D. Ferraris, and C. Fernandez-Gago, "TrUStAPIS: A Trust Requirements Elicitation Method for IoT", *International Journal of Information Security*, pp. 111-127, 2019.
http://doi.org/10.1007/s10207-019-00438-x
NICS Lab. Publications: https://www.nics.uma.es/publications

TrUStAPIS: A Trust Requirements Elicitation Method for IoT

Davide Ferraris[a], Carmen Fernandez-Gago[a]
[a]*NICS Lab*, *University of Malaga*, 29071 Malaga, Spain
{ferraris,mcgago}@lcc.uma.es

***Abstract*—The Internet of Things (IoT) is an environment of interconnected entities, which are identifiable, usable and controllable via the Internet. Trust is useful for a system such as the IoT as the entities involved would like to know how the other entities they have to interact with are going to perform. When developing an IoT entity, it will be desirable to guarantee trust during its whole life cycle. Trust domain is strongly dependent on other domains such as security and privacy. To consider these domains as a whole and to elicit the right requirements since the first phases of the System Development Life Cycle (SDLC) is a key point when developing an IoT entity. This paper presents a requirements elicitation method focusing on trust plus other domains such as security, privacy and usability that increase the trust level of the IoT entity developed. To help the developers to elicit the requirements, we propose a JavaScript Notation Object (JSON) template containing all the key elements that must be taken into consideration. We emphasize on the importance of the concept of traceability. This property permits to connect all the elicited requirements guaranteeing more control on the whole requirements engineering process.**

***Index Terms*—Trust, Internet of Things (IoT), Requirements Engineering, System Development Life Cycle (SDLC), JavaScript Notation Object (JSON)**

I. Introduction

The Internet of Things (IoT) is an environment of interconnected entities, which are identifiable, usable and controllable via the Internet [1].

In a system such as the IoT, trust is the basis of the communication between the smart entities. In fact, an entity should interact with another only if trust is established between them. Due to the uncertainty, interoperability and the heterogeneity of IoT achieving trust between is still a challenge. In addition, considering that these aspects have been tackled by isolated research communities, a holistic approach is desirable [2].

Requirements engineering is one of the first phases of the System Development Life Cycle [3] and Software Development Life Cycle [4] (in this paper we will refer to both of them as SDLC). Collecting requirements in the early phases of the SDLC is an important task that brings benefits to the following phases of the SDLC and avoid problems that could happen in later phases. The requirements are usually elicited by developers following stakeholders needs. The stakeholders are persons or companies having an interest in the system or software developed.

Existing requirements languages have been widely used with the introduction of Goal-Oriented methodologies [5–8] but they have not been developed for IoT and do not consider trust in relation to other security domains. Similarly, trust and related domains such as security, identity, usability and privacy were not considered properly in the first phases of SDLC [9]. On the contrary, in order to guarantee trust, it is important to consider also other domains related to it, as Hoffman [10] and Pavlidis [11] stated. Following this premise, Rios et al. [12] have proposed a work considering privacy in trust negotiation, and Gago et al. [2] moved forward considering both identity and privacy connected to trust in the IoT field.

We aim to continue in this direction considering trust-related domains holistically in IoT, extending the work in [13] where the authors designed a framework to ensure trust in an IoT entity during the whole SDLC.

In this paper, we propose a method to elicit requirements related to trust and the other domains identified. We define a JavaScript Notation Object[1] (JSON) template to help developers to elicit the requirements. Finally, we emphasise on an important property that could help the developer: traceability [13]. This property creates a connection among requirements and avoids domino effects or unintended consequences in case of relaxing connected requirements. Furthermore, traceability to the previous phases permits to justify why a requirement has been elicited.

The paper is structured as follows. Section II describes the background and the related work. In section III we introduce our requirements elicitation method, TrUStAPIS, which type of requirements we consider and how traceability works. In section IV we describe an IoT scenario to illustrate how to elicit requirements using TrUStAPIS. Finally, in section VI we conclude the paper and discuss the future work.

II. Related Work

A. Trust and Related Domains in IoT

IoT is a network of interconnected objects. Roman [1] states that the goal of IoT is to enable things to be connected anytime, anyplace, with anything and anyone, ideally using any path network and any service. It is expected that these entities will often have to interact with each other in uncertain conditions. Mechanisms to solve this lack of information are needed and trust can help address this need and overcome uncertainty.

Trust is a difficult concept to define "because it is a multi-dimensional, multidisciplinary and multifaced concept" [14].

[1]https://www.json.org

Jøsang [15] defines trust as personal and subjective, for McKnight [16] trusting someone means to depend on him, no matter the consequences. Typically, there are two entities (at least) involved in a trust interaction, one is the trustor (the entity which places trust) and the other is the trustee (the entity in which trust is placed).

According to Hoffman et al. [10] and Pavlidis [11] trust is strongly dependent on other domains like privacy, identity and security. From these definitions, Ferraris et al. [13] stated that, in an IoT entity development, it is important to centrally consider trust and related domains such as usability or identity. Trust is related to each of them and they cover all the aspects that can increase and guarantee trust in an entity.

In [17], usability is defined as "the capability of a product to be understood, learned, operated and is attractive to the users when used to achieve certain goals with effectiveness and efficiency in specific environments". In this work, the authors identified various characteristics to enhance the usability in the mobile device domain such as effectiveness, efficiency, satisfaction and reliability.

Mahalle et al. [18] have proposed identity features that are important to be taken into consideration such as authentication and authorization.

B. K-Model

In [13], the authors proposed a framework to guarantee trust in an IoT entity during the whole SDLC also considering connected properties. This framework is composed of the K-Model and transversal activities (documentation, metrics, gates, traceability, threat analysis, risk management and decision making). Furthermore, the context layer is always present in each phase. In an IoT environment, due to its dynamicity and heterogeneity, we always need to take context into consideration. The context can depend on the environment, on law regulations or on the rules of the company that develops the product. K-Model includes different phases to cover all the system life cycle of a product, from cradle to grave. The first one refers to the needs phase, where it is understood the purpose of the new IoT product and the stakeholders have a key role in it. After this phase, we define the requirements phase where the developer elicits the requirements considering all the important aspects of the previous needs.

The K-Model is shown in Figure 1. In this paper we focus on the second phase, but also the output of the first phase is fundamental to elicit the proper requirements. Moreover, the requirements must be verified and validated in the final phases of the K-Model. The arrows represent traceability among phases. As we will explain in section III-C, traceability is also important within phases, mostly in the requirement phase.

C. Requirement Engineering

In the state of the art, requirements engineering has been widely used with the introduction of Goal-Oriented methodologies. I* developed by Yu [8] introduced the notions of actor, goal and dependencies. SI* [6] is an extension of I* regarding security and including notions related to security and

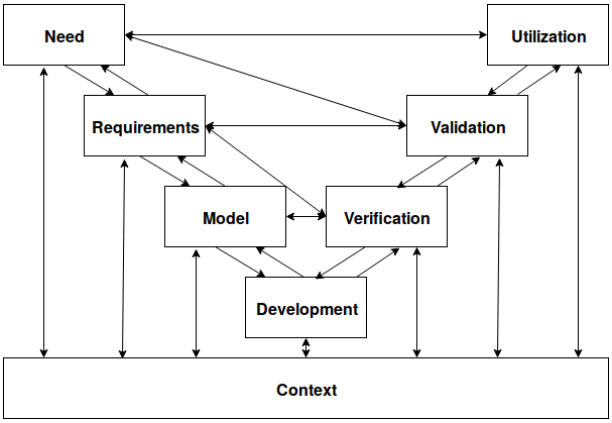

Fig. 1: K-Model [13]

trust. In addition, TROPOS [5], founded on the I* framework methodology, was intended to support all the design activities in the SDLC. Finally, Mouratidis and Giorgini extended the Tropos methodology using Secure Tropos [7] by making explicit which actor owns a service and is able to provide it. Considering trust-related domains, Rios et al. [12, 19] have highlighted how important is to consider privacy characteristics during requirements engineering to ensure trust in trust negotiation processes.

