Deliverable reference:	Date:	Version:	Responsible partner:
D1.2	09.11.2017	V1.0	SINTEF

DiversIoT

Project co-funded by the Norwegian Research Council
IKTPluss

http://its-wiki.no/wiki/DiversIoT:Home

Title:

D1.2 Survey of IoT Diversity Enablers

Editor(s):	Approved by:
Brice Morin	Project Technical Manager: Franck Fleury
	Classification: Unrestricted

Abstract / Executive summary:

Diversity by itself is not a challenge for the IoT. The IoT is indeed a very diversified ecosystem by nature.

This survey gives an overview of diversity enablers that can be leveraged to build diversified IoT systems.

Document URL:	ISBN:

The Research Council of Norway
With funding from
The Research Council of Norway

Content

Version History **4**

1 Introduction **5**

2 The very diverse nature of the IoT **7**

3 Implementation and Behaviour Diversity **8**

3.1 PROGRAMMING LANGUAGES 8

3.2 SEMANTICALLY-EQUIVALENT IMPLEMENTATIONS 10

3.3 INFRASTRUCTURE & DISTRIBUTION 12

4 Communication Diversity **13**

5 Managing Diversity in Space and in Time **14**

6 Conclusion: Challenges & Roadmap for further research **15**

Table of Figures

NO TABLE OF FIGURES ENTRIES FOUND.

Version History

Version	Description	Date	Who
0.1	First complete version		Brice Morin

1 Introduction

The digitalized society, powered by the Internet of Things (IoT), will have enormous potential in terms of economic growth and improved quality of life. However, despite tremendous opportunities[1], many companies are currently risk assessing their IoT adoption, because evidence show that it can have pernicious effect on security, privacy and safety for both individuals and the society as a whole[2]. Even more so, since organizations face an increasing pressure to comply[3] with a growing number of regulatory requirements related to management, control and monitoring of privileged access to sensitive data11, which make organization liable as they have to "*protect it [personal data] from misuse and must respect certain rights of the data owners which are guaranteed by EU law*". Gartner reports that more stringent regulations will lead to a rise of 40% in penalties imposed by regulatory bodies because of privacy and security issues[4].

In nature, diversity has been a key mechanism for resilience for all forms of life, by enabling them to adapt to changes in the environmental conditions and evolve immunity to perturbations. More precisely in ecology, the *diversity–stability hypothesis* states that higher diversity within biological communities tends to increase resilience and reduces the risks of an ecological collapse[5]. Automatic software diversity is a promising solution, which is getting an increasing attention from the community[6]. However, because it goes against well-established software best practices, it is often impractical and current solutions are limited to low level mechanisms[7]. The scientific merit of this project is to challenge these well-established principles, introducing diversification allowing the development of more robust and trustworthy IoT systems. Recent advances in model compilation techniques offer new opportunities to apply diversification at a higher level of abstraction.

Recent multidisciplinary research on software engineering and ecology suggests that a promising way to approach resilience in software systems is to diversify software[8,9], implying that each instance of a service has a different implementation and operates differently, still ensuring that its global behaviour is consistent and predictable. Thus, instead of exposing the very same code in millions of instances, each individual instance will be unique, making the overall system more resilient. One example nowadays used by most operating system is the randomization/obfuscation

[1] Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015, see http://www.gartner.com/newsroom/id/3165317

[2] Gartner Says Worldwide IoT Security Spending to Reach $348 Million in 2016, see http://www.gartner.com/newsroom/id/3291817

[3] Protection of Personal Data, see http://ec.europa.eu/justice/data-protection/

[4] Gartner Research, Market Guide for Privileged Access Management, 2015

[5] Anthony R. Ives et al., «Stability and diversity of ecosystems,» Science 317, 58 – 62., 2007

[6] P. Larsen et al., «SoK: Automated Software Diversity,» i IEEE Symp. Security and Privacy, 2014.

