Guidelines for the Specification of IoT Requirements: A Smart Cars Case

Asmaa Achtaich, Camille Salinesi, Nissrine Souissi, Ounsa Roudies, Raul Mazo

Guidelines for the Specification of IoT Requirements: A Smart Cars Case

Asmaa Achtaich[1, 2] Camille Salinesi[1] Nissrine Souissi[2,3]
Raul Mazo[4,5] Ounsa Roudies[2]

[1] CRI -Paris 1 Sorbonne University, Paris, France
[2] Siweb – Université Mohammed 5, Rabat, Maroc
[3] ENSMR, Rabat, Maroc
[4] Lab-STICC, ENSTA Bretagne, Brest, France
[5] GIDITIC, Universidad EAFIT, Medellín, Colombia

Abstract: In the last five years, we witnessed the shift from the vision of the Internet of things (IoT), to an actual reality. It is currently shifting again from specific and single applications, to larger and more generic ones, which serves the needs of thousands of users, across borders and platforms. To avoid losing the personification of applications, on account of genericity, new approaches and languages that use generic knowledge as a steppingstone, while taking into consideration users and context's specific and evolutive needs are on the rise. This chapter aims to provide a framework to support the creation of such approaches (DSPL4IoT). It is later on used to asses notable IoT specification approaches, and extract conclusion of the trends and persistent challenges and directions. An approach for the specification of Natural Language (NL) requirements for IoT systems is also provided to assist domain and application engineers with the formulations of such requirements.

Keywords: IoT, DSPL, Framework, modelling language, requirement engineering, smart cars

1 Introduction

The Internet of Things (IoT) is here to stay. In 2020, a swapping 99% of companies maintained or increased their budget for IoT (Gartner, 2020). And as the world recovers from a pandemic that froze life as we know it (Covid-19) (World Health Organization, 2020), companies that have succeeded digitalizing their business are the most likely to survive the aftermath. While for some businesses this translates to moving their assets to the cloud, for others like manufacturing, health or agriculture, digitalizing the business is much more complicated. It requires much more advanced devices and technologies, mainly, IoT related. This means that sensors, actuators, smartphones, computers, vehicles, buildings and even people and animal should evolve into "things" (ITU, 2012). That simply -but not so simply- means that they should all eventually possess the ability to remotely communicate, collaborate, have an impact on the environment they serve and, in advanced scenarios, be artificially intelligent.

This new reality emphasizes the need for IoT dedicated standards, best practices and Frameworks. While a plethora of existing works covers various IoT related issues, from its enabling technology like middleware (Ngu et al, 2017), dedicated operating systems or lightweight communication protocols (Baccelli et al., 2018), to storing and processing the big amount of generated data

(Chang et al., 2020)(Mohammadi et al. , 2018) , there's been less focus on requirement specification for this category of systems. After all, requirements are the core of any software system as they convey the expectations of its users. Therefore, a "good" IoT system highly depends on the accuracy, exhaustiveness and quality of the expressed and specified requirements. Conceiving approaches that rise to the expectations of IoT developers and users ought to follow clear guidelines that respect the requirement engineering process and that are drawn from theory and practice in the field of the IoT. This chapter unfolds and organizes these guidelines in the form of an IoT reference framework (DSPL4IoT). The chapter also presents and implementation of the Framework in the form of a semi-formal language for the specification of requirements for IoT development. The language, presented in Natural Language, and delineated by an EBNF grammar (Feynman & Objectives, 2016), can be used by IoT engineers as a blueprint for the definition of their specification. It presents a) an exhaustive view of requirement types that should be considered, at one point or the other, b) interactions with various elements of the environment, including the execution and running context, other devices, people, etc, and finally c) considers the technical, business and contextual evolutions of the IoT field.

The chapter is organized as follows. First it introduces DSPL4IoT along with a definition of its dimensions. Then, in section 3, the chapter defines the fundamentals of requirement specification. Section 4 presents a motivational example. Section 5 introduces the NL specification template, and section 6 maps other existing languages with the proposed Framework, in order to identify current trends in IoT development as well as future directions. Section 7 exposes related similar works before concluding the chapter in section 8.

2 IoT design framework

In order to design comprehensive IoT solutions that guarantee the characteristics discussed in the previous section, three essential aspect should be taken into account :

- **Reusability :** in a specific domain, IoT applications tend to share similar functionalities and qualities, as a result of related requirements. Therefore, existing knowledge shall not be designed for single use. It should rather be stored, organized and capitalized upon for the creation of different IoT applications, for different users and usages.
- **Personalization :** IoT applications shall represent the exact needs and expectations of their users. While reusability helps develop new solutions, faster, and with lowers costs and resources, IoT solutions are very specific to their client's needs, in terms of the choice of devices, the choice of components, and the execution environment.
- **Evolution :** The internet of things connects smart devices, in a "dumb" environment. The first should adapt to the latter to maintain the required functionality and performance, and at times, to adjust it. In addition to that, the environment is also uncertain. This often leads to change in requirements after the execution of IoT software. Besides the evolution of requirements, the composition of IoT solutions is also unstable, new devices can be added, while others can get broken or disconnected. In a like manner, embedded software changes too. All these adjustments shall be thought-out and managed.

The two first qualities are at the heart of software product line engineering. It's a paradigm that manages variability by considering a *domain* layer, where reusable knowledge is organized in variability models, and an *application* layer, where single products are derived, in conformity with

final client's requirements. Evolution is partially tackled in dynamic software product line engineering, which restates the (re)configuration capabilities at runtime. This additional layer, commonly referred to as *adaptation* layer, helps build self-adaptive software product lines.

Moreover, requirements only exist in a context (Pohl, 2010), therefore, at each engineering level, the dependencies between requirements and the context in which they are valid should clearly be stated, and separated from the larger environment that can be relevant for the system, but which does not have a direct impact on it, at a time being.

2.1 Three engineering processes

Building IoT software that provides for all three characteristics follows the guidelines of the three respective engineering processes, namely domain, application and adaptation engineering, as previously stated by Mazo et al. (Mazo, 2018). The three engineering processes are illustrated in Figure 1.

- **Domain engineering** is a development phase for reuse. It is a systematic approach to identify the similarities and differences between protentional applications in a domain, particularly in terms of requirements, architectures and components that can be reused across the IoT product line (Pohl et al., 2005). This phase in the IoT development process is realized by a domain expert, who's likely to have a comprehensive knowledge of it. It offers a description of all the artifacts and their dependencies, provides the means for their effective use and proposes an approach for their implementation. Connectivity for example is a prominent concern for the IoT, as IoT application depend entirely on the internet (Hinai & Singh, 2018) (Lin et al., 2017) (Sánchez-Arias et al., 2017). Communication protocol can therefore be designed as a variable which ought to be implemented as a Wi-Fi, Zigbee, RFID or Bluetooth, etc.
- **Application engineering** is a development phase through reuse (Pohl et al., 2005). It's a process where the reusable artefacts, defined during the study of the domain, are exploited for the construction of compliant products. Thus, and based on the needs of end users, the selection and assembly of the artefacts is carried out at this level, by an application engineer. The result of this activity is an executable IoT application, an architecture, a test unit, etc. During this phase, the application user can decide to alternate between the Wi-fi or RFID protocols to connect his devices, according to a pre-set logic.
- **Adaptation engineering** maintains activities of application engineering after the IoT solutions is executed. It manages the behavior of the derived product, i the face of changing requirements, environments and internal alterations (Danny Weyns, 2017). As a reaction to an internet interruption for example and in order to ensure service durability, the communication protocol can dynamically switch to RFID technology instead if a Wi-Fi based communication.

2.2 Three requirement engineering aspects

Three addition dimensions should be considered in IoT design. The system, the (relevant) context, the (generic) environment, as illustrated in Figure 1.

- **The system** is a collection of components, organized to perform a function (INCOSE, 2018). It's everything that "is" the IoT application, including devices, the communication protocols, but also the components and their current configuration. As adaptations occur, the devices involved in the IoT application along with the configuration of active

components may change, therefore changing the system too. It is therefore a dynamic dimension.