Mavropoulos et al. proposed a method to elicit security requirements for IoT using JSON [20]. They state that "using JSON format the process of requirements elicitation can be automated, thus making the analysis of large IoT networks more efficient".

Our work takes these works into consideration and goes further considering trust strongly related to other domains such as security, usability and identity. In addition, we introduce traceability to connect all these requirements among them. Traceability is a property that is absent in the languages previously mentioned. Finally, we propose a JSON template to help developers to elicit the proper requirements considering all the elements related to TrUStAPIS. All the requirements will be written following the rules proposed in section III.

In conclusion, in the state of the art, there are a lot of works related to eliciting requirements, but none of them takes into consideration trust concerning other domains, neither takes into consideration the dynamicity of IoT. This paper fills this gap and proposes a requirements elicitation method that helps developers to consider trust through the IoT SDLC.

III. TrUStAPIS: A Method for IoT Requirements Elicitation

A. Overview

As we outlined in section II, the second phase of the K-Model concerns requirements elicitation. To include trust in the SDLC, K-Model identifies a set of seven types of requirements related to seven domains: trust, usability, security, availability, privacy, identity and safety. They will be written according to the needs identified in the previous phase and will be elicited following the IEEE 830-1993 specification[21].

Each requirement must be elicited according to its domain characteristics, i.e. for trust we need to consider its transitivity and/or asymmetry and for privacy we need to consider anonymity and/or confidentiality. Another important consideration is that it is possible to have the same characteristics in different domains (i.e. confidentiality belongs both to privacy and security domains). Finally, it is possible that a requirement could have one or more sub-requirements. This is an important feature that helps the developers to specify a requirement using additional information for each of the sub-requirements.

To fulfil these needs, we introduce a requirement elicitation method named TrUStAPIS. Using this method, the developer could take into consideration trust and its related domains through the requirements elicitation process. TrUStAPIS is an acronym that comes from the use of the first letters of each of the seven domains considered: Trust (fully written because it is the central one), Usability, Security, Availability, Privacy, Identity and Safety. This method allows the developer to elicit the requirements regarding their domain and connect them through traceability. Moreover, it is possible to elicit the requirements considering dynamicity aspects related to IoT thanks to the context parameter. The method is composed of several elements: actors, actions and measures, goal and context.

- **Actor**. An *actor* can be a human or an IoT entity. It is the one that fulfils or asks to fulfil a goal to another actor. An actor could have different *roles* (i.e. in a trust domain, the actors are trustor and trustee). According to Gago et al. [2] humans can be considered as IoT entities. In the case of requirement elicitation process, we consider them separately because to elicit the proper requirements we shall distinguish the type of actor.
- **Action**. An *action* is related to the task performed by the actor. An action could have zero or more measures. A *measure* is a value related to its domain. It helps stakeholders and developers to model the right requirements that should be verified and validated in the later phases of the K-Model.
- **Goal**. A *goal* is the final purpose for which the requirement is being identified. It is achieved by actors through proper actions. This goal is related to a particular capability of the IoT entity and it depends on the requirement domain. Depending on the dynamicity of IoT, the same actor can have a different goal or perform different actions to fulfil it. Each goal is associated with a particular action, also called goalAction.
- **Context**. The *context* is strongly related to the requirements domain: trust, usability, security, availability, privacy identity and safety. Each of these domains could have its own characteristics. The context could change dynamically according to IoT paradigm and it is related to the environment (where the action is performed) and to the scope of the goal.

We propose a conceptual model in Figure 2 to show the relationships among the components of TrUStAPIS.

In this conceptual model, we present all the components related to TrUStAPIS method. Concerning the *actor*, we can see that it could be a human or an IoT entity and it is the subject of the requirement. Each actor plays a *role*, depending on the context. An *action* is the verb of the requirement and it could be performed to fulfil or to request a *goal*. In the first case, the object of the requirement is the goal. In the second case, the object of the requirement is another actor that can perform an action to fulfil the proposed goal. It is possible that an action could have zero or more *measures*. These measures are important to reach the goal and to verify and validate the requirements in the following phases of the K-Model. Finally, we can see that a *context* is composed of three components: a scope, an environment and a domain. The *scope* is related to the purpose of the goal an actor wants to fulfil. Then, the *environment* is related to the physical place where the action is performed. With the term *domain*, we refer to the seven types of requirements identified. Each domain could have its proper *characteristics* that we will present in the following subsections.

Relationships among concepts are given by the arrows. Each arrow contains some text that describe the dependence from one concept to another. The direction is that of the arrow. An optional dotted arrow is considered if a goal must be fulfilled by a secondary actor. The triangle represents specialization (i.e. an actor could be a human or an IoT Entity). Finally, the rhombus is a composition element (i.e. the context is always composed of a domain, an environment and a scope).

In addition, to help the developers to elicit the proper requirements, we propose a JSON template that is shown in Figure 3.

We have chosen JSON because it is schematic and it is supported by many languages such as Java[2]. It is easily readable by humans and machines. This aspect permits to share the requirements code between stakeholders and application [22]. Furthermore, it is possible to map the needs identified in the first phase of the K-Model into the JSON code and this implementation is useful in the following phases of the K-Model (model, development and verification), helping the developers to automatize the process.

The JSON template is composed of all TrUStAPIS main elements. The IoT requirement is always composed of a context, an actor, an action and a goal.

Iot Requirement : (Context, Actor, Action, Goal)

The context is divided into the requirement domain, the environment and the scope of the requirement. It is important to underline that the domain could be only one for each requirement. Each domain could have one or more characteristic that will define the requirement when it will be written.

Context : (Domain, Environment, Scope)

Each actor plays a role and falls into a type (human or IoT entity). All actors involved in the requirement must be set.

Actor : (Role, Type[Human, IoT Entity])

An action is composed of types and optional measures.