[7] M. Stamp, «Risks of Monoculture,» Communication of the ACM, vol. 47, nr. 3, 2004.

[8] Laperdrix et al., «Beauty and the Beast: Diverting modern web browsers to build unique browser fingerprints,» in IEEE Symposium on Security and Privacy (S&P'16), 2016.

[9] F. Fleurey et al., «Multi-tier diversification in Web-based software applications,» IEEE Software, 32 (1), pp. 83-90, 2015.

of memory addresses[10], to prevent a number of exploits, making it difficult for a malware program to jump at an offset in the memory and access critical data.

[10] Bhatkar, Sandeep, Daniel C. DuVarney, and Ron Sekar. "Address Obfuscation: An Efficient Approach to Combat a Broad Range of Memory Error Exploits." *USENIX Security Symposium*. Vol. 12. No. 2. 2003.

2 The very diverse nature of the IoT

IoT systems are composed by a highly heterogeneous interconnection of platforms and devices offering a wide diversity of capabilities[11]. On the one end of the continuum, cloud platforms provide virtually unlimited and ”on-demand” resources in terms of computation power, storage and bandwidth. On the other end, the already vast and rapidly increasing number of smart objects, sensors, embedded systems and mobile devices connected to the Internet offers the connection to the users and to the physical world. These IoT devices:

- run on diverse platforms (ARM, MIPS, AVR, PIC)
- can be rather resource-constrained (as low as 2 KB RAM and 8 MHz CPU)
- or can be rather powerful (1GB RAM, 1GHz CPU)
- can run an OS (fully-fledged Linux for the most powerful, embedded Linux for the
- or can be too limited to even run an OS
- can communicate through different protocols: WiFi, Bluetooth, Bluetooth Low Energy, ZigBee, LoRa, Z-Wave, etc

While offering an immense potential for innovative IoT systems, the heterogeneity, diversity and vast distribution of the computing continuum represent daunting challenges[12]. Current software engineering approaches tend to provide support for managing and exploiting only parts of the continuum. For example, current cloud computing and service-oriented software engineering practices provide efficient abstractions for virtualizing the infrastructure in order for the software engineer to concentrate on the business logic of the applications. However these techniques merely support integration of mobile devices, sensors and actuators ”as-a-service” and lack specific support for advanced exploitation of these small devices.

Furthermore, although sensor network nodes, gateways, smart-phones, and most smart-objects are provided with APIs and development environment which allow advanced exploitation of their capabilities. There are no common practices to expose the level of flexibility, languages, APIs and customization supported by different devices and platforms at the level of what is provided in cloud and service oriented approaches. Thus, most pervasive systems tend to be proprietary silos not easily exploitable by other systems as they offer only black box provisioning of higher level services.

[11] Morin, Brice, Nicolas Harrand, and Franck Fleurey. "Model-Based Software Engineering to Tame the IoT Jungle." *IEEE Software* 34.1 (2017): 30-36.

[12] Morin, Brice, Franck Fleurey, and Olivier Barais. "Taming heterogeneity and distribution in scps." *Proceedings of the First International Workshop on Software Engineering for Smart Cyber-Physical Systems*. IEEE Press, 2015.

3 Implementation and Behaviour Diversity

3.1 Programming Languages

A typical IoT infrastructure spans across a continuum of devices and platforms ranging from microcontroller-based devices up to Clouds. Software for the different classes of devices are typically built using different approaches and languages. In order to understand the skills and capabilities required to develop services on top of such an infrastructure, we queried a popular open-source repository (GitHub) to evaluate the heterogeneity of programming languages across the continuum. The following sets of keywords were used:

1. Cloud: server with virtually unlimited resources,
2. Microcontroller: resource constrained node (few KB RAM, few MHz)
3. Mobile: an intermediate node, typically a smartphone,
4. Internet of Things: Internet-enabled devices,
5. Distributed systems, as services exploiting CPS have to be distributed across the continuum,
6. Embedded systems, as a large and important part of the service implementations will run as close as possible to physical world, embedded into sensors, devices and gateways.