- **The context** is every information that can be used to characterize the situation of an entity. Therefore, everything that surrounds IoT applications, and which has a direct impact on it, is part of the context (Sezer et al., 2018). Users that interact with the IoT application's interfaces, the weather, the state of batteries or the statistics about device' usage is considered part of the context. The elements of a context are not static. They are relevant in specific configurations but can become inconsequential in other. This means that the context is a dynamic dimension too, as its elements alternate between contextual and environmental. Moreover, the composition of the IoT application itself evolves as adaptations take place. Therefore, the context and system dimension are partially blended as well.
- **The environment** is a dimension that is not affected, nor does it affect the system. It contains information relating to the domain of the IoT application, along with some other correlated domains which has the potential to be of value.

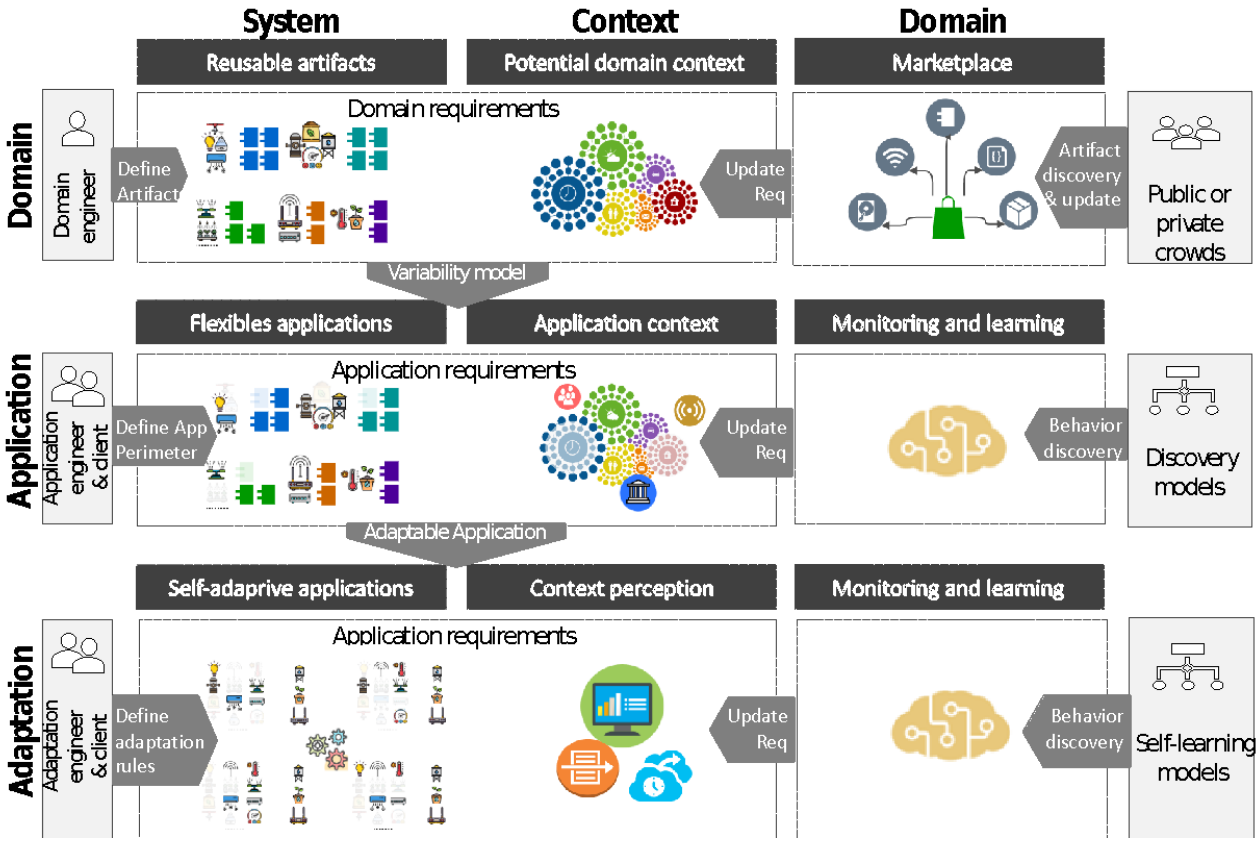

Figure 1 : IoT design Framework

Figure 1 presents the IoT design framework which shows the different concern levels that should be specified, for a proper realization of IoT. The vertical dimensions guarantee reusability, personalization and evolution, and the horizontal dimensions represent the IoT system, along with its dependencies to its context and environment.

3 REQUIREMENTS SPECIFICATION FUNDAMENTALS

Requirements are the heart of any software (Chakraborty et al., 2012), including IoT systems. A proper understanding of their engineering process is fundamental to a proper design, and ultimatly, to a better user experience. Before diving into a classification of requirements (in section 4) and mapping formal requirement specification approaches for IoT to the proposed framework (in section 6), it is important to grasp the fundamental concepts of requirement engineering development, from elicitation, specification, to verification and validation (Pohl, 2016).

While the focus of this chapter remains of the aspects of specification, the other activities are discussed to provide the reader with a holistic view of the requirement engineering process, for the development of IoT solutions.

3.1 Requirement Elicitation

Gathering requirements is a decisive phase in a requirement engineering development approach (Lahboube et al., 2014). While it may appear evident, however, deciding what to build is sometimes the hardest part about software development (Bowen & Hinchey, 1999). Building software for the internet of things is specifically challenging, due to the fact that requirements are constantly gathered, even after the IoT application is running (Antonino et al., 2018). First, at the domain level, requirement elicitation is the responsibility of domain engineers who define what the IoT application can do by eliciting reusable requirements. Also, with new devices, new protocols and new services lunched to the market with a speed that's never been witnessed before (Atzori et al., 2017), come new requirements that shall dynamically be discovered. As illustrated previously in the framework in figure 1, a marketplace has the potential to collect the new requirements, formally through specialized crowdsourcing platforms (Salinesi et al., 2018), or informally through public marketplaces. Then, at the application level, requirements are gathered from the users of the final application. Through questioning, observation, or by the means of other elicitation techniques (Khan et al., 2014). Yet again, contrary to conventional software, IoT benefits from the powerful services that come with real time context-awareness, cloud data supplied by thousands of connected devices, and the latest artificial intelligence algorithms (Hwang & Chen, 2017). These cognitive capabilities empower IoT applications to anticipate completely new user needs. Finally, at the adaptation level, requirements are expressed by both clients and experts that are aware of context's implications on a running application. And similarly, to the previous engineering process, new adaptation requirements can be learned, progressively, by intelligent services, made possible thanks to smart monitoring, and therefore, elicited dynamically.

Regardless of their source, requirements are analyzed. This activity's main concern is to determine if the collected requirements are unambiguous, complete and consistent. It also helps detect existing conflicts, inconsistencies or dependencies between requirements. At the end of this phase, it's not unlikely to build a simplistic prototype to confirm the expressed and collected requirements.

3.2 Requirement specification

Once gathered, understood, revised and improved, requirements are documented. At this stage, a requirement specification language is used to record requirements, in a formal, semi-formal or even informal fashion.

According to the proposed Framework, at the domain level, domain experts provide a full map of all possible capabilities, qualities, components at different levels of abstractions, and the relationships and dependencies that govern these entities. They can also identify context elements that potentially affect the behavior of the IoT application. Furthermore, as new requirements arise to accompany a technological or business evolution in the IoT domains, the specification can be performed by the use of external resources and services (i.e. Marketplace APIs) automatically (self-adaptive model). Then, at the application/adaptation level, the project manager/expert who accompanies the client at the elicitation phase drafts the specification document, taking into account the causes and consequences of each derivation/adaptation. Once more, as new requirements are learned on the go (i.e. New usage patterns, new execution environment, etc), the corresponding specification should automatically be formulated too.

Requirement specification is very critical to the overall process. As a matter of fact, it's a binding contact between whoever the IoT solution is conceived for, and whoever is building it. Any mistakes or even imperfections at this level may have exponential repercussions as the project evolves, which often comes at a high cost or loss for both parties (Knauss et al., 2009).

Consequently, a variety of rigorous requirement specification languages exist. Some of them are formal, other are unformal, and some are in between, and are therefore semi-formal (also called hybrid or structured).