[2]https://www.java.com

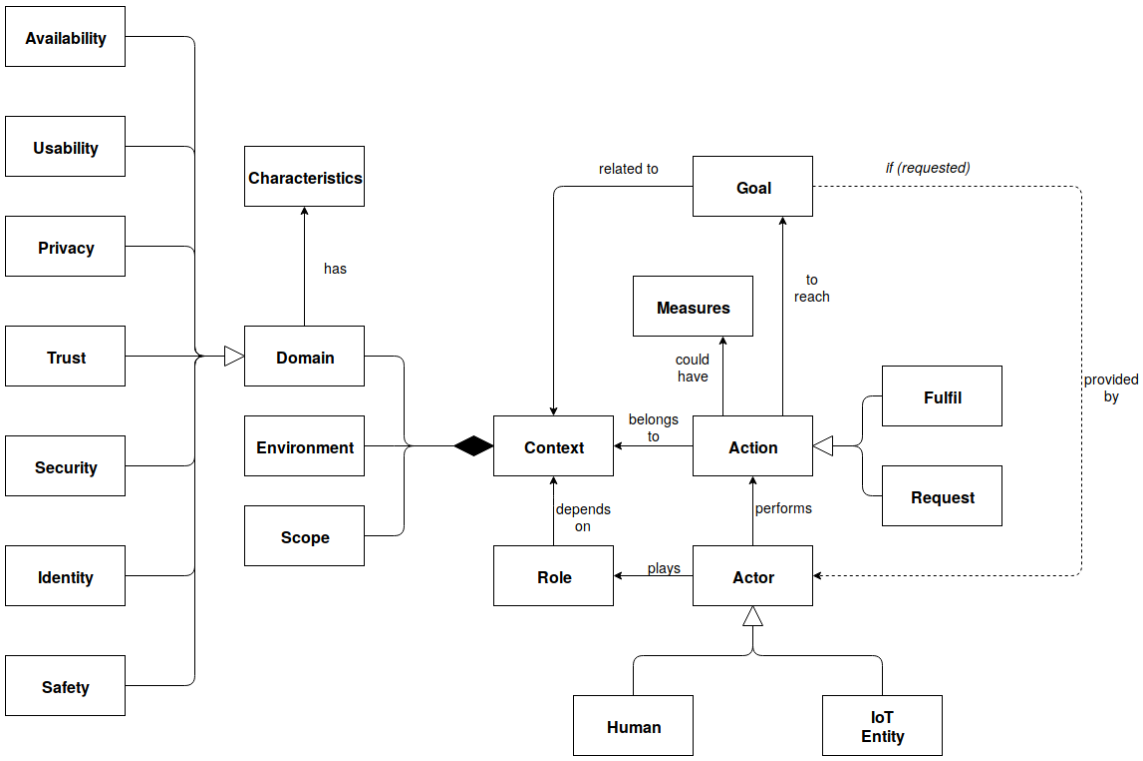

Fig. 2: Conceptual Model for TrUStAPIS

```
{
"IoT_requirement" : {
  "Context" : {
    "Domain" : [{
      "Trust" : {
        "Characteristic" : ["CharTrst_1", "..." ,"CharTrust_N"] } ,
      "Usability" : {
        "Characteristic" : ["CharUsab_1", "..." ,"CharUsab_N"] } ,
      "Security" : {
        "Characteristic" : ["CharSec_1", "..." ,"CharSec_N"] } ,
      "Availability" : {
        "Characteristic" : ["CharAvbt_1", "..." ,"CharAvbt_N"] } ,
      "Privacy" : {
        "Characteristic" : ["CharPriv_1", "..." ,"CharPriv_N"] } ,
      "Identity" : {
        "Characteristic" : ["CharIdnt_1", "..." ,"CharIdnt_N"] } ,
      "Safety" : {
        "Characteristic" : ["CharSft_1", "..." ,"CharSft_N"] } }] ,
    "Environment" : " " ,
    "Scope" : " " } ,
  " Actor " : {
    "Role" : " ",
    "Type" : [ "Human", "Iot Entity"] } ,
  "Action" : {
    "Type" : ["Fulfil", "Request"] ,
    "Measure" : ["Meas_1", "..." , "Meas_N"]},
  "Goal" : " "
  }
}
```

Fig. 3: TrUStAPIS JSON template to elicit requirements

Action : (Type[Fulfil, Request], Measure (optional))

Finally, the goal can help the developer to seek the connections beneath the requirements. In fact, it is possible that requirements belonging to different domains could be related if they have a similar or a same goal.

Following the JSON template schema, it is possible to elicit proper requirements. A written requirement need to be at least composed of one *actor*, a keyword ("*shall*") and a *goal* fulfilled by an *action*.

A written requirement could optionally have:

- One or multiple secondary actors performing an action to fulfil the goal.
- One or multiple actions necessary to perform the final goalAction.
- One or multiple measures. These values are useful for stakeholders and developers to model and verify/validate the requirements.

This structure is formalized by the following *statement (1)*.

(1) *Actor* **shall** *predicate*

The subject of *statement (1)* is also known as the main Actor.

We have decided to use *shall* as a keyword and not should or must because shall defines that the requirement is contractually binding, it must be implemented and verified/validated.

The *predicate* in its basic form is composed of a goalAction. In addition, it is possible to have more complex predicate. It could be composed of secondary actors, actions and/or measures. This composition is strictly dependent on the context.

In the following subsections, we explore and describe which characteristics must be taken into consideration when the developer writes a requirement specification for each requirement domain.

B. Requirements

In this section, we propose the seven requirements domains composing TrUStAPIS. It is important to state that some characteristic might be present in more than one domain.

Moreover, their descriptions will be different to better represent the domains they are belonging to. We will show how the characteristics are used in section IV.

1) Trust Requirements: Trust has some characteristics defined by many authors in state of the art [11, 23–30]. These characteristics will be used to write the proper trust requirements. We have highlighted the ones related to IoT.

1) **Direct**. Trust is based on direct experience between the trustor and the trustee. In this case, we can also say that trust is history-dependent because it depends on the past experience between the two actors. In this case, trust can also be subjective.
2) **Indirect**. When the trustor and the trustee have no past interactions, trust is based on the opinion and the recommendation of other entities. These entities must be trusted by the trustor. Objective trust and reputation are usually considered as indirect characteristic of trust.
3) **Transitive**. We can say that trust is conditionally transferable from an entity to another.
4) **Dynamic**. Trust is not static over time, even if it is not strictly time dependent.
5) **Local**. It depends on the couple trustor/trustee considered (i.e. Alice/Bob) and if we consider another couple (i.e. Alice/Charlie) it is possible that Alice does not trust Charlie, also if Bob trusts Charlie.
6) **Global**. This is useful when there is no direct knowledge of an entity, so a global reputation value is used to compute an initial trust value.
7) **Specific**. The trustor (i.e. Alice) can trust the trustee (i.e. Bob) for a particular action or service but not for other purposes. For example, we can say that Alice can trust Bob as a developer but not as a driver.
8) **General**. If needed, it is possible to compute a general trust value by aggregating all the specific values related to point 7.
9) **Asymmetric**. Trust can be asymmetric, this means that two entities tied to a relationship may trust each other differently, so the fact that Alice trusts Bob does not imply that Bob should trust Alice.
10) **Measurable**. Trust needs to be measured. This is important for trust modelling and computation.
11) **Composite-property**. Trust can be a composite property of different parameters: attributes (i.e. reliability, dependability, honesty) and characteristics (all the precedents). Each of these parameters may have different weights and must be computed even if it is measured through different metrics.

In this domain, the actors related to *statement (1)* are basically two and as we mentioned earlier they are named trustor and trustee. It is possible that another actor, or even more, are present as a trusted third party to fulfill a trusted goal. Finally, it is possible to have measures (i.e. trust thresholds) to decide whether to trust or not another entity.

2) Usability Requirements: Usability is an important property that can increase the level of trust of an IoT entity. Usability has a different meaning depending on whether the IoT entity is a person or an object. For this reason, the context is fundamental to decide which characteristic is more important to elicit the proper usability requirements. According to [17], we refine usability related characteristics that are important for IoT. They are:

1) **Effectiveness**. It is important to achieve the desired result.
2) **Efficiency**. It is important to save resources and speed up the communications between IoT entities.
3) **Simplicity**. It is important to reduce the complexity of communication between IoT entities. This is also an important characteristic for the users. If they have to interact with a device, a simple user interface improves usability.
4) **Understandability**. In IoT, there are many communication protocols such as ZigBee[3] or Zwave[4]. It is important to consider this aspect early in the SDLC to elicit the proper requirements. In fact, knowing the different protocols used for IoT communication it is possible to implement them or not according to the final IoT architecture (i.e. distributed or centralized) giving to the IoT entity the ability to communicate with the others. Also for the user, an understandable user interface increases the usability.
5) **Accessibility**. To grant accessibility via the internet is an aspect that increases the stability of an IoT entity but it raises security issues. For example, to be able to access to a smart home IoT entity even from outside ensures accessibility of the device improving its usability (i.e. not only at home).
6) **Flexibility**. To make an IoT entity connectible and reachable from different users and devices increases its usability.
7) **Reliability**. This is an important aspect strongly related to trust. Increasing the reliability of an IoT entity improves its usability and increases its trust level.

