systems data storage that uses encryption techniques specific so as to achieve practical implementation of partial homomorphicencryption [20-27].

Serverless choreographies: are cloud-based encrypted choreographies with automatic deployment mechanisms that guarantee or maximize ownership of "self-sovereignty" over private data.

B. High-level relationships between the elements of the choreography

The base of executable choreography implementation is represented by the following elements: locations (nodes or adapters), agents (mobile code executed on different locations) and actions.

The agents in order to perform actions need to visit processing nodes (nodes and adapters) in a specific location. Multiple nodes and adapters are grouped in organizations. All of the agent units (processes) with a specific target are grouped in swarms, the actions part of the swarm structure.

Fig. 1 illustrates the high-level relationships which exist among the elements of the choreography, focusing particularly on the relationships among units and the relationships among groups.

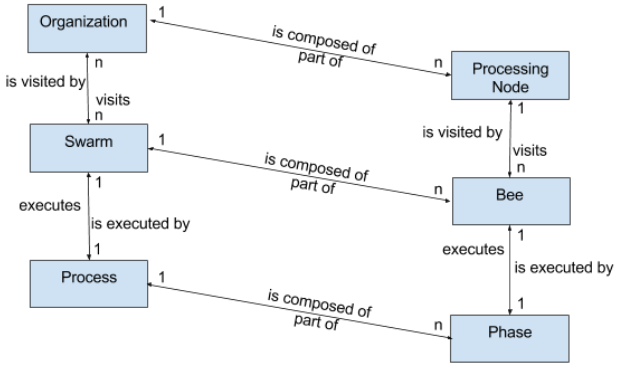

Fig.1. High-level relationships among the elements of the choreography

Organizations can be composed from multiple **nodes** used to process swarm phases. The **processing nodes** have specific type of responsibilities and actions that can be invoked once the swarm gets on node. Usually there are different types of nodes and all the nodes of the same type are offering the same API. For those who are trying

to use a specific node type the availability of different nodes that are offering the same API represents a real advantage. There is also a special type of nodes that are called **home nodes**. Those nodes are the ones that start a swarm.

In Swarm terminology, the messaging objects that have processing power are called **bees**. These can travel between nodes in order to perform specific actions available on nodes. All the code executed on a single visit of a bee on a specific processing node it is called **phase**.

A **process** in swarm perspective is represented by a collection of phases executed by a group of bees in order to achieve an objective. The description of swarm phases specifies: the instructions which must be executed by the current instance, the result which must be returned home by the current instance and the bees that must be cloned from the current instance including the node and all phases of each cloned instance.

The group of bees spanning between nodes and generations is called **swarm**.

C. General architecture for swarm based choreograophies

From the implementation point of view, the architecture of a system based on swarm executable choreographs provides the following components (Fig. 2):

- A message queue system. In the classic SwarmESB implementation, the PUB/SUB channels are provided by the NoSqlRed is database server;
- Processes of the operating system that are capable of running JavaScript code and exposing a set of functions (APIs) to choreographies. Typically, these processes are called **adapters** in SwarmESB terminology. This name is due to the fact that one of their main roles is to adapt the system's functionalities to the world of executable choreographies. Adapters are subscribed to queues, meaning they can receive or send messages to another adapter;
- A secure messaging system between organizations (called relay). Relay servers are simple https servers that enable secure messaging and file communication between organizations. Relays allow login requests from organizations that are involved in choreographies accepted by the current organization. Relay also subscribes to the organization's mail queue to communicate messages received from adapters from another organization to the local adapters, or to receive messages that they have to deliver to another organization.

III. EXECUTABLE CHOREOGRAPHIES FOR IOT

The SwarmESB porting to IoT systems reduces to the ability to run the code of executable choreographies in IoT nodes and the ability of IoT nodes to read messages from an accessible and optimized message queue to be used by IoT devices.

A. Node.js and IoT

Popularity of JavaScript on client side applications made possible the development of Node.js engine in order to use the same programming language and asynchronism mechanism on server side. Now the growing adoption rate of Node.js as the backbone for the development of many popular applications, simplicity and adaptability of the JavaScript is supporting the idea that JavaScript can accomplish tasks in embedded systems [28] opening a new world for JavaScript developers to explore.