Figure 1 presents the results of those queries. The queried keywords are presented on the x axis together with the number of matches for that keyword. For each keyword, the y axis represents the popularity (in percent of the total number of matches) of each of the 10 most popular programming languages that we encountered. This simple study indicates that no programming language is popular across the whole continuum. A general trend indicates that Java and JavaScript (and to some extent, Python and Ruby) are popular in the higher-end of the continuum (cloud and mobile), whereas C (and to some extent, C++) is a clear choice for developers targeting embedded and microcontroller-based systems. Other languages do not score more 10% for any of the keywords. For all keywords except Cloud, the combined popularity of Java, JavaScript and C/C++ (i.e., the sum of the percentages) is above 70%. For Cloud, we observe a certain homogeneity with Python, Ruby also being very popular, so the combined popularity of Java, JS and C/C++ is only 50%. It is also worth noticing that the most popular language for a given keyword scores very poorly (less than 5%) for at least another keyword. While it might appear that a combination of C/C++,

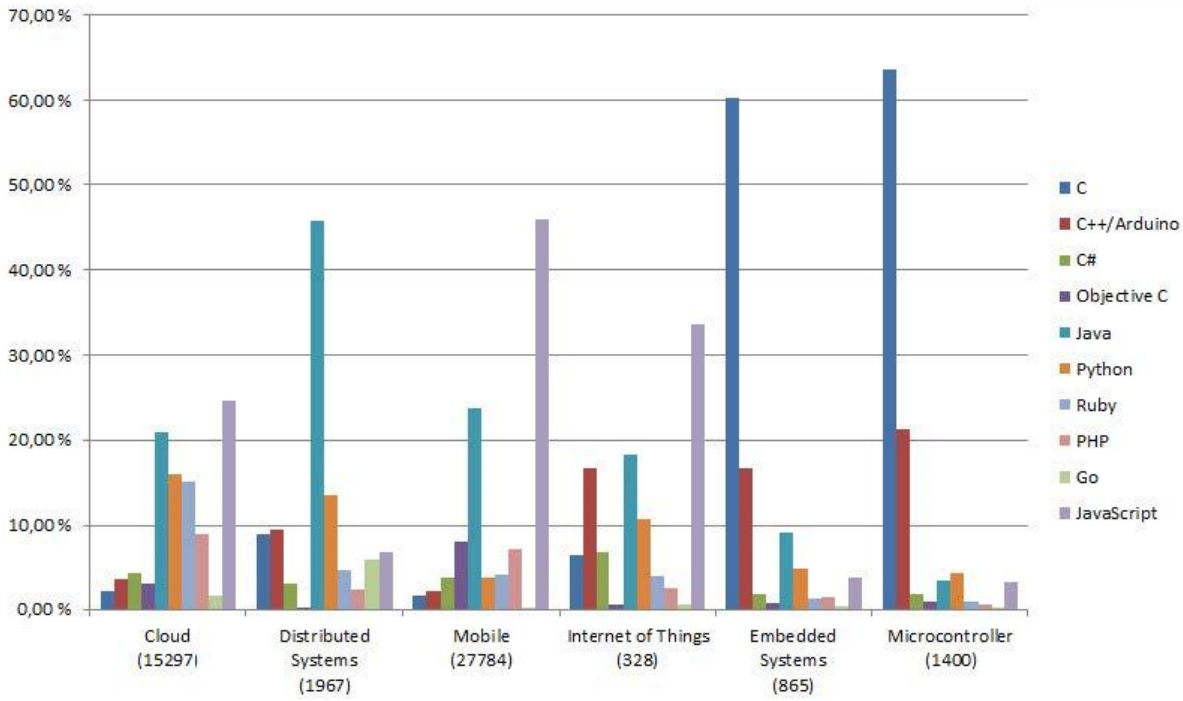

Figure 1: Programming languages used within different domains. The bars show the percentage of the primary programming language that used in GitHub repositories for each of the tags along the horizontal axis.