Usability is strongly connected to the interface of the developed IoT entity. A device-to-human interface is different from a device-to-device one, so it is important to consider the proper actor (human or IoT entity) to elicit the right requirement. Thus, it is very important to consider how to increase usability for the proper actor involved.

3) Security Requirements: In IoT field, as stated before, it is important to take into consideration the dynamicity, the heterogeneity and the context during the requirement elicitation process. To guarantee security in IoT it is mandatory to consider this fundamental aspect of IoT.

According to [31, 32], we take into consideration the following security characteristics:

1) **Authentication**. It is important that each entity is authenticated.
2) **Authorization**. An entity must be authorized to perform an action.

[3]http://www.zigbee.org/
[4]http://www.z-wave.com/

3) **Integrity**. The integrity of the data is a fundamental characteristic that increases the trust level of the IoT entity.
4) **Confidentiality**. The communication between IoT entities must be confidential (depending on the context).
5) **Delegation**. In IoT, more than non-delegation, it is important to delegate rights to other entities. This delegation is strictly related to the context and could be user-dependent or time-dependent.
6) **Non-repudiation**: it is important that the action performed could be stored to analyze which entities have been involved in it.

All the previous security characteristics, according to the context of the IoT entity, must be taken into consideration during the security requirements elicitation process.

Considering *statement (1)*, we can state that the actor shall perform or request a security action to fulfil a security goal. Examples of security measures can be security levels (i.e. usually expressed in bits) or types of encryption.

4) Availability Requirements: Availability is considered as a characteristic of security in other works such as [10, 11, 32]. Along with Confidentiality and Integrity they are known as the CIA triad. We have decided to put it in a self-domain because it is related also to other aspects (i.e. identity, usability) and also dependent on the hardware. The type of availability is strongly related to the context and the dynamicity of IoT. According to this, the availability of a service can be dependent on a particular period of time or service. To guarantee that availability could improve the level of trust, it is fundamental to grant the service when it is needed. This requirement can be functional or non-functional [21]. The main actor, in this case, is the IoT entity and the goalAction is to provide a service even if something wrong has occurred. As a measure, it is useful to have a threshold to raise the alarm or a warning to the user and perform the corrective actions.

Considering the work proposed by Farooq et al. [33], we identified several availability characteristics that are important for IoT:

1) **Resilience**. Resilience is the ability to keep working and recover quickly even if bad conditions occur (i.e. malfunctioning, attacks). It guarantees availability
2) **Scalability**. The IoT is an environment extremely dynamic. Scalability is required to maintain the availability of the services produced by the entity.
3) **Redundancy**. It refers to the replication of the information. It is strictly connected to scalability.
4) **Integrity**. Data and devices could be available only if their integrity is not compromised.

5) Privacy Requirements: High privacy could ensure customer's trust but can be a problem for the vendor's trust. In fact, a vendor should prefer to collect information on the users for business purposes and, on the contrary, a customer does not prefer to spread personal information.

To elicit privacy requirements for an IoT entity, according to [31, 34], we take into consideration the following characteristics of privacy:

1) **Anonymity**. It should be important for the entities to remain anonymous during determined actions.
2) **Pseudonymity**. This characteristic can be important in case it is not possible to remain anonymous.
3) **Undetectability**. This characteristic guarantees to the entity to not be sufficently distinguished, it is a fundamental characteristic to avoid attacks.
4) **Unlinkability**. The user could not be linked to particular data. Together with Anonymity, it guarantees Unobservability.
5) **Confidentiality**. The communication between entities could be encrypted to guarantee confidentiality.

These characteristics of privacy comply with the General Data Protection Regulation (GDPR) [5].

Privacy requirements could arise in conflicts with identity requirements. Depending on the context and the IoT environment it is possible to consider privacy more than identity or vice versa.

In this domain, the actors presented in *statement (1)* shall perform actions to fulfill a privacy action. So, the main actor shall guarantee privacy or avoid privacy issues considering the characteristics highlighted earlier. The measures related to privacy could be helpful to monitor privacy levels (i.e. according to differential privacy [35]) and whether they are respected or not.

6) Identity Requirements: Identity is very important for IoT entities. To know the interacting entities is a basis to trust them. Identity is strongly related to privacy. The more is known about an entity the less its privacy is guaranteed. For this reason, it is very important, since the early phases of the SDLC, to decide and apply how much information it is possible to obtain regarding an entity.

According to [18], we identify the following identity characteristics to elicit the proper identity requirements in IoT:

1) **Authentication**. This is a security characteristic, it is important also for identity management because if an actor is authenticated the system can identify and trust or not trust him/her;
2) **Authorization**. It is related to security and authentication, once authenticated, an entity could be authorized to perform an action.
3) **Attributes**. Attribute exchange is very important in identity management because through attributes it is possible to perform identification (often preserving privacy).
4) **Interoperability**. Identity management enhances interoperability between IoT entities because if an entity is identified could interact with the other IoT entities.
5) **Storable**. Identity information must be stored and protected.
6) **Manageable**. Identity data must be manageable by the IoT entity. We intend manageable generally. It could be related to the computation of user data or to store them.

[5]https://gdpr-info.eu/

7) **Scalability**. In a particular context, scalability permits to have a resilient way to store and manage identity information.
8) **Accountability**. Through identity management, it is possible to recognize which actor has performed an action. This characteristic permits to guarantee accountability.

Identity and privacy could raise conflicts between their domain requirements, so context, traceability and specification of the identity characteristics during the requirement elicitation process help to solve them or permits to discover them early in the SDLC. Anyway, it is out of the scope of this work to solve conflicts among requirements, it will be done in future work. The main actor presented in *statement (1)* must fulfill an identification action following one or more of the identity aspects highlighted before. In this case, the secondary actor could verify the identity properties of the first actor authenticating the identity attributes of the actor (i.e. code, id).

7) Safety Requirements: Safety is defined by the Oxford dictionary[6] as "The condition of being protected from or unlikely to cause danger, risk, or injury".

Safety depends on people and machinery [36], basically, it is related to the physical point of view of an IoT smart object. The parameters that must be taken into consideration are the ones related to avoiding the possibility to damage the actors involved.

The characteristics of safety are basically related to the hardware level and physical functionalities. In IoT, it is important because all the devices have embedded software and its utilization is strongly dependent on the device itself.

We propose the following safety characteristics according to [36–38]:

1) **Feedback**. The users must be aware of malfunctions, so the IoT entity shall provide to him/her information about its status. This characteristic could prevent harmful situations.
2) **Protection**. On one hand, the entity must be preserved from physical damages. On the other hand, the users must be preserved by the entity's process that could be dangerous for them.
3) **Resilience**. Resilience is the capacity of a system to continue its work also under attacks or malfunctions. It is important for safety of the device and the users.
4) **Integrity**. This characteristic is related to the physical integrity of the device. Through protection, IoT entity shall be preserved from external and internal damages that can compromise its working state.

Safety, if maximized, can improve the level of trust of an IoT device. Like the other domains, it is strongly dependent on the context (in particular on the environment). The safety goal shall guarantee safety for all the actors involved for example to prevent the IoT entity to overheat.