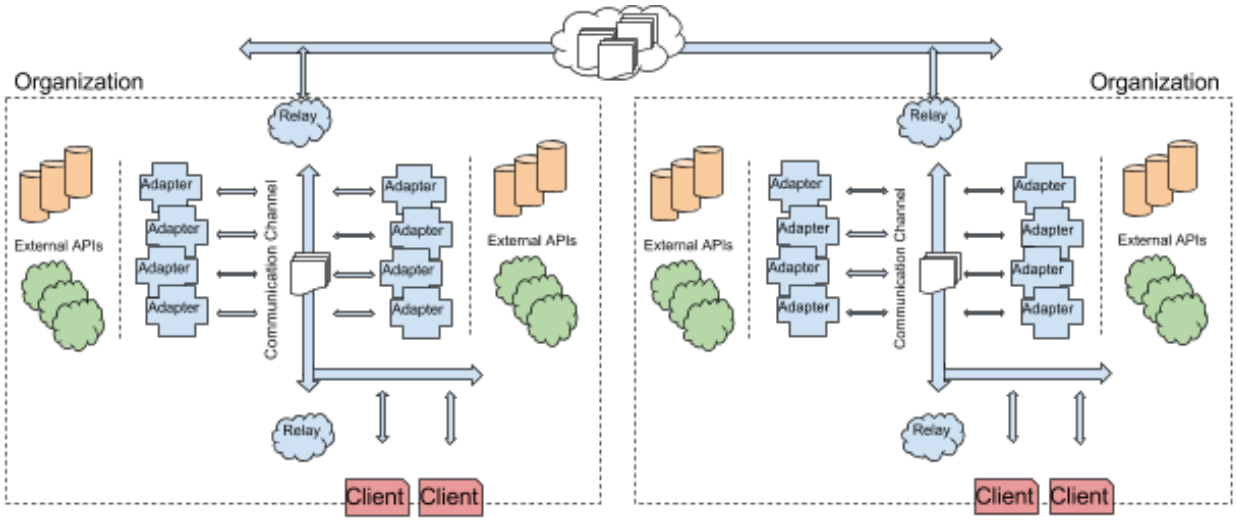

Fig. 2. General architecture for swarm based choreograophies

Basically a team of JavaScript developers can set the base of an embedded system with custom backend support and crazy interface written with a minimal set of tools and within the same JavaScript environment. Also there are many open source projects that are working hard to open the horizon of JavaScript and push it to the limit. The growth of public repositories (like NPM for example) that store and make sharing more easy of JavaScript and Node.js modules are also helping to make more public libraries that can expand the reach of the platform. Standards like Zigbee and Z-wave are also reachable from JavaScript using minimal hardware if need it in conjunction with specialized libraries like ZBee, Node-zigbee, OpenZWave, IOT.js and others.

B. JavaScript running on IoT hardware

Development boards for enthusiasts used to power dreams and test ideas are also capable to run a JavaScript engines [28]. This kind of boards are a good alternative because they are relative cheap, accessible and in the same time powerful enough to run a basic video streaming server, network video recorder, home automation solution, etc. There are many variations of enthusiastic boards built by companies like Intel, Arduino AG, Raspberry Pi Foundation and many others to support and develop amateur IoT projects.

Small footprint JavaScript engines can handle the memory size and performance of current microcontrollers. JerryScript [29]and Duktape [30] are very good options of JavaScript interpreters that can handle a lot more than their size. JerryScript is a lightweight JavaScript engine that was developed by Samsung in order to run on small microcontroller and power IoT devices. In big lines it supports on-device compilation, execution and provides access to peripherals covering pretty well the basics.DukTape is also a lightweight engine that can handle execution of JavaScript code and in the same time can expose C code to be called from JavaScript. In addition was developed DukLuv as separate project which has added libuv to the Duktape engine to produce something that works like Node.js for embedded devices.

C. Message queues for embedded systems

MQTT (MQ Telemetry Transport) is a lightweight messaging system between embedded equipment with simplicity in use, performance, reliability, low resource consumption and messaging quality. The equipment that communicates through MQTT infrastructure is divided into two categories: broker and client. Broker machines are the nodes responsible for transmitting messages between client equipment. Quality of Service (QoS) between clients connected to a broker is implemented using a 3-tier system. The first level is responsible for transmitting the message only once without confirming receipt, second level transmits the message at least once, and it is necessary to acknowledge reception and the last level is the one that sends the message only once and requires a change of 4 control messages to Confirms receipt[31]. In addition to the QoS implementation, the last message sent by an MQTT broker is retained so that any new client connected can be updated with the LKG (Last Know Good) information [32].