JavaScript and Java should be able to cover the whole continuum of IoT, in practice it does not exclude the need for other programming languages. For example, the Fibaro Home Center 2 (a gateway for home automation based on the Z-Wave protocol) uses Lua as scripting language to define automation rules[13]. Another example is the BlueGiga BlueTooth Smart Module, which can be scripted using BGScript, a proprietary scripting language. This shows that each part of an infrastructure might require the use of a niche language, middleware or library to be exploited to its full potential. Other examples of such languages include micro-python[14], micro-ruby[15], Rust[16] or even resource-constrained JavaScript interpreters such as Espruino[17].

A wide diversity of programming languages can be used and combined to implement IoT systems. Different languages will, for semantically equivalent programs, have a different impact on resources (CPU, memory, I/Os, etc), yielding diversified footprints.

[13] A brief introduction to Lua, Fibaro, see http://www.fibarouk.co.uk/support/lua/brief-introduction-lua/
[14] https://github.com/micropython/micropython
[15] http://microruby.com/
[16] https://www.rust-lang.org/en-US/
[17] https://github.com/espruino/Espruino

3.2 Semantically-equivalent implementations

Say that for a given language we can implement the same behaviour in many different ways. Give some examples of concrete strategies that could be applied.

For a given programming language, it exists infinite ways of coding the expected behaviour of a program. To take a very simple example, affecting the value 1 to a variable can be achieved by:

- `var i = 1`
- `var i = 1 + 1 - 1`
- `var i = f1() //call to a function returning 1`
- `var j = 1; var i = j`
- `...`

This diversity can be achieved by producing different versions of the source code, or from a single source code, by letting the compilers to produce diversified machine code. A rather large community is providing techniques for compilers to diversify machine code automatically[18], [19], [20].

Another interesting point of variability is the communication between distributed components. Strategies for distributions and communication are described in more details later in this document. In this section, we describe different communication patterns between two processes (A and B), independently of the actual way message are serialized and transported through the network. Figure 2 describes a few patterns that can be used and combined to introduce diversity when communicating, still preserving semantics.

A major choice is whether to "stream" data (A.) (i.e. sending multiple fragment of data, which we denote as a, b, and c) or to send data as a batch (B.) when component A and B need to exchange data. Those two patterns have different implications:

- streaming might be heavier on the network traffic, providing more opportunities to intercept messages. However, each message only carries a subset of the information.
- batch can reduce overall network traffic, reducing the probability that a message is intercepted. However, if intercepted, the message carries the whole information.

The timing (C.) and ordering (D.) of the messages that are sent can also be adjusted, so as to avoid exposing any patterns, which could be more easily interpreted.

---

[18] JAJODIA, Sushil, et al. (ed.). *Moving Target Defense II: Application of Game Theory and Adversarial Modeling*. Springer Science & Business Media, 2012.

[19] DAVI, Lucas, et al. Isomeron: Code Randomization Resilient to (Just-In-Time) Return-Oriented Programming. In: *NDSS*. 2015.

[20] Homescu, Andrei, et al. "Librando: transparent code randomization for just-in-time compilers." *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013.

The routing (E.) of the messages between A and B can also be adjusted, for example by introducing intermediate jumps so as to blur the origin of the message, though additional information should be added so as B can know that a message comes from A. This is a mechanism somehow similar to the Tor network.