- **Formal** specification languages are specification documents, who's syntax and semantics are expressed and defined formally, using logic, algebra or standard mathematics (Spivey, 1989)(Jones, 1995). Formal languages are usually automatically processed and can preserve traceability throughout the complete engineering process. Formality remains however difficult to attain and use, especially when untrained software engineers are the ones usually responsible to draft the specifications document, without much tool support either.
- **Informal** specification languages mainly refer to the use of human language to document the specifications of a software system. Natural languages (NL) can either be unrestricted and without any defined format, which leaves room to ambiguity, personal interpretation, bias, and other quality defects. They are widely adopted nevertheless, thanks to their instinctive and universal format
- **Semi-formal** specification languages rely on predefined graphical or textural notations that constrain the expression and form of specification documents. Although they may lack formality in the definition of their syntax or semantics, but they specify requirements in a structured form, regulated by clear guidelines, and supported by tools. UML (OMG, 2017) and KAOS (Lamsweerde, 2009) are some of the most notable semi-formal specification languages. Some NL specification languages also belong to this category of languages due to the fact that they have been improved by complementary concepts that enhance their uniformity, like templates (Robertson & Robertson, 2012), ontologies (Körner & Brumm, 2009), or metamodels (Videira & Da Silva, 2005).

3.3 Requirement validation

Verification and validation are the final steps of requirement devolvement. During this stage, the specifications document is assessed, to verify its correctness, completeness and consistency with regards to the expressed needs of final users, before moving to the software system development phase (Boehm, 1984). A variety of validation techniques can be employed (Maalem & Zarour, 2016). For instance, when the specification language is formal, this stage is often automatically achieved, as most formal approaches generate prototypes (Yang et al., 2019). In the case of semi-formal languages, traceability between user goals and the specifications is model-based (Iqbal et al., 2020). Natural language specifications usually involve both clients and the requirements development team.

4 Motivational example : The chameleon smart car

Chameleon is a hand-picked smart car manufacturer on the rise. What differentiates it from the competition is the diversity of its catalogue, assembled from years of experience in the automotive, electronics and digital fields. The power of Chameleon smart cars goes beyond car related applications, like smart tracking, smart parking, smart traffic, or smart braking. It also includes services from other areas such as smart health, smart surveillance, or smart supply chain. It can be destined for a variety of clients; like a logistics company, a special needs centre, a housing complex or even the city. Therefore, confronted with such a diversity and genericity, decisions regarding the quality and quantity of devices, components or services embedded in the car for a specific application, is tedious. Furthermore, even after building the car, requirements are prone to change, and sometimes, evolution. This change is a result of the varying and uncertain circumstances in which cars run. In addition to changing requirements as projects progress, the chameleon car faces challenges related to the consistent evolution of the embedded devices and their software on the one hand, and the dynamic physical composition of the smart car on the other. Both cases require runtime adaptation. This section describes the case in further details and presents an application scenario.

The domain of the Chameleon solutions englobes all knowledge that has been collected in the domain of smart cars. Starting from the mechanical parts and their components (i.e: engine, wheels, brakes, etc), going through the embedded smart devices and their potential configurations (i.e: Cameras, speed or proximity sensors, etc), all the way to possible smart applications (i.e: Face recognition, smart braking, photo analysis, etc) or technologies (communication technologies, routing protocols, etc). Requirements that articulate this knowledge are connected to contextual information that might be relevant for the final user. Some cities for example may prohibit self-driving cars during rush hours, while others don't. Time and location are therefore potentially relevant contextual elements. Moreover, information about domains other than smart cars, like smart health, may not appear relevant for all applications, but can if the car is destined for elderly clients. New devices, which have proven more practical, and more accurate for the measurements of cardiovascular endurance, should be presented to domain engineers in order to be considered as an alternative or replacement for old versions of the same device, insuring thus state-of-the-art smart products.

These three dimensions of the Framework can be respectively implemented through domain variability, context, and crowdsourcing models that enable public contribution.

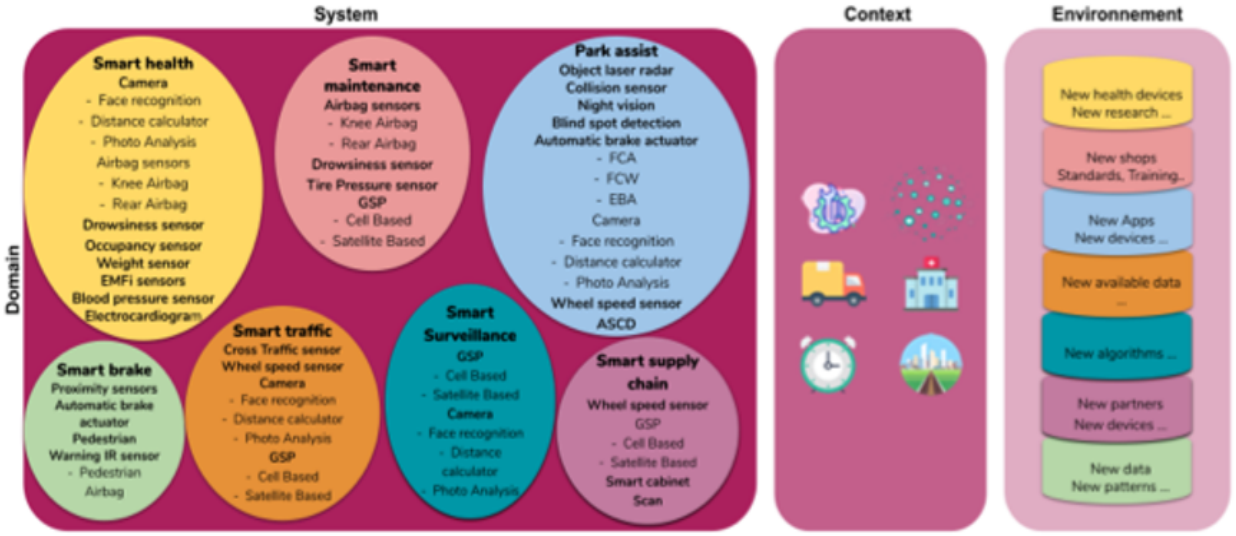

Figure 2: The Chameleon cars domain

Figure 2 illustrates the first layer of the framework; the domain. The chameleon domain can embed a multitude of functionalities, like smart health smart maintenance, park assist, smart brake, smart supply chain, smart surveillance and smart traffic. Each of which can be implemented in a variable manner as well. The context illustrated in the figure includes time and place, nearest hospital and logistics, city public data, etc. The environment of the smart car is a marketplace that contributes to each of the functionalities of the system dimension with new devices, new requirements, new research, new patterns, etc.

With every client, new requirements come to light. Some of which are derived from the domain knowledge, and some are specific to the new clients.

Both cars derived for a retirement home and a logistics company may want to use path calculation algorithms, along with the sensors and cloud information required for that purpose. several algorithm options are selected. Each activated depending on the state of certain devices and the availability of specific data. The first application focuses on smart health devices and applications to monitor vital signs. At specific times during the day, other services such as smart brake and parking are also enabled to anticipate elderly drivers' slow reflexes and reduced sight. The second application requires smart traffic and surveillance devices, together with data analysis application, without much regard to the other possible options.

New requirements should also be considered, as each client operates according to its own specific agenda. For instance, the Chameleon domain requirements to manage connectivity include Wi-fi, RFID and 4G communication protocols. However, as the logistics company operates using a ZigBee built-in platform, related requirements are added accordingly.