[6]https://en.oxforddictionaries.com/definition/safety

C. Traceability

Our way to use traceability was introduced in [13] but we will expand it in this paper going deeply in its use and definition. Traceability permits to connect phases forward and backward between them in the K-Model. In addition, it is particularly important in the requirements engineering phase because it permits to connect requirements to the corresponding documents, models or other requirements. The traceability permits to avoid unintended consequences or domino effects due to the deletion of a particular requirement. In fact, the deletion could affect other connected requirements. Using traceability it is possible to be aware of these connections and to relax or modify them according to the new asset.

In IEE 830-1993 [21] two types of traceability have been proposed: Backward Traceability (BT) and Forward Traceability (FT). BT permits to trace the source of a requirement. It can be a need, a document or another requirement as stated in [13]. FT leads to children requirements, model specifications or system features.

According to the K-Model, we identify another type of traceability: Inner Traceability (IT). By IT we mean the one between the requirements of different domains. So we can connect a trust requirement to a privacy one and this connection needs to be specified. While we use BT to connect a requirement with its parent-requirement and FT to connect a requirement with its sub-requirements, we consider IT to connect requirements of the same level.

It is easier to guarantee BT, FT and IT with a unique identification number. The connections between requirements will be mapped in the same requirement data. How a requirement is composed and saved in its domain database is shown in Table I.

In the first cell, there is the field *ID*. This is the identification number of the requirement. The ID is unique and depends on the type of requirement. For example, an identity requirement ID is "IDNT-XX" and a security requirement ID is "SEC-XX" where XX is the number of the requirement.

The second cell is related to the requirement specification. This is the text of the requirement and it must be written following the IEEE 830-1993 standard [21] and considering the TrUStAPIS paradigm proposed in *statement (1)*. This field is very important and permits developers to understand what the IoT entity shall do to fulfil the needs elicited in the first phase of the K Model. The third, fourth and fifth cells are related to BT, FT and IT. They are the IDs of the other related requirements.

The requirements must be saved in a proper database, containing all the requirements divided in tables related to their domains (i.e. there will be a table containing all the trust requirements, another one related to the privacy requirements and one for each other type of requirements).

TABLE I: Requirement composition

ID	Requirement specification	BT	FT	IT

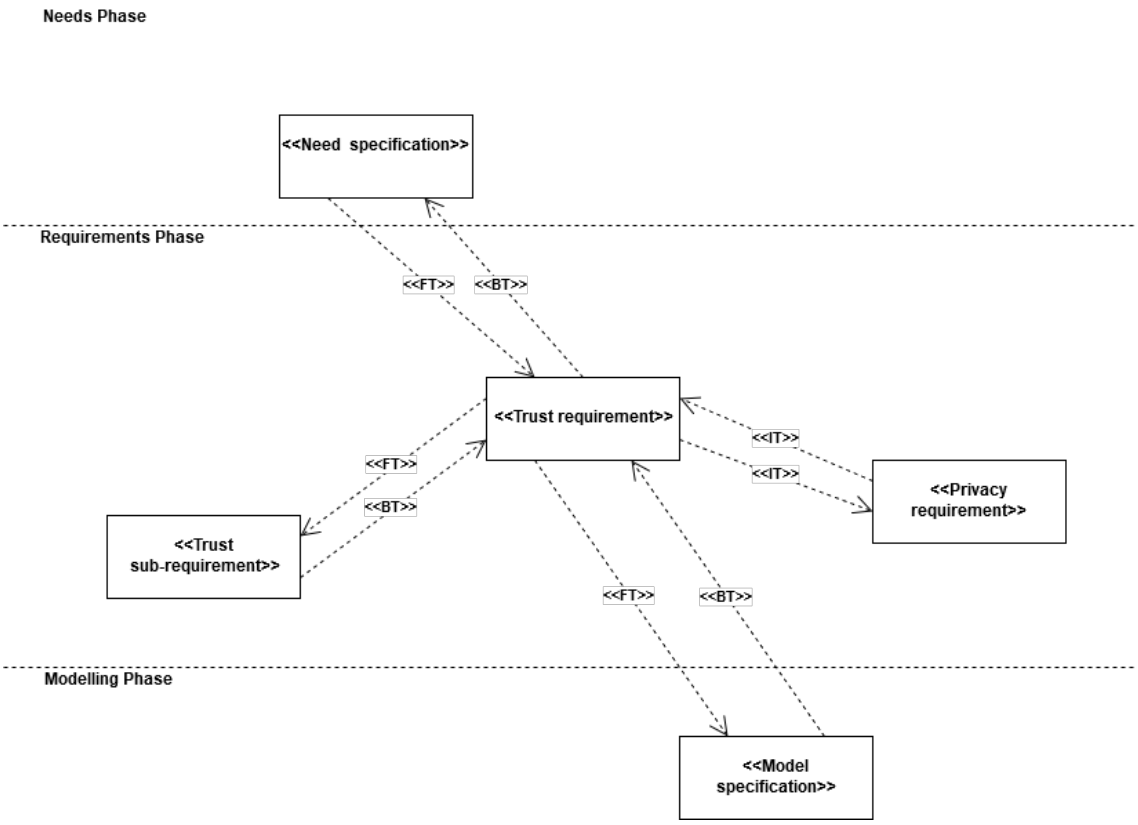

Fig. 4: Requirements Relationships

Figure 4 shows the relationships between requirements of the same type (parent and child), different type (Trust requirement and Privacy requirement) and connected phases of the framework. These connections are critical because they can map why a decision has been made. In addition, in case modification is needed and a requirement must be changed, it is possible to understand which other requirements will be affected by this modification. Knowing this information, it is possible to avoid problems in later phases. Figure 4 is an example where a trust requirement is shown (in the centre) that has connections with a specific need, its sub-requirement and a privacy requirement. In case the central requirements must be deleted, this change will affect the other requirements connected and the proposed need will be not fulfilled. In case of requirements modification, the traceability is important in order to follow the impacts that this change could have on the connected requirements and perform the corresponding corrective action (i.e. delete or modify the connected requirements).

D. Methodology

In this section, we present a step-by-step format about how to apply the TrUStAPIS methodology. The steps are shown in Figure 5.

The output of each step is used during the following steps.

1) The first step is related to the output of the previous phase of the K-Model (i.e. Need). This output is provided through the documentation containing related to the smart entity under development according to the stakeholders' needs.
2) As the second step, according to the needs, the developer must fill the JSON template proposed in Figure 3 considering the conceptual model elements shown in Figure 2. The conceptual model helps the developer to order the idea of writing the requirement and to fill correctly the JSON code. Both are useful to write

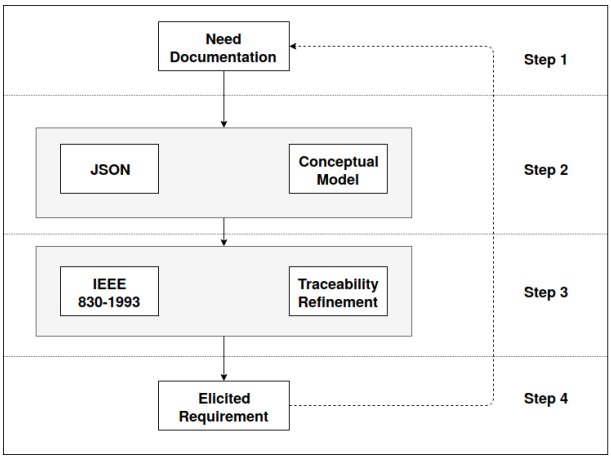

Fig. 5: Requirements elicitation: step-by-step methodology

the proper requirements. In addition, the JSON code will be considered as an input for the following phases of the K-Model (i.e. Verification). Through, the JSON code it will be possible to automatize the requirement elicitation process too as proposed by [20]. Anyway, in our approach, we will use it to order the idea and to give to the developer a tool to elicit requirements.