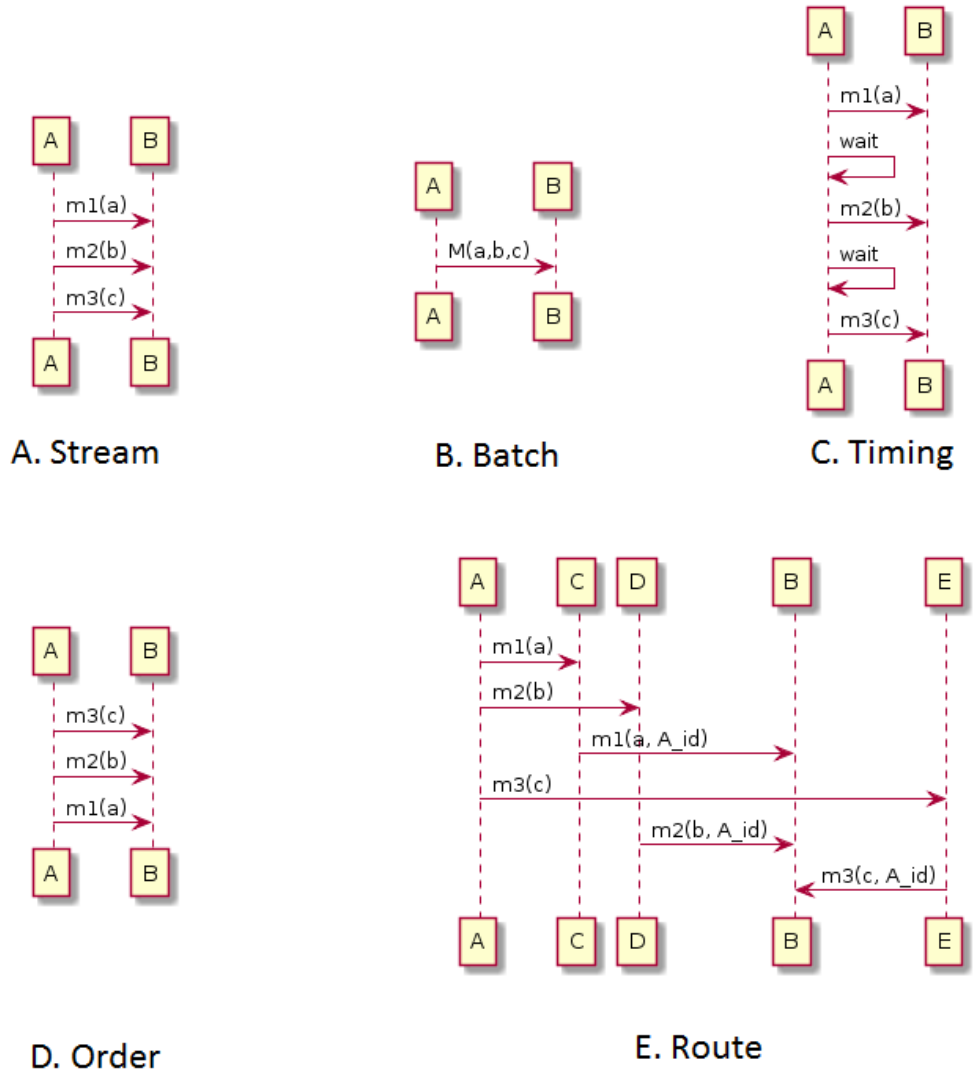

Figure 2: Communication patterns between components.

3.3 Infrastructure & Distribution

For a given piece of software, provided it is architectured as a set of components able to communicate together asynchronously can be distributed in different ways on the available infrastructure[21].

A typical, yet quite naive, distribution pattern for IoT systems is depicted on the left of Figure 3, where most of the logic is delegated to the cloud back-end. In this setup, devices sends raw data to gateways, which simply forward to the cloud. Processes running in the cloud analyse those data and eventually device to enact some commands, which are sent to the gateways and forwarded to the proper devices.