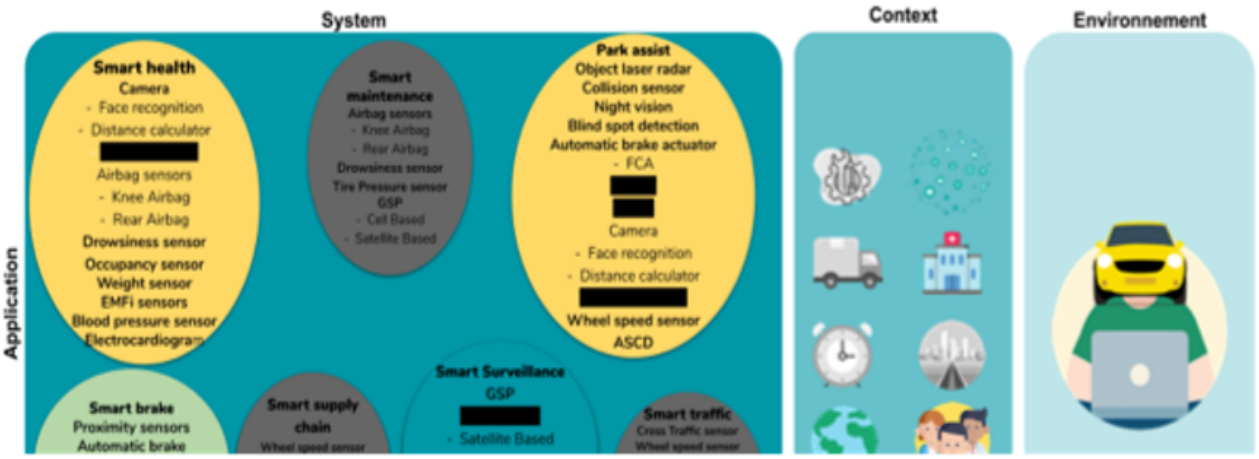

Figure 3 : The case of the Village Retirement Home fleet of cars

As illustrated in Figure 3 for the case of the Village Retirement Home (VRH), the smart maintenance, smart supply chain and smart traffic are not selected. Amongst the remaining functions, irrelevant devices and component for this client are disabled as well. Thus, creating a fleet of cars that answer, without access, the needs of the client, while maintaining a certain level of autonomy for reconfiguration at runtime.

As a matter of fact, the chameleon car is set to operate in dynamic circumstances, where the context is constantly changing, the running software often updated and the composition of the car itself is likely to be altered. For instance, the smart cars can be part of a national safety program that helps track wanted profiles. Embedded cameras are empowered with face recognition applications, connected to the police information system. This is called, smart surveillance. When traveling in different countries, these applications are prohibited, as they are conceived a breach of privacy. There are therefore disabled as a response to change in the context. More flexible requirements are also expressed to deal with the uncertainties of the context, like the state or battery level of sensors or the availability of certain services like road information.

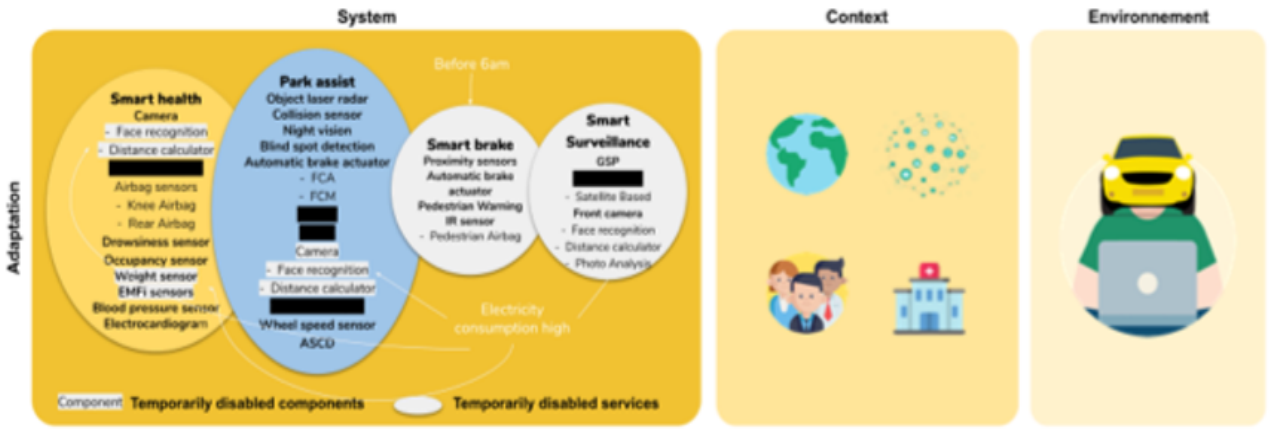

Figure 4 : An adaptation for the Village Retirement Home cars

Figure 4 shows simple adaptations of the smart car, as a result of decrease in charge, and specific time in the day. In the first event, optional functionality like smart surveillance, along with optional components like face recognition or weight sensors, are disabled to maintain the operability of other mandatory services. The second event illustrates the smart brake functionality case, which is only enabled at night-time, when the eyesight of elderly people decreases.

5 How to specify NL requirements for IOT systems

In order to develop a representative typology of requirements for the specification of IoT systems, a first draft of the classification is drawn from the chameleon cars case. A modified systematic search in the current state of the art is then carried out in order to complete the list with new types of requirements.

The typology of requirements presented below follows the rules of an EBNF grammar. The syntax used is summarized in Table 1.

Symbol	Description
`<non-terminal>`	Syntax variables that define the requirement Natural Language
`« terminal »`	Character strings that appear in the Natural Language
`+`	(Non) Terminals which can be instantiated in one or more occurrences
`?`	Optional (Non)terminals
`|`	An OR relation between two (Non)terminals

Table 1 : Description of the EBNF grammar

The resulting requirements fall under three main categories. First, high level requirements that ensure reusability, they describe the main capabilities and qualities of a category of systems, along with various possible compositions of single systems. Then, application requirements, which describe the properties of the running system, taking into account the specificities of its final user

and its execution environment. And finally, runtime requirements, which are typical to autonomous systems, as context-awareness and self-adaptation are innate properties of IoT systems.

<Requirement> ::= **<DomainRq>**|**<ApplicationRq>**|**<AdaptationRq>**

5.1 Specification of domain requirements

Domain requirements are characteristic of a particular market segment, they describe the basic functions that any system belonging to that domain is likely to have (Pohl et al., 2005). These functions can be qualitative or quantitative, and thus are specified by means of functional or non-functional requirements. In addition to that, variability requirements are introduced to describe the possible relationships between "things", which are categorized as hierarchical, group, dependency or numeric (Mazo et al., 2012; Salinesi et al., 2011). The first three refer to the possible compositions of the IoT product, and the last one describes parametric elements that can be configured at runtime.

<DomainRq>	::=	**<FunctionalRq>**|**<NfunctionalRq>**|**<VariablityRq>**
<FunctionalRq>	::=	*The* **<thing>** (*shall*|*could*|*Might*) (*Insure*|*maintain*) **<Task>**
<NFunctionalRq>	::=	*The* **<thing>** (*shall*|*could*|*Might*) *be* **<Quality>**
<VariabilityRq>	::=	**<HierchicalRq>**|**<GroupRq>**|**<DependencyRq>**|**<NumericRq>**
<HierchicalRq>	::=	*The* **<thing>** (*shall*|*could*|*Might*) *be composed with* (**<thing>** ("*AND*" **<thing>**)+)
<GroupRq>	::=	*The* **<thing>** (*shall*|*could*|*Might*) *be composed of at least* **<min>** *AND at most* (**<max>**|*) (*instances of* **<thing>**|(*among* (**<thing>** ("*AND*" **<thing>**)+))
<DependencyRq>	::=	*The* **<thing>** *shall* (*combine*|*dissociate*) **<thing>** ("*AND*" **<thing>**)+)
<NumericRq>	::=	*The* **<thing>** (*shall*|*could*|*Might*) *be valuated within* (**<domain>**|**<enumeration>**)

The grammar presented above describes a semi-formal NL approach that can be followed to specify the elicited high-level reusable requirements for IoT applications. The non-terminal **<**thing**>** used in the grammar refers to any element that composes the IoT application. This is derived from the very definition of the term in the IoT glossary (ITU, 2012). A sample of domain requirements for the chameleon car manufacturer, specified using this grammar, is presented in the following table.