3) The third step is about writing and eliciting the requirement, they must comply with the IEEE 830-1993 formalism [21]. According to this recommended practice, a requirement must be correct, complete, unambiguous, complete, consistent, ranked, verifiable, modifiable and traceable. *Statement* (1) guarantees that the requirement is complete and it helps the developer to write a correct, unambiguous and consistent requirement. The JSON template guarantees that a requirement could be verified automatically in the following phases of the SDLC. In the case of modification either the written requirement and the JSON must be changed. A modification could be made because of conflict arisen with another requirement or because of a stakeholder need change. Anyway, in this paper we do not consider these two cases. Finally, as we explained in section III-C traceability is guaranteed in TrUStAPIS through BT, FT and IT. The JSON code helps the developers understanding which traceability must be considered among requirements. In fact, if two or more requirements have a common goal or related characteristics (i.e. the same one) they should have a traceability relationship. In the case of the traceability is among different domains it will be an IT type. Because a requirement must be correct, complete, consistent and unambiguous it could be helpful to create sub-requirements to specialize it. In this case, the traceability relationship will be a BT or FT one.
4) The fourth step of the methodology is related to the final elicited requirement, compose as shown in Table III. As we can see, there is a dotted line moving from Step 4 to Step 1. This arrow is needed because it is possible that a requirement could be rewritten according to new or modified needs. In this case, the developer rewrites the requirement following the steps as shown before.

This systematic methodology helps the developers to follow a guideline that reduces the subjectivity of the experts eliciting the requirements.

In the next section, we provide a scenario to show how TrUStAPIS method must be implemented.

IV. Scenario: Smart Cake Machine

In this section we exemplify the methodology presented in Section III-D in a use case of a Smart cake machine.

A. Step 1

Let us expand the use case scenario regarding a Smart Cake Machine (SCM) introduced in [13]. According to that, we have assumed that analyzing the needs documentation, it is possible to understand that the customers need an SCM that tells them which ingredients are necessary in order to bake a cake. The temperature of the SCM must be checked and it could not overcome 250°C. The recipes must be downloadable from the vendor website or inserted manually by authenticated users. Authentication must be done by a user name and a password. The SCM could interact with a Smart Fridge (SF) to check whether a particular ingredient is in the fridge or not. If not, the SCM could interact with a trusted Smart Supermarket (SM) through the home Smart Hub (SH) and order the missing ingredient. The communication among the smart home entities must be guaranteed and encrypted. These relationships are shown in Figure 6.

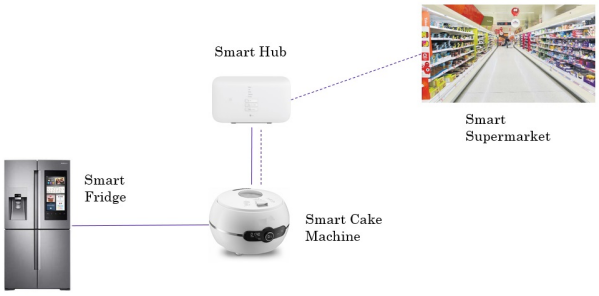

Fig. 6: Smart Cake Machine and its relationships with the other IoT entities

The dotted line between the SCM and the SH and between the SH and the SM are related to the fact that there is an indirect connection between the SCM and the SM. In fact, the SCM must delegate the SH to interact with the SM.

Considering these needs and the context, we can elicit the proper requirements using the TrUStAPIS method. This phase is of fundamental importance to continue the development of the SCM. According to the K-Model, the output of this phase will be the input for the modelling phase.

B. Step 2

After the need documentation is analyzed in the previous step, we continue to step 2.

Following the JSON template that is shown in Figure 3, we set up the JSON code related to our use case and requirements.

Using the JSON template and the conceptual model, it is possible to fill the JSON according to the needs identified before. The JSON code presented in Figures from 7 to 11 contains the parameters related to each requirement. These parameters are elicited from the needs.

We model the relationship between the SCM and the SM. To assign a trust value we need to consider some characteristic of trust (i.e. direct, measurable) and the type of action (i.e fulfill). This trust value will be used by the SCM to decide whether to trust or not to trust the SM.

We analyze next the JSON code in Figure 7 (from line 2 to 16). The specific identifier related to the requirements is shown in rows 2.

*row*2 – **"IoT_requirement_TRST01"**

```
1  {
2    "IoT_requirement_TRST01" : {
3      "Context" : {
4        "Domain" : [{
5          "Trust" : {
6            "Characteristic" : ["Direct","Indirect","Global","General","Measurable"]
                 } }],
7        "Environment" : "Smart City",
8        "Scope" : "Establish trust between SCM and SM" } ,
9      "Actor" : {
10       "Role" : ["Trustor, Trustee"],
11       "Type" : ["SCM", "SM"] },
12     "Action" : {
13       "Type" : ["Fulfill"] ,
14       "Measure" : ["Trust level"]},
15     "Goal" : "To fix a trust value"
16   },
17   "IoT_requirement_USAB01" : {
18     "Context" : {
19       "Domain" : [{
20         "Usability" : {
21           "Characteristic" : ["Simplicity, Understandability"] } }],
22       "Environment" : "Smart Home",
23       "Scope" : "User Interface" } ,
24     "Actor" : {
25       "Role" : "User",
26       "Type" : ["Human User"] },
27     "Action" : {
28       "Type" : ["Fulfill"] ,
29       "Measure" : [" "]},
30     "Goal" : "Let the user insert new recipes"
31   },
```

Fig. 7: JSON code part 1

As we mentioned earlier, the trust requirements has some characteristics (row 6) needed to compute the trust value. These characteristics must highlight particular aspects that we want to represent with the elicited requirements.

*row*6 – **"Characteristic" : ["Direct", "...", "Measurable"]**

In this case, the characteristics are: direct, indirect, global, general and measurable. We use direct, indirect and global to define that the trust level must be computed. A subjective value for the SCM (depending on the past interactions with the specified SM) is known as a direct trust. Then, an objective value is needed. There are two kinds of objective values: an indirect value and a global value. The indirect value is computed from known entities, a global value is computed by a centralized authority according to the trusted entities of the system. In this case, an indirect value could be the one computed by the SF (in the case it has a past relationship with the SM) and the global value could be computed by the vendor website (collecting the data of all the SCM interacting with an SM). Because an IoT environment is dynamic, it is important to take into consideration the possibilities that other entities could join or leave the network [39]. General is related to the fact that it is possible to have a single trust value for each entity, in this case, for example, the SM could have a single computed value related to its trust level even if there are different important parameters (i.e. time, distance, price, quality). We do not discuss now how these values could be computed, but it is important to underline that to compute trust we need trust metrics. For this reason, the final trust characteristic taken into consideration is measurable.

As we can see in row 10, the roles are both a trustor and a trustee.

*row*10 – **"Role" : ["Trustor", "Trustee"]**

In fact, in our case, the SCM is the trustor (the entity that orders the ingredients) and the Trustee is the SM (the entity that must provide the ingredients).

With this requirement, we basically want to fix a trust value (goal: row 15) between the two IoT entities: the SCM and an SM (row 11). The action is a goalAction because the type is *fulfill* (row 13) and it has a measure (row 14) computed from the characteristics defined before.

*row*11 – **"Type" : ["SCM", "SM"]**
*row*13 – **"Type" : ["Fulfil"]**
*row*14 – **"Measure" : ["Trust level"]**
*row*15 – **"Goal" : To fix a trust value**

The second requirement that we model from the needs is a usability requirement:

*row*17 – **"IoT_requirement_USAB01"**

It is represented in Figure 7 from lines 17 to 31.