An important functionality when talking about integration with IoT equipment implemented by the MQTT protocol is "Last Will and Testament (LWT)". LWT makes it possible to detect IoT customers that disconnects for technical reasons and running a decision-making process at the level of broker MQTT equipment to handle the event [33]. Depending on the importance of the node within the organization, it can be from a simple logon or notification to the system administrator until the startup and reconnection processes of the node start.

For security reasons and pragmatic reasons for lack of performance, energy consumption, and the lack of libraries able to support the connection of IoT systems to Redis in an optimal way, our proposal is to look at the organization's IOT devices as separate nodes that communicate through tails of messages distinct from other systems of the organization. This approach involves distributing IoT devices around relay nodes capable of connecting to message tails designed for IoT devices. Our documentation and experiments suggest that the MQTT system is an appropriate solution when it comes to embed equipment and systems to replace Red is in the Swarm architecture. At the same time, MQTT's QoS, LKG and LWT deployments add value and increase the trustworthiness of the messaging system between organizations' nodes. Moreover, there are JavaScript libraries capable of facilitating MQTT communication such as MQTT.js

The MQTT.js provides a full-featured JavaScript library for the MQTT protocol. It is fully isomorphic, which means it can run in the browser and in Node.js starting with version 0.8 including development boards such as Intel Edison and Raspberry PIs. For example, it is bundled in the base image of Intel Edison [34]. It can support long running clients with weak connectivity. It has good on-disk storage for local offline messaging. Even if at the beginning it was under a "closed" license type now it is under MIT license type and available on GitHub [35].

IV. CONCLUSIONS

This paper presents the possibilities of implementing IoT executable choreographies, the minimum adaptation level of the SwarmESB project and the benefits to the IoT environment and to IoT applications by using the swarms. Starting from the current state of the IoT environment, the capacity and performance of the equipment present on the market, the interest of organizations to communicate, the need to ensure a high

level of privacy and security,we have demonstrated that the SwarmESB project is able not only to provide current needs but also to provide an environment conducive to the development of innovative solutions using a minimum of resources.

All the transition and preparation of the SwarmESB project for use in the IoT environment is supported by the development of numerous JavaScript interpreters specially designed for embedded equipment, by the many JavaScript libraries capable of facilitating communication in distributed environments and the use of wireless communication protocols and last but not least of technology used to develop the SwarmESB project.

The IoT environment is not one of the main areas covered by classical ESB-like systems, but rather part of a niche at least from the perspective of this type of system. As any new niche to be covered requires changes and adaptation to use in the IOT environment of the SwarmESB system, it causes minor changes in architecture, namely replacing the Redis message system with a more environmentally-friendly one, namely MQTT. In other words, Redis's exchange with MQTT makes it possible to use the system in the IoT equipment environment and facilitates the implementation and running of executable choreographies with all the advantages of this in a "hostile" environment like Internet of Things.

In the following papers we will present in more detail the results obtained and facilitated by SwarmESB in the environment of IoT equipment and systems.

REFERENCES

[1] Caraguay, A, Peral, A, et al.,*SDN: Evolution and Opportunities in the Development IoT Applications*, International Journal of Distributed Sensor Networks, vol 2014,Article ID 735142,January 2014.

[2] Atzori, L, Iera, A, Morabito, G,*The Internet of Things: A survey*, Comput. Netw. (2010), doi:10.1016/j.comnet.2010.05.010.

[3] Miorandi, D, Sicari, S, Pellegrini, F, Chlamta, I, *Internet of things: Vision, applications and research challenges*, Ad Hoc Networks, Elsevier, Volume 10, Issue 7, September 2012, Pages 1497–1516.

[4] Alboaie, L, Alboaie, S, Barbu, T, *Extending swarm communication to unify choreography and long-lived processes*, 23rd International Conference on Information Systems Development (ISD 2014), Varazdin, Croatia, September, 2-4, 2014.

[5] Alboaie, L, Alboaie, S, Panu, A, *Swarm Communication - A Messaging Pattern Proposal for Dynamic Scalability in Cloud*, 15th IEEE International Conference on High Performance Computing and Communications (HPCC 2013). IEEE, Zhangjiajie, China, November, 13-15, 2013, pp. 1930 - 1937.

[6] Chappell, D, *Enterprise service bus*, O'Reilly Media, Inc., ISBN 0-596-00675-6, June, 2004.

[7] David A. Chappell, *Enterprise Service Bus: Theory in Practice*, Publisher: O'Reilly Media, 2009.

[8] Curry, E,*Message-oriented middleware*, Middleware for communications, ISBN 978-0-470-86206-3, July 2004.