ID	Req Type	Requirements
Rq_6	**FunctionalRq**	*The cars could insure automatic brake*
Rq_19	**NfunctionalRq**	*The cars could be efficient in terms of electric consumption*
Rq_45	**HierchicalRq**	*The bumper may include pedestrian airbags*
Rq_66	**HierchicalRq**	*The seat shall include drowsiness sensors*
Rq_84	**NumericRq**	*The cars shall include at least three front and two back proximity sensors*
Rq_398	**GroupRq**	*The gateway could mutate between three means of communication*
Rq_576	**DependencyRq**	*The cars shall combine every airbag, with at least two active seat sensors, and a control unit.*

Table 2: Sample of domain requirement for the Chameleon cars

5.2 Specification of application requirements

Application requirements are designed in collaboration with the clients. They include functionalities and qualities retained, represented by means of functional and non-functional requirements. (Abbas et al., 2010; D'Ippolito et al., 2014; Muñoz-Fernández et al., 2018; Yang et al., 2013). These requirements can be operationalized in various manners, in accordance with other user requirements, like preference, cost, optimization or autonomy (Soares et al., 2017). Preferability requirements describe explicit choices. Cost requirement describe the budgetary constraints assigned to the product in question. Proportionality requirements specify the rules that define logical relationships between various elements. Optimization requirements define an optimum on the level of required performance or on specific values of the application And finally, autonomy requirements represent flexibility, which offers adaptation alternatives for dynamic contexts at runtime (Vassev, 2015) .

<ApplicationRq>	::=	**<FunctionalRq>**|**<NFunctionalRq>**|**<PrefereabilityRq>**|**<CostRq>**| **<OptimizationRq>**|**<ProportionalityRq>**|**<AutonomyRq>**
<FunctionalRq>	::=	*The* **<thing>** *shall* (*Insure*|*maintain*) (**<Task>**|**<AppParam>**)
<NFunctionalRq>	::=	*The* **<thing>** *shall be* (**<Quality>**|**<AppParam>**)
<PrefereabilityRq>	::=	*The* **<thing>** *shall* (*Not*)? include (**<thing>**|**<AppParam>**) (*with the value* **<Operator><Value>**)?
<CostRq>	:==	*The* **<thing>** *shall cost (at most|at least)* **<Price>**
<OptimizationRq>	::=	*The* **<thing>** *shall* (*maximize*|*minimize*) *the* (**<Quality>**|*number of* **<thing>**)
<ProportionalityRq>	::=	*The* **<thing>** *shall repsectively select* (**<Param>** ("*AND*" **<Param>**)+) **<thing>** *together with* (**<Param** > ("*AND*" **<Param>**)+) **<thing>**
<AutonomyRq>	::=	**<VariabilityRq>**
<AppParam>	::=	**<NewTast>**|**<NewQuality>**|**<NewThing>**

The grammar above defines rules for a natural language specification of application requirements. A sample of such requirements, as expressed in the retiring home case, is presented in Table 3.

ID	Req Type	Requirements
VRH_Rq4	FunctionalRq	*The cars shall insure smart health monitoring.*
VRH_Rq66	NfunctionalRq	*The cars shall be energy efficient.*
VRH_Rq34 VRH_Rq166	PreferenceRq	*The bumper shall include pedestrian airbags.* *The cars shall not have front recording cameras*
VRH_Rq89	CostRq	*The cars shall cost at most 15000$*
VRH_Rq104	OptimizationRq	*The cars shall maximize the number of proximity sensors*

VRH_Rq153	ProportionalityRq	*The cars shall respectively select 2, 4 and 5 airbags together with 2, 5 and 7 seats*
VRH_Rq193	AutonomyRq	*The cars could include smart assist services.* *The gateway could mutate between various means of emergency communications*

Table 3 : A sample from application requirement for the of the Village Retirement Home case

5.3 Specification of adaptation requirements

Adaptation requirements describe the behaviour of IoT applications in dynamic contexts. They describe the necessary reconfigurations to maintain required levels of satisfaction. Contextual requirements define circumstances under which requirements shall be satisfied. Temporal requirements determine the time, order or frequency with which requirements must be satisfied. Optimization requirements maintain their role of maximization or minimization of parameter values or cardinalities (Uthariaraj & Florence, 2011). Relaxable requirements refer to those that are necessary in particular contexts, but which may prove to be less essential in other ones (Whittle et al., 2010). Awareness requirements are sensitivity requirements that constrain the degrees of success or failure in implementing other adaptation requirements (Souza et al., 2013). And finally, resilience requirements also called evolution requirements, determine the requirements which specify the response to be given in the event of failure to implement other adaptation requirements (Souza et al., 2012).

<AdaptationRq>	::=	**<ContextualRq>**|**<TemporalRq>**|**<OptimizationRq>**|**<RelaxableRq>**|**<AwarenessRq>**|**<ResilienceRq>**
<ContextualRq>	::=	*When* **<Event>** *if* **<condition>**, **<Requirement>**
<TemporalRq>	::=	(*At* **<Time>** | (*Before*|*After*) (**<Time>**|*reinforcing that* **<Requirement>**)| *Between* **<Time>** *and* **<Time>** | *as soon as* (**<Time>**|**<Requirement>** *is realized*)), ***<Requirement>***
<OptimizationRq>	::=	*When* **<Event>** *if* **<condition>**, *the* **<thing>** shall (*maximize*|*minimize*) *the* (**<Quality>**|number of **<thing>**)
<RelaxableRq>	::=	**<OptimizationRq>**, (eventually|until) **<Requirement>**
<AwarenessRq>	::=	**<AggregationRq>**|**<MetaRq>**|***<Delta*Rq>**
<AggregationRq>	::=	***<Requirement>*** *should (succeed|fail)* ***<Percentage>*** *((More|less) than* ***<Requirement>)?***
<MetaRq>	::=	**<Requirement>** *should be satisfied within* ***<TimeDuration>***
***<Delta*Rq>**	::=	**<Requirement>** *success rate should not (decrese|increase) (<**TimeDuration**> <**Frequency**>)*
<ResilienceRq>	::=	*When* **<Event>** *if* **<condition>**, **<Requirement>** *shall be* (*ignored*|*modified* (*into* **<Requirement>**)?)

The grammar described above introduces for each requirement type, a semi-formal approach for the specification of adaptation requirements. A sample from the specification document of the village retiring home case is presented Table 4.

ID	Req Type	Requirements
VRH_Rq566	ContextualRq	*When a seat detector detects a new passenger, the cars shall be able to communicate with the occupant's health monitor wearable.*
VRH_Rq587	TemporalRq	*After 6 am, the cars shall enable all devices that contribute to smart brakes.*
VRH_Rq645	OptimizationRq	*When electricity consumption is higher than 60%, the cars shall optimize the use of slave sensors*
VRH_Rq699	RelaxableRq	*The cars shall use the least possible sensors, eventually, all sensor's battery levels shall stay superior than 40%.*
VRH_Rq702	AwarenessRq	*The car shall be able to reach the closest ambulance in the case of a crash, within 10 seconds*
VRH_Rq725	ResilienceRq	*When the health emergency state is active, if VRH_Rq645 is not satisfied, the cars shall use another meant of emergency communication*

Table 4 : A sample from adaptation requirements for the of the Village Retirement Home case

Reusable domain requirements of IoT solutions on the one hand, and application and adaptation requirements on the other, reveal the need to specify an extremely diverse set of configuration requirements. Although this typology is based on the standard typology of requirements (IEEE, 2009)(Lin et al. 1996), they differ from the latter as they are not always binding. In reality, they are only verifiable in specific contexts. The specification of these requirements, which can be described as "dynamic", is a steppingstone for the realization of highly reconfigurable IoT solutions, which meet different needs, while ensuring the natural evolution of this category of systems.

6 Outlook on next generation IOT design

For the last decade, a multitude of approaches have been proposed in order to specify and design applications in the challenging but promising field of the Internet of things. This section presents and overview of these approaches. Furthermore, to understand the scope of these contributions along with the other aspects that need further investigation, each of the selected approaches is mapped to the framework.

- **Approach 1** : SysADL (Leite et al., 2017) is an architecture-based approach for building IoT applications. According to the approach, all elements of the IoT application are defined before they are used in the system architecture. Along with the structural definition, data, components, actions, connections and executables are all defined as part of the system's environment. Both are defined in block definition diagrams. Data that flows in and out of each element is also specified using the concept of ports. Instances of systems correspond in SysADL to configurations. They describe how components are connected, thus, how actual instances of the IoT application can be configured. Internal block diagrams are used to describe possible configurations of components, connected using ports that send their respective values. All of the above is defined as part of a ***structural viewpoint***, which correspond to the ***domain system and context*** dimensions of the proposed Framework. This next view, called ***behaviour viewpoint***, describes how the IoT elements contribute to the fulfilment of high-level requirements described in this previous viewpoint. This shows

different application scenarios using an activity diagram and correspond to the ***application system and context*** dimensions or the proposed framework. Several other SysML based approaches like SysML4IoT are used for the specification of IoT solutions (Costa et al., 2016). The mapping of these approaches with the IoT design Framework is presented in figure 6.