This usability requirement is composed by the sentence "The recipes must be downloadable from the vendor website or inserted manually by authenticated users." Because a requirement must be complete, we must create (at least) two separate requirements, one for the "download" part, the other one for the "manual" part. In this case, we focus only on the second part, so the requirement USAB01 will be elicited according to this need. For a user, it is important that the procedure of inserting new recipes must be simple and the user interface understandable. For this case, we do not decide how the recipes must be inserted (i.e. by a smartphone, a website, the user interface of the SCM), we only model that the user shall be able to insert new recipes. To have more specific requirements, sub-requirements are needed.

*row*21 –
"Characteristic" : ["Simplicity", "Understandability"]
*row*21 – **"Goal" : ["Let the user insert new recipes"]**

```
32   "IoT_requirement_SEC01" : {
33     "Context" : {
34       "Domain" : [{
35         "Security" : {
36           "Characteristic" : ["Authentication"] } }],
37       "Environment" : "Smart Home",
38       "Scope" : "User Authentication" } ,
39     "Actor" : {
40       "Role" : ["User", "IoT Entity"],
41       "Type" : ["Human User","SCM"] },
42     "Action" : {
43       "Type" : ["Fulfill"] ,
44       "Measure" : [" "]},
45     "Goal" : "To authenticate the user"
46   },
47   "IoT_requirement_SEC02" : {
48     "Context" : {
49       "Domain" : [{
50         "Security" : {
51           "Characteristic" : ["Delegation"] } }],
52       "Environment" : "Smart Home",
53       "Scope" : "Rights and delegation" } ,
54     "Actor" : {
55       "Role" : ["Delegator", "Delegatee"],
56       "Type" : ["SCM", "SH"] },
57     "Action" : {
58       "Type" : ["Request", "Fulfill"] ,
59       "Measure" : [" "]},
60     "Goal" : "To delegate another IoT entity because of rights limitations"
61   },
```

Fig. 8: JSON code part 2

In Figure 8, we have two security requirements. One is related to the authentication characteristic (row 36) and the

other one related to delegation (row 51). According to the needs, the authentication one is related to the fact that a user, to interact with the SCM, must be authenticated. The second requirement is related to delegation, in fact, the needs document specifies that the SCM must communicate with the SM only through the SH, so in this case the SCM delegates the SH to order the ingredients.

*row*36 – **"Characteristic" : ["Authentication"]**
*row*51 – **"Characteristic" : ["Delegation"]**

Concerning the actors, in SEC01 the actors are the human user and the SCM (row 41). Whereas, in SEC02 the actors are two and they are the SCM and the Smart Hub (row 56).

*row*41 – **"Type" : ["Human User", "SCM"]**
*row*56 – **"Type" : ["SCM", "Smart Hub"]**

```
62  "IoT_requirement_AVBT01" : {
63    "Context" : {
64      "Domain" : [{
65        "Availability" : {
66          "Characteristic" : ["Resilience"] } }],
67      "Environment" : "Smart Home",
68      "Scope" : "Connectability between entities" } ,
69    "Actor" : {
70      "Role" : ["Connector, Connectee"],
71      "Type" : ["SCM", "Smart Hub"] },
72    "Action" : {
73      "Type" : ["Fulfill"] ,
74      "Measure" : [" "]},
75    "Goal" : "To provide connection between the IoT entities"
76  },
77  "IoT_requirement_PRIV01" : {
78    "Context" : {
79      "Domain" : [{
80        "Privacy" : {
81          "Characteristic" : ["Confidentiality"] } }],
82      "Environment" : "Smart Home",
83      "Scope" : "Information excahnge between entities" } ,
84    "Actor" : {
85      "Role" : ["Encryptor, Decryptor"],
86      "Type" : ["SCM", "Smart Fridge"] },
87    "Action" : {
88      "Type" : ["Fulfill"] ,
89      "Measure" : [" "]},
90    "Goal" : "To guarantee an encrypted communication between the IoT entities"
91  },
```

Fig. 9: JSON code part 3

According to the needs, we can state that for the availability requirement the important characteristic is *Resilience*. In fact, to guarantee the availability of the communication between the entity, resilience guarantees to be available also in the case of malfunctions or attacks. In this case, we require only that the communication is available. How resilience is guaranteed will be implemented in the following phases of the SDLC or by other sub-requirements. The "scope" is related to the connectability between IoT entities (row 68) and the "goal" is related to providing the connection between the IoT entities (row 75). In the first part of Figure 9, we can see the JSON related to the availability requirement.

*row*66 – **"Characteristic" : ["Resilience"]**
*row*68 – **"Scope" : "Connectability between entities"**
*row*75 –
"Goal" : "To provide connection between the IoT entities"

In the second part of Figure 9, it is represented the privacy requirement. It is elicited by the need that expresses that the communications must be encrypted.

We model the requirement considering the confidentiality characteristic (row 81) and both the IoT entities involved (row 86) have the roles of "Encryptor" and "Decryptor" (row 85).

*row*81 – **"Characteristic" : ["Confidentiality"]**
*row*85 – **"Role" : "Encryptor, Decryptor"**
*row*86 – **"Type" : ["SCM, Smart Fridge"]**

```
92  "IoT_requirement_IDNT01" : {
93    "Context" : {
94      "Domain" : [{
95        "Identity" : {
96          "Characteristic" : ["Authentication"] } }],
97      "Environment" : "Smart Home",
98      "Scope" : "User Authentication" } ,
99    "Actor" : {
100     "Role" : "User",
101     "Type" : ["Human User"] },
102   "Action" : {
103     "Type" : ["Fulfill"] ,
104     "Measure" : [" "]},
105   "Goal" : "To authenticate the user"
106 },
107 "IoT_requirement_IDNT01.1" : {
108   "Context" : {
109     "Domain" : [{
110       "Identity" : {
111         "Characteristic" : ["Authentication"] } }],
112     "Environment" : "Smart Home",
113     "Scope" : "Username and Password Authentication" } ,
114   "Actor" : {
115     "Role" : "User",
116     "Type" : ["Human User"] },
117   "Action" : {
118     "Type" : ["Fulfill"] ,
119     "Measure" : [" "]},
120   "Goal" : "To authenticate the user by username and password"
121 },
```

Fig. 10: JSON code part 4

The authentication need expressed in the Set-Up phase must be covered also by identity requirements. In Fig 10, we can see that we modeled two identity requirements, where the second one is a sub-requirement of the first one (rows 92 and 107).

*row*92 – **"IoT_requirement_IDNT01"**
*row*107 – **"IoT_requirement_IDNT01.1"**

According to the need, we model the sub-requirement specifying that the authentication must be performed by username and password. An important aspect is that the goal of the first identity requirement is the same as the first security requirement shown in Figure 8. This is because the authentication characteristic belongs to both the domains. These connections are taken into consideration concerning the traceability as we will see later.

Finally, in Figure 11 we can see two JSON codes related to safety requirements. As for the identity requirements, the second one is the specification of the first one. The need expressed here is related to the temperature. In fact, according to the needs, the SCM must check the temperature level and it cannot overcome 250°C.