[9] Hohpe, G, Woolf, B, *Enterprise integration patterns: Designing, building, and deploying messaging solutions*, Addison-Wesley Professional, 2004.

[10] *Enterprise Integration Patterns* http://www.enterpriseintegrationpatterns.com/patterns/messaging/

[11] Ko, Ryan KL., *A computer scientist's introductory guide to business process management (BPM)*, Crossroads, 15(4), ACM Press, 2009.

[12] Silver, B,*BPM Method and Style*, 2nd Edition, with Bpmn Implementer's Guide: A Structured Approach for Business Process Modeling and Implementation Using Bpmn 2

[13] *WS-BPEL standard*, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf

[14] WSCDL Specification- https://www.w3.org/TR/ws-cdl-10/

[15] Peltz, C, *Web Services Orchestration and Choreography*, Journal Computer, IEEE Computer Society Press Los Alamitos, Volume 36 Issue 10, pp. 46-52, CA, USA, 2003.

[16] F. Akkawi, D.P. Fletcher, T. Cottenier, D.P. Duncavage, R.L. Alena, T. Elrad, An executable choreography framework for dynamic service-oriented architectures, IEEE Aerospace Conference, January 2006

[17] Besana, P, Barker, A, *An Executable Calculus for Service Choreography*, OTM Confederated International Conferences, On the Move to Meaningful Internet Systems, pp. 373-380, Springer Berlin Heidelberg, November 2009.

[18] Akkawi, F, Fletcher, D.P., Cottenier, T, Duncavage, D.P., Alena, R.L. and Elrad, T, *An executable choreography framework for dynamic service-oriented architectures*, Aerospace Conference, IEEE, 2006

[19] Alboaie, S, Nita, L, Stefanescu, C, *Executable Choreographies for Medical Systems Integration and Data Leaks Prevention*, The 5th IEEE International Conference on E-Health and Bioengineering - EHB 2015.

[20] Gentry, C, *Fully homomorphic encryption using ideal lattices*, Proceedings of the 41st Annual ACM Symposium on Theory of Computing, Bethesda, MD, May–June 2009.

[21] Erlingsson, U, Pihur, V, Korolova, A, *RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response*, proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS 2014). ACM, pp. 1054-1067.

[22] *Randomized Aggregatable Privacy-Preserving Ordinal Response*: http://dl.acm.org/citation.cfm?id=2660348

[23] Zyskind, G, Pentland N.A., *Enigma: Decentralized Computation Platform with Guaranteed Privacy*, 2015

[24] *Enigma*, http://enigma.media.mit.edu/

[25] Popa, R.A., Redfield, C, Zeldovich, N, *CryptDB: Protecting Confidentiality with Encrypted Query Processing*, Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP), Cascais, Portugal, October 2011.

[26] Popa, R.A., Li, F.H., Zeldovich, N, *Processing Analytical Queries over Encrypted Data*, Proceedings of the 39th International Conference on Very Large Data Bases (VLDB), Riva del Garda, Italy, August 2013.

[27] *CryptDB*, http://www.eecs.berkeley.edu/~raluca/CryptDB-sosp11.pdf

[28] Mulder, P, Breseman, K, Node.js for Embedded Systems: Using Web Technologies to Build Connected Devices, O'Reilly Media, October 2016.

[29] Gavrin, E, Lee, S., Ayrapetyan, R, Shitov, A, *Ultra lightweight JavaScript engine for internet of things*, Companion Proceedings of the 2015 ACM SIGPLAN International Conference on Systems, Programming,

Languages and Applications: Software for Humanity, October, 2015, pp. 19-20.

[30] Kim, M, Jeong, HJ, Moon, SM, *Small Footprint JavaScript Engine*, Components and Services for IoT Platforms, September 2016, pp. 103-116.

[31] *Technical documentation of MQTT*, http://mosquitto.org/man/mqtt-7.html

[32] *MQTT Version 3.1.1 Public Draft*, http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/csprd02/mqtt-v3.1.1-csprd02.html

[33] Webb, R, *A Brief, but Practical Introduction to the MQTT Protocol and its Application to IoT*, https://zoetrope.io/tech-blog/brief-practical-introduction-mqtt-protocol-and-its-application-iot/

[34] Tuan B, *Building and Running Mosquitto* MQTT on the Intel® Edison board*, https://software.intel.com/en-us/blogs/2015/02/20/building-and-running-mosquitto-mqtt-on-intel-edison

[35] *MQTT.js GitHub project page*, https://github.com/mqttjs/MQTT.js/blob/master/LICENSE.md