- **Approach 2 :** Authors in (Hussein et al., 2019) extended the previously discussed notation, SysML4IoT, with self-adaptation capabilities, using a publish/subscribe adaptation paradigm to model environment information and their relationship with the system. This is preformed using a system management component which model adaptation triggers and runtime configurations using the concept of states. This model matches with the ***system and application adaptation*** dimensions of the proposed Framework as illustrated in figure 6.
- **Approach 3** : State Constraint Transition is a language for the formal specification of IoT systems (Achtaich et al., 2019). SCT is a variant of finite state machines (FSM) whose power of expression is extended by means of the concept of constraints. This modelling language provides an answer to the problems linked to the specification of dynamic requirements by introducing the concept of configuration states, in which requirements are translated into constraints. First, all IoT elements, together with the anticipated domain context are defined in the form of variables. A ***domain variability model*** specifies the dependencies and relationships between these elements, in the form of constraints. A ***configuration model*** specifies application requirement as an instance of the domain variability model. Contextual elements that arise with each specific application are defined within the application's corresponding state. A ***reconfiguration model*** specifies adaptation requirements, by the means of configuration states embedded with constraints which formally specify dynamic requirements. A ***perception model*** informs the generated constraint program with real time contextual data or parameters, which potentially leads to a reconfiguration of the IoT solution. The models described above are a projection of ***domain, application and adaptation*** requirements, regarding both the ***system and its context.*** A projection of these models on the framework described above are presented in figure 6.
- **Approach 4** : The approach proposed by Karakostas (Karakostas, 2016) is based on the author's observation that it is difficult to predict with certainty when and if events will occur in an IoT application. Thus, his proposal implements a ***bayesian model*** that predicts relevant events and consequences. Through an air flight case study, the authors predict if a connecting flight will be departing late, by calculating the probability of the incoming/arriving flights doing departing or arriving late. This model is an implementation of the ***application environment*** dimension and is presented in figure 6. Similar implementations are found in the literature, like the works of Basu et al. (Basu et al., 2018) who tackle the problem using cognitive bio-inspired models.

- **Approach 5** : This contribution by authors Lunardi et al. (Lunardi et al., 2018) is based on the Model Centred Architecture (MCA) paradigm, where a system is a compound of various models and model handlers. Specifically, authors define ***M1*** to specify all system and data components. This coincides with ***system and context*** dimensions of the ***domain engineering process*** in the proposed framework. Concrete functions and processes are later on specified at an M0 level, which is an instance of M1 models. M0 corresponds to the ***application level***, and

both system and context dimensions are specified using this approach. Furthermore, the authors propose an extension to a semantic core model, using probabilistic ontology, in order to predict human actions. This engine injects new knowledge as new behavioural patterns are predicted. This model refers to the ***environmental dimension of the application engineering process***. The mapping of this approach with the Framewok is illustrated in figure 6.

- **Approach 6 :** Google Nest[1] is one of the most commercially successful cases of IoT. It's a smart home solution that sets up, monitors and manages home appliances like thermostats, cameras and locks. While the design process and methods are not displayed to the public, we can deduct from the actual solution that Google Nest solution follows generic and adaptable specifications, which we labelled ***domain***. While this solution offers a range of possible configurations (***applications***), the derived applications are not flexible in terms of requirements, and solutions are only personalized within the boundaries of the supported devices and configurations. Devices and related applications are available on the Google Store, which is a marketplace that is often updated with the latest supported technology. This corresponds to the ***environment dimension*** of the framework. The various dimension implemented by this technology are displayed in figure 6.
- **Approach 7 :** Comma[2] is a self-driving car software. It's an IoT solutions that can operate with any car that supports automatic acceleration, brake and parking. The design of Comma is one of the most fitting to the proposed Framework. On the one hand, it offers a generic solution that fits to different car application. It can work with a Honda as well as it does with a Toyota, therefore supporting the ***domain system and context*** dimensions. The software is opensource and available for the public. It is therefore crowdsourced, which makes for the ***environment*** dimension of the ***domain***. Comma collects application requirements through a user interface in order to present a solution that is tailored to each specific case scenario. This coincides with the ***application system and context***, presented in the Framework. As the car runs, adaptation scenarios are successfully specified and implemented. Moreover, comma is a self-learning software. Its design specifies basic functionality and learns on the ground and as the car drives itself, in order to improve its functionality, and provide most fitting actions and reactions. This corresponds to the ***adaptation engineering*** process in totality. Figure 6 illustrates the projection of the comma specification process on the Framework (Yellow).

[1] https://nest.com/
[2] https://comma.ai/

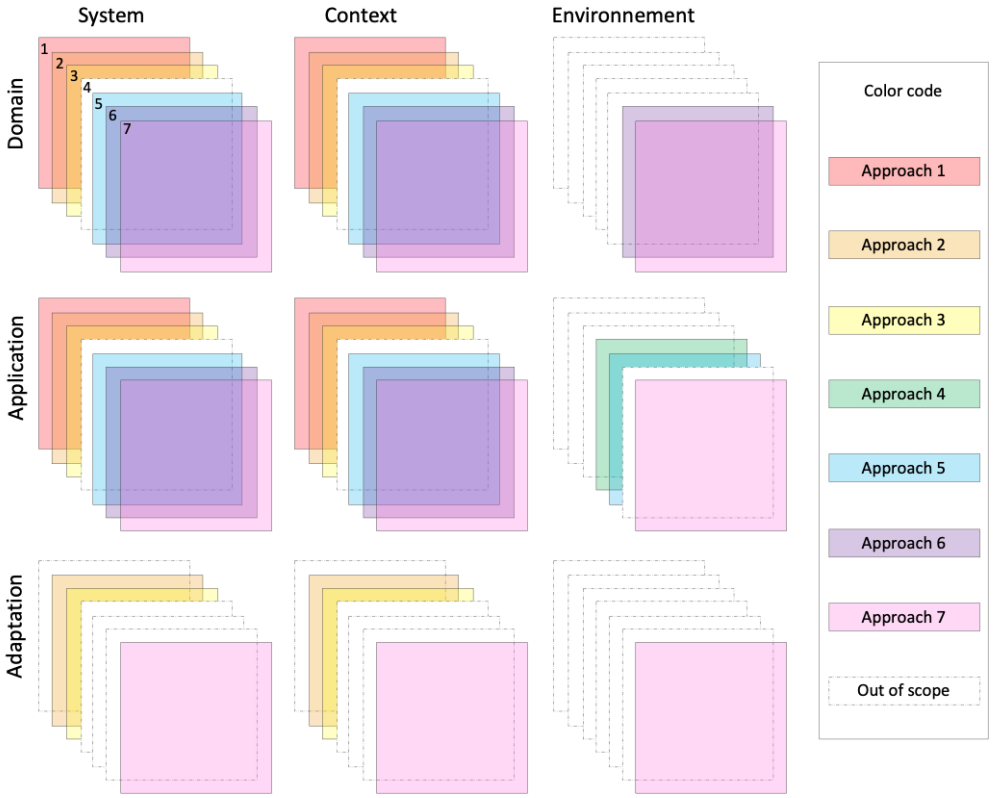

Figure 6 : IoT design trends

The approaches presented above are selected from different IoT applications, including smart homes, smart cars and smart cities. They also represent examples evoked both from academic research and from the industry. This clearly outlines the current trends in IoT development, but also pinpoints the aspects that are just as important for building comprehensive technologies for the future, that require more attention from academia and from the smart industry.

6.1 IoT and SoS

A great number of IoT solutions are specified, partially or completely, using the SysML notation and principles. This is clearly displayed in figure 6, as most approaches focus on domain and application requirement specification. This is not a coincidence. As a matter of fact, ever since IoT became a hot topic in research, a lot of authors debated the need for new terminology to something that had already existed and matured in the literature, under the name System of Systems (SoS) (de C Henshaw, 2016; Mahya & Tahayori, 2016; Nikolopoulos et al. 2019). While there are definitely new capabilities and specification challenges brought by the smart nature of devices used in IoT applications, broadening the scope of SoS could be an approach to embracing such progress instead of rethinking and reinventing a whole new paradigm.