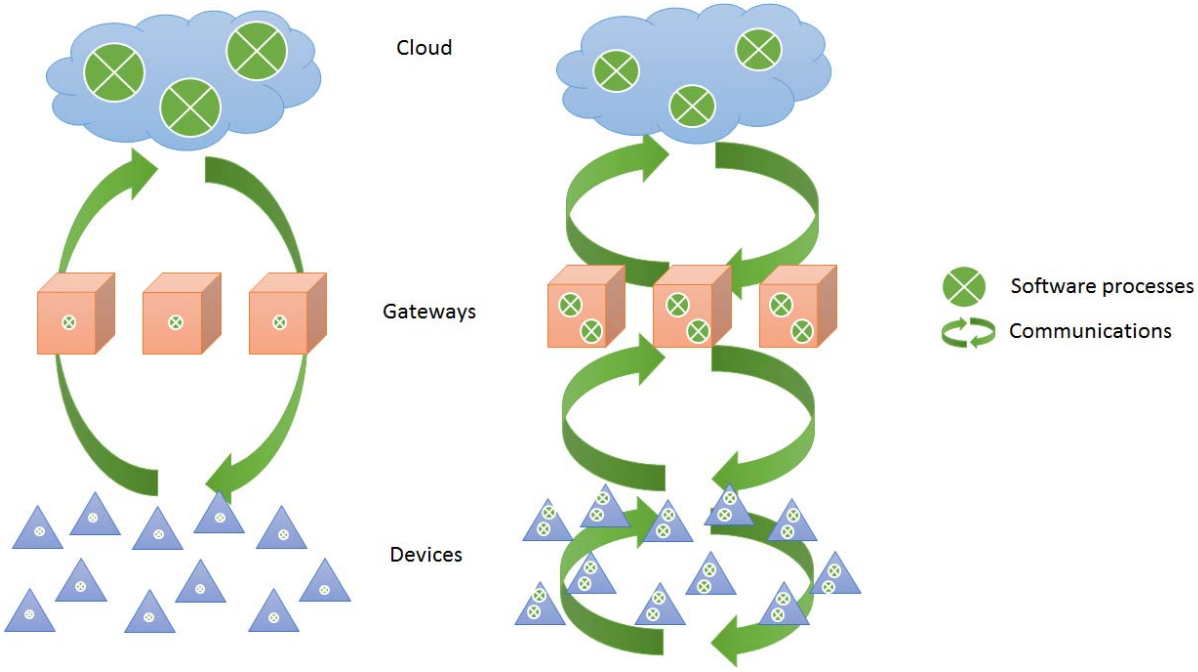

Figure 3: Distribution patterns for IoT software. Traditional centralized pattern on the left, and a more distributed approach on the right.

A more robust way of distributing IoT systems shown on the right of the figure, as advocated by EU FP7 HEADS project[22], is to distribute the logic in a more even way across the IoT infrastructure[23]. This way, decisions can be taken and enacted more locally and the overall system can be made more robust, as it can indeed work (potentially in a degraded mode) without a permanent connection to the cloud. This is for example the way the new Ikea Trådfri home-automation equipment work.

---

[21] VINOSKI, Steve. CORBA: integrating diverse applications within distributed heterogeneous environments. *IEEE Communications magazine*, 1997, 35.2: 46-55

[22] http://heads-project.eu/

[23] MORIN, Brice, et al. A generative middleware for heterogeneous and distributed services. In: *Component-Based Software Engineering (CBSE), 2016 19th International ACM SIGSOFT Symposium on*. IEEE, 2016. p. 107-116.

4 Communication Diversity

As we have described previously, a given IoT system can be deployed in many different ways and rely on different communication patterns. Communication itself can be implemented in very diverse ways, as we describe in this section.

Serialization is an important step in order to be able to send messages on the network for two components to communication together. A large set of libraries and languages provide facilities to ease the serialization and deserialization of messages into strings or binary formats: Google's FlatBuffer[24], Google's Protocol Buffer[25], Apache Avro[26], Apache (formerly Facebook's) Thrift[27], Cap'n Proto[28], SBE (Simple Binary Encoding)[29], MessagePack[30], eProsima Fast Buffers[31], JSON[32], XML[33], etc.