6.2 IoT and autonomy

Specifying ad-hoc IoT applications for a static usage, without considering the dynamic context and use cases, can be considered unrealistic. This approach may even lead to error-prone and contradictory results a consequence of uncertainty. Self-adaptation, both to context and to requirements, is a rising interest in IoT development. The main goal of research in this area is to introduce new languages, patterns and algorithms that not only handle uncertainty, but also discover and implement autonomously new requirements and usage scenarios at runtime.

6.3 IoT and AI

The focus around IoT for the last decade have revolved around three major topics. First, standardizing the architecture in order to define and classify the main components and interfaces depending on their features and purposes. Then, reinforcing the security of IoT software, devices and the networks they're connected to in order to protect the user's data and infrastructure. And finally, building light software and communication protocols to cope with IoT constrained storage, processing and bandwidth resources. This coming decade will evidently revolve around the 3 Ds. Data, Discovery and Decision. Data is most relevant enabler for the future of IoT. As the amount of collected information grows, and the quality and precision of data analytics algorithms evolve, new requirements, devices and services can seamlessly be discovered, and decisions regarding their implementation can be determined.

6.4 IoT and globalization

IoT is bigger than us. It's bigger than one company. It's even bigger than one country. If the ethical and political implications of this acclamation are put aside, to reach its full potential, IoT ought to belong to everyone and device ought to be connected with everything across the globe. If one cannot drive his connected car across countries, seamlessly and without constraints and complications, smart self-driving car solutions cannot compete with current cars. If smart health is not globalized, including patient records and monitoring, a sick person could never comfortably travel abroad without having to worry about consequences. In other words, IoT should be for all, and all should be at its service.

7 Conclusion

The first ever application of the Internet of Things was created in 1990. Ever since, and for the last three decades, new applications, devices, standards, and approaches expanded the reach and significance of the internet of things paradigm. While several authors have proposed frameworks and blueprints to structure the growing knowledge and complexity, the main efforts remained on connectivity, security and data. This chapter is positioned as a reference IoT design Framework to assist the requirement specification process. The main idea of the framework is to assist IoT engineers in specifying reusable, client-tailored, self-adaptive, dynamic and cognitive IoT applications. The chapter illustrates these capabilities using the case of a smart car company that designs specific fleets of self-adaptive smart cars from a reusable and expandable set of functionalities. Furthermore, an approach for the specification of natural language requirements for IoT systems was elaborated. It describes on the one hand a typology for the requirements that

engineers often deal with while specifying this category of systems. On the other hand, it provides a template for a proper formulation of requirements at the specification phase, in order to increase precision and avoid ambiguity. The smart car case is used to provide explicit examples for the specification of the various requirement types. Furthermore, in order to assess the IoT specification current state, seven approaches from the literature and the industry were briefly discussed and mapped to the proposed framework. This process emphasized the current trends in IoT requirement specification, which mostly revolve around reusability and adaptation. It also revealed emerging areas of interest, especially in terms of self-learning and artificial intelligence capabilities.

References

Abbas, N., Andersson, J., & Löwe, W. (2010). Autonomic Software Product Lines (ASPL). *ACM International Conference Proceeding Series*, (January), 324–331.

Achtaich, A., Roudies, O., Souissi, N., Salinesi, C., & Mazo, R. (2019). Evaluation of the State-Constraint Transition Modeling Language: A Goal Question Metric Approach. *Software Product Line Conference Proceedings - Volume B*. Paris.

Antonino, P. O., Morgenstern, A., Kallweit, B., Becker, M., & Kuhn, T. (2018). Straightforward Specification of Adaptation-Architecture-Significant Requirements of IoT-enabled Cyber-Physical Systems. *Proceedings - 2018 IEEE 15th International Conference on Software Architecture Companion, ICSA-C 2018*, 19–26. IEEE.

Atzori, L., Iera, A., & Morabito, G. (2017). Understanding the Internet of Things: definition, potentials, and societal role of a fast evolving paradigm. *Ad Hoc Networks*, *56*, 122–140. Elsevier B.V.

Baccelli, E., Gundogan, C., Hahm, O., Kietzmann, P., Lenders, M. S., Petersen, H., Schleiser, K., et al. (2018). RIOT: An Open Source Operating System for Low-End Embedded Devices in the IoT. *IEEE Internet of Things Journal*, *5*(6), 4428–4440.

Basu, S., Karuppiah, M., Selvakumar, K., Li, K. C., Islam, S. K. H., Hassan, M. M., & Bhuiyan, M. Z. A. (2018). An intelligent/cognitive model of task scheduling for IoT applications in cloud computing environment. *Future Generation Computer Systems*, *88*(June), 254–261. Elsevier B.V.

Boehm, B. W. (1984). Verifying and Validating Software Requirements and Design Specifications. *IEEE Software*, *1*(1), 75–88.

Bowen, J. P., & Hinchey, M. G. (1999). *High-Integrity System Specification and Design*. *High-Integrity System Specification and Design*. Springer London.

de C Henshaw, M. J. (2016). Systems Of Systems, Cyber-Physical Systems, The Internet-Of-Things…Whatever Next? *INSIGHT*, *19*(3), 51–54. Wiley.

Chakraborty, A., Kanti Baowaly, M., Arefin, A., & Newaz Bahar, A. (2012). The Role of Requirement Engineering in Software Development Life Cycle. *Journal of Emerging Trends in Computing and Information Sciences*, *3*(5), 723–729.

Chang, V., Sharma, S., & Li, C. S. (2020). Smart cities in the 21st century. *Technological Forecasting and Social Change*, *153*. Elsevier Inc.

Costa, B., Pires, P. F., & Delicato, F. C. (2016). Modeling IoT Applications with SysML4IoT. *Proceedings - 42nd Euromicro Conference on Software Engineering and Advanced Applications,*

SEAA 2016, 157–164.

D'Ippolito, N., Braberman, V., Kramer, J., Magee, J., Sykes, D., & Uchitel, S. (2014). Hope for the Best, Prepare for the Worst: Multi-tier Control for Adaptive Systems. *Proceedings of the 36th International Conference on Software Engineering*.

Danny Weyns. (2017). Software Engineering of Self-Adaptive Systems: An Organised Tour and Future Challenges. *Handbook of Software Engineering*, 1–41.

Feynman, R., & Objectives, C. (2016). EBNF A Notation to Describe Syntax, 1–19.

Gartner. (2020). IoT Adoption Trends. Retrieved June 23, 2020, from https://www.gartner.com/en/innovation-strategy/trends/iot-adoption-trends

Hinai, S. Al, & Singh, A. V. (2018). Internet of things: Architecture, security challenges and solutions. *2017 International Conference on Infocom Technologies and Unmanned Systems: Trends and Future Directions, ICTUS 2017* (Vol. 2018–January, pp. 1–4). Institute of Electrical and Electronics Engineers Inc.

Hussein, M., Li, S., & Radermacher, A. (2019). Model-driven development of adaptive IoT systems. *CEUR Workshop Proceedings*, 17–23.

Hwang, K., & Chen, M. (2017). *Big-Data Analytics for Cloud, IoT and Cognitive Computing - Kai Hwang, Min Chen - Google Books*. John Wiley & Sons.

IEEE. (2009). *830-1998 - IEEE Recommended Practice for Software Requirements Specifications*. Retrieved March 1, 2020, from https://standards.ieee.org/standard/830-1998.html

INCOSE. (2018). Systems Engineering Handbook. *INSIGHT*, *1*(2), 20–20. Wiley. Retrieved February 9, 2020, from http://doi.wiley.com/10.1002/inst.19981220

International Telecommunication Union — ITU-T Y.2060. (2012). *Overview of the Internet of things*. *Next Generation Networks — Frameworks and functional architecture models*.

Iqbal, D., Abbas, A., Ali, M., Khan, M. U. S., & Nawaz, R. (2020). Requirement Validation for Embedded Systems in Automotive Industry through Modeling. *IEEE Access*, *8*, 8697–8719. Institute of Electrical and Electronics Engineers Inc.

Jones, C. B. (1995). *Systematic software development using VDM*. Prentice Hall International.

Karakostas, B. (2016). Event Prediction in an IoT Environment Using Naïve Bayesian Models. *Procedia Computer Science*, *83*(Ant), 11–17. Elsevier Masson SAS. Retrieved from http://dx.doi.org/10.1016/j.procs.2016.04.093

Khan, S., Dulloo Aruna B, & Verma, M. (2014). Systematic Review of Requirement Elicitation Techniques. *International Journal of Information and Computation Technology*, *4*(2), 133–138. Retrieved February 12, 2020, from http://www.irphouse.com/ijict.htm

Knauss, E., Boustani, C. E. I., & Flohr, T. (2009). Investigating the impact of software requirements specification quality on project success. *International Conference on Product-Focused Software Process Improvement* (Vol. 32 LNBIP, pp. 28–42). Springer Verlag.

Körner, S. J., & Brumm, T. (2009). Natural language specification improvement with ontologies. *International Journal of Semantic Computing*, *3*(4), 445–470. World Scientific Publishing Co. Pte Ltd.

Lahboube, F., Haidrar, S., Roudies, O., Souissi, N., & Adil, A. (2014). Systems of Systems Paradigm in a Hospital Environment: Benefits for Requirements Elicitation Process. *International Review on Computers and Software (I.RE.CO.S.)*, *9*(10), 1798–1806.

Lamsweerde, A. Van. (2009). *Requirements Engineering: From System Goals to UML Models to*

Software Specifications. *Change*.

Leite, J., Batista, T., & Oquendo, F. (2017). Architecting IoT applications with SysADL. *Proceedings - 2017 IEEE International Conference on Software Architecture Workshops, ICSAW 2017: Side Track Proceedings*, 92–99.

Lin, J., Yu, W., Zhang, N., Yang, X., Zhang, H., & Zhao, W. (2017). A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications. *IEEE Internet of Things Journal*, *4*(5), 1125–1142. Institute of Electrical and Electronics Engineers Inc.

Lunardi, G. M., Machot, F. Al, Shekhovtsov, V. A., Maran, V., Machado, G. M., Machado, A., Mayr, H. C., et al. (2018). IoT-based human action prediction and support. *Internet of Things*, *3–4*, 52–68. Elsevier B.V.

Maalem, S., & Zarour, N. (2016). Challenge of validation in requirements engineering. *Journal of Innovation in Digital Ecosystems*, *3*(1), 15–21. Elsevier BV.

Mahya, P., & Tahayori, H. (2016). IoT is SoS. *Int'l Conf. Internet Computing and Internet of Things* (pp. 38–42).

Mazo, R. (2018). *Software Product Lines, from Reuse to Self Adaptive Systems*. Université Paris 1 Panthéon - Sorbonne, France.

Mazo, R., Salinesi, C., Djebbi, O., Diaz, D., & Lora-Michiels, A. (2012). Constraints: the Heart of Domain and Application Engineering in the Product Lines Engineering Strategy. *International Journal of Information System Modeling and Design*, *3*(2). IGI Global.

Mohammadi, M., Al-Fuqaha, A., Sorour, S., & Guizani, M. (2018, October 1). Deep learning for IoT big data and streaming analytics: A survey. *IEEE Communications Surveys and Tutorials*. Institute of Electrical and Electronics Engineers Inc.

Muñoz-Fernández, J. C., Mazo, R., Salinesi, C., & Tamura, G. (2018). 10 Challenges for the specification of self-adaptive software. *Proceedings - International Conference on Research Challenges in Information Science*, *2018–May*(June 2019), 1–12.

Ngu, A. H., Gutierrez, M., Metsis, V., Nepal, S., & Sheng, Q. Z. (2017). IoT Middleware: A Survey on Issues and Enabling Technologies. *IEEE Internet of Things Journal*, *4*(1), 1–20. Institute of Electrical and Electronics Engineers Inc.

Nikolopoulos, B., Dimopoulos, A. C., Nikolaidou, M., Dimitrakopoulos, G., & Anagnostopoulos, D. (2019). *A System of Systems Architecture for the Internet of Things exploiting Autonomous Components*. *Int. J. System of Systems Engineering*.

OMG. (2017). *Unified Modeling Language Specification*. Retrieved February 12, 2020, from https://www.omg.org/spec/UML/About-UML/

Pohl, K. (2010). *Requirements engineering: fundamentals, principles, and techniques*. Springer Publishing Company, Incorporated.

Pohl, K. (2016). *Requirements Engineering Fundamentals, 2nd Edition: A Study Guide for the ... - Klaus Pohl - Google Books*. Rocky Nook.

Pohl, K., Böckle, G., & van der Linden, F. J. (2005). *Software Product Line Engineering. Foundations, Principles, and Techniques*. *Uwplatt.Edu* (Vol. 49).

Robertson, S., & Robertson, J. (2012). *Mastering the Requirements Process: Getting Requirements Right*. Addison-Wesley,.

Salinesi, C., Kusumah, I., & Rohleder, C. (2018). New Approach for Supporting Future Collaborative Business in Automotive Industry. *2018 IEEE International Conference on Engineering, Technology and Innovation, ICE/ITMC 2018 - Proceedings*. Institute of Electrical and Electronics

Engineers Inc.

Salinesi, C., Mazo, R., Djebbi, O., Diaz, D., & Lora-Michiels, A. (2011). Constraints: The core of product line engineering. *Fifth International Conference On Research Challenges In Information Science* (pp. 1–10). IEEE.

Sánchez-Arias, G., González García, C., & Pelayo G-Bustelo, B. C. (2017). Midgar: Study of communications security among Smart Objects using a platform of heterogeneous devices for the Internet of Things. *Future Generation Computer Systems*, *74*, 444–466. Elsevier B.V.

Sezer, O. B., Dogdu, E., & Ozbayoglu, A. M. (2018). Context-Aware Computing, Learning, and Big Data in Internet of Things: A Survey. *IEEE Internet of Things Journal*, *5*(1), 1–27. Institute of Electrical and Electronics Engineers Inc.

Soares, M., Jéssyka, V., Guedes, G., Silva, C., & Castro, J. (2017). Core Ontology to Aid the Goal Oriented Specification for Self-Adaptive Systems. *Advances in Intelligent Systems and Computing*, *571*, V–VI.

Souza, V. E. S., Lapouchnian, A., & Mylopoulos, J. (2012). (Requirement) evolution requirements for adaptive systems. *ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems* (pp. 155–164).

Souza, V. E. S., Lapouchnian, A., Robinson, W. N., & Mylopoulos, J. (2013). Awareness requirements. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *7475 LNCS*, 133–161.

Spivey, J. M. (1989). *The Z notation: a reference manual | Guide books*. Prentice-Hall, Inc.Division of Simon and Schuster One Lake Street Upper Saddle River, NJUnited States.

Uthariaraj, V. R., & Florence, P. M. (2011). QoS With Reliability And Scalability In Adaptive Service-Based Systems, 37–56.

Vassev, E. (2015). Requirements Engineering for Self-Adaptive Systems with ARE and KnowLang. *EAI Endorsed Transactions on Self-Adaptive Systems*, *1*(1), e6.

Videira, C., & Da Silva, A. R. (2005). Patterns and metamodel for a natural-language-based requirements specification language . *CAiSE'05* (pp. 189–194).

Whittle, J., Sawyer, P., Bencomo, N., Cheng, B. H. C., & Bruel, J. M. (2010). RELAX: A language to address uncertainty in self-adaptive systems requirement. *Requirements Engineering*, *15*(2), 177–196.

World Health Organization. (2020). *Coronavirus disease 2019 (COVID-19) Situation Report-72 HIGHLIGHTS*.

Yang, Q. L., Lv, J., Tao, X. P., Ma, X. X., Xing, J. C., & Song, W. (2013). Fuzzy self-adaptation of mission-critical software under uncertainty. *Journal of Computer Science and Technology*, *28*(1), 165–187.

Yang, Y., Li, X., Ke, W., & Liu, Z. (2019). Automated Prototype Generation From Formal Requirements Model. *IEEE Transactions on Reliability*, 1–25. Institute of Electrical and Electronics Engineers (IEEE).