In addition to these "standard" and very popular approaches most programming languages provide ways to manipulate strings and arrays or buffers of bytes, allowing for custom serialization formats.

Different ways of serializing a message will typically ultimately end up with different arrangements of bits on the network.

To actually transport a payload, different protocols can be used, such as HTTP/REST[34], CoAP[35], MQTT[36], AMQP[37], Bluetooth[38], Bluetooth Low Energy[39], Serial[40], ZigBee[41], LoRa[42], SigFox[43], LTE-M[44], etc. Different protocols provide different tradeoff in terms of performances, energy consumption, etc.

[24] https://google.github.io/flatbuffers/
[25] https://developers.google.com/protocol-buffers/
[26] https://avro.apache.org/
[27] http://thrift.apache.org/
[28] https://capnproto.org/
[29] https://github.com/real-logic/simple-binary-encoding
[30] http://msgpack.org/
[31] http://www.eprosima.com/index.php/products-all/eprosima-fast-buffers
[32] http://www.json.org/
[33] https://www.w3.org/XML/
[34] https://spring.io/understanding/REST
[35] http://coap.technology/
[36] http://mqtt.org/
[37] https://www.amqp.org/
[38] https://www.bluetooth.com/
[39] https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works/le-p2p
[40] http://www.ni.com/white-paper/11390/en/
[41] http://www.zigbee.org/
[42] https://www.lora-alliance.org/
[43] https://www.sigfox.com/en
[44] https://www.link-labs.com/blog/what-is-lte-m

5 Managing Diversity in Space and in Time

Diversity can not only be managed in space, i.e. by having diversified instances, but also in time, by having a given instance to change over time. In order to support diversity in time, the running system should be able to undergo changes at runtime, after it has initially be deployed. The challenge is to minimize downtime. Over the past 20 years, the software engineering community has actively been working on dynamically adaptive systems[45], which are able to be reconfigured, or to reconfigure themselves[46], at runtime. In the state of the practice, approaches like Docker containers[47],[48] (built on top of Linux containers and C-groups) now provide convenient way to package, deploy and update micro-services systems in a very agile and dynamic way.

[45] MORIN, Brice, et al. Taming dynamically adaptive systems using models and aspects. In: *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*. IEEE, 2009. p. 122-132.

[46] KEPHART, Jeffrey O.; CHESS, David M. The vision of autonomic computing. *Computer*, 2003, 36.1: 41-50.

[47] https://www.docker.com/

[48] STUBBS, Joe; MOREIRA, Walter; DOOLEY, Rion. Distributed systems of microservices using docker and serfnode. In: *Science Gateways (IWSG), 2015 7th International Workshop on*. IEEE, 2015. p. 34-39.

6 Conclusion: Challenges & Roadmap for further research

Diversity by itself is not a challenge for the IoT. The IoT is indeed a very diversified ecosystem by essence. This survey gave a brief overview of diversity enablers that can be leveraged to build diversified IoT systems.

The key research challenges, related to diversity, to be addressed in the rest of this project are:

- determine what kind of diversity is actually useful to implement robust, resilient and trustworthy IoT systems
- manage this "useful" diversity in a seamless and automated way so as to ensure that the different parts of an IoT system (typically devices, gateways and cloud) can still interoperate and collaborate together, in time (the same instance "looks" different at different points in time) and in space (different instances "look" different), though they might be "different"
- though automated management is needed to tame this diversity, it should:
    - not limit the potential of diversity, so as not to end up with a limited and predictable set of variants, which could easily be understood by attackers
    - not expose explicitly any key or footprint

DiversIoT will address those challenges by raising the level of abstraction of IoT systems. By adopting a Model-Driven generative approach to produce diversified concrete implementations, DiversIoT will be able to manage diversity at a more abstract level and in a more automated and systematic way. A prototype also delivered in Phase 1 shows the feasibility of such an approach.