C. Step 3

After the second step (see section III-D) is complete, we write down the requirements starting from the JSON codes shown in Figures 7 to 11. These requirements will be written following IEEE 830-1993 and *statement (1)* formalism. Traceability is considered according to common goals, characteristics and sub-requirements. The output of this third step will be the final elicited requirement. The requirements are

```
"IoT_requirement_SFT01" : {
  "Context" : {
    "Domain" : [{
      "Safety" : {
        "Characteristic" : ["Integrity"] } }],
    "Environment" : "Smart Home",
    "Scope" : "Temperature of the device" } ,
  "Actor" : {
    "Role" : "Checker",
    "Type" : ["SCM"] },
  "Action" : {
    "Type" : ["Fulfill"] ,
    "Measure" : ["Temperature Level"]},
  "Goal" : "To check the level of the temperature"
},
"IoT_requirement_SFT01.1" : {
  "Context" : {
    "Domain" : [{
      "Safety" : {
        "Characteristic" : ["Integrity"] } }],
    "Environment" : "Smart Home",
    "Scope" : "Temperature of the device" } ,
  "Actor" : {
    "Role" : "Checker",
    "Type" : ["SCM"] },
  "Action" : {
    "Type" : ["Fulfill"] ,
    "Measure" : ["Temperature Level"]},
  "Goal" : "To fix the maximum value of the temperature in Celsius degree"
}
}
```

Fig. 11: JSON code part 5

shown in Table II. We have created a requirements database using Microsoft Access.

TABLE II: Requirements elicited using TrUsTAPIS

Trust Req.	**TRST01** - The SCM shall trust a Smart Supermarket with a trust level above 0.5
Usability Req.	**USAB01** - The user shall be able to insert new recipes
Security Req.	**SEC01** - The user shall be authenticated
Security Req.	**SEC02** - The SCM shall delegate the Smart Hub to order the missing ingredients
Availability Req.	**AVBT01** - The SCM shall be able to connect to the Smart Hub
Privacy Req.	**PRIV01** - The SCM shall perform an encrypted communication with the Smart Fridge
Identity Req.	**IDNT01** - The user shall be authenticated
Identity Req.	**IDNT01.1** - The user shall be authenticated by user name and password
Safety Req.	**SFT01** - The SCM shall be able to check its temperature level.
Safety Req.	**SFT01.1** - The SCM temperature level shall be lower than 250°C

For the sake of simplicity and space motivations, we consider only these few requirements but in a project, it is possible to have hundreds or thousands of requirements.

Analysing the requirements presented in Table II, we can confirm that there is always at least one actor, an action and a goal as explained in section III. Each of these requirements is stored in a requirement database for each domain table.

As stated before, we have traceability between SEC01 and IDNT01. This type of connection is always present if the goal is the same for different requirements or if a requirement is a specialization of another one (Figure 10 and Figure 11).

D. Step 4

According to the requirement composition shown in Table I we obtain Table III, Table IV and Table V related to the final elicited requirements SEC01, IDNT01 and IDNT01.1.

As we can see, Inner Traceability (IT) is kept and there is a connection between the two requirements. We can also see that

TABLE III: Security requirement: SEC01

SEC01	The user shall be authenticated	na	na	IDNT01

TABLE IV: Identity requirement: IDNT01

IDNT01	The user shall be authenticated	na	IDNT01.1	SEC01

TABLE V: Identity sub-requirement: IDNT01.1

IDNT01.1	The user shall be authenticated by username and password	IDNT01	na	na

for requirement IDNT01, Forward Traceability (FT) points to requirement IDNT01.1. Symmetrically, Backward Traceability (BT) points from IDNT01.1 to IDNT01. With the term *na* we specify that there is no connection present.

These relationships are represented in Figure 12.

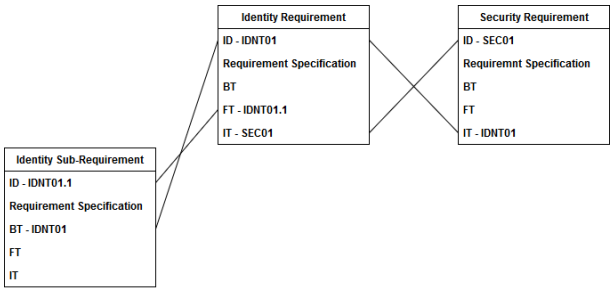

Fig. 12: Traceability between requirements

As we can see, traceability is maintained between the identity requirement and the security requirement. This relationship is bidirectional because if it is true that traceability exists between the ID of IDNT01 and the IT field of SEC01, it is also true that there is traceability between the ID of SEC01 and the IT field of IDNT01. We can see that the same happens regarding IDNT01 and IDNT01.1 (saved in the DB table "Identity sub-requirements" shown in Figure 12).

E. From Step 4 to Step 1

Now, we assume that the stakeholders decide to change the authentication process passing from a user and password authentication to a code authentication. Because the needs are changed, so the requirements must be changed too. Thus, the developers must delete the requirement (IDNT01.1) and add another one (IDNT01.2). It is better adding a new requirement deleting the old one and do not modify the old requirement. In fact, each requirement ID must be related to only one requirement elicited. So, in the case of a deletion, we have to create a new ID for the new requirement. Through this procedure, we implement traceability in the requirements database connecting requirements among them. In our case, as shown in Figure 12, there are connections between the identity requirement and the identity sub-requirement and between the identity requirement and the security requirement.

If we try to add the new requirement and delete the old one without releasing the connections, an error is raised. In fact, the developer cannot accidentally delete the relationship due

to the connection between IDNT01 and IDNT01.1. The traceability feature allows the developers to check the connection and, only after releasing it, it is possible to proceed with the change of the requirement. This feature guarantees to avoid domino effects in case of relaxing requirements.

In Figure 13, we propose the JSON related to IDNT01.2. We can state that it is similar to the one related to IDNT01.2 but the scope (row 8) and the goal (row 15) are related to the code authentication instead of user and password authentication.

*row*8 – **"Scope" : "Code Authentication"**
*row*15 – **"Goal" : "To authenticate the user by code"**

```
1  {
2  "IoT_requirement_IDNT01.2" : {
3    "Context" : {
4      "Domain" : [{
5        "Identity" : {
6          "Characteristic" : ["Authentication"] } }] ,
7      "Environment" : "Smart Home" ,
8      "Scope" : "Code Authentication" } ,
9    " Actor " : {
10     "Role" : "User",
11     "Type" : [ "Human User"] } ,
12   "Action" : {
13     "Type" : ["Fulfil"] ,
14     "Measure" : [" "]},
15   "Goal" : "To authenticate the user by code"
16   }
17 }
```

Fig. 13: JSON code for IDNT01.2

Finally, we can see the new final elicited requirement IDNT01.2 in Table VI and the modified connection regarding IDNT01 in Table VII.

TABLE VI: Identity sub-requirement: IDNT01.2

IDNT01.2	The user shall be authenticated by code	IDNT01	na	na

TABLE VII: Identity requirement: IDNT01

IDNT01	The user shall be authenticated	na	IDNT01.2	SEC01

Considering a more complex scenario, traceability is a powerful feature that permits to take into consideration and manage all the changes that a modification could produce to the database.

V. Discussion

This method guarantees in-depth research and focuses on the requirement elicitation process. On one hand, it is important to say that this method requires to spend a considerable amount of time during the requirement elicitation process. On the other hand, the benefits of doing an in-depth requirement elicitation process are large in terms of saved money and increased quality of the project [40]. The benefits given by a deep analysis of the security requirements are shown in [41]. In our work, we consider, in addition to security, other requirements connected to trust. This holistic view of the requirements is needed to perform a complete requirement elicitation process.

Moreover, the proposal of a step-wise systematic methodology helps to mitigate subjective issues eliciting requirements. In fact, one known issue of the requirement methodologies is that even using the same method, the subjectivity of the expert eliciting requirement is still present [42]. Our method, even if it does not solve completely this problem, minimize it giving to the developer a systematic step-by-step methodology. Moreover, the JSON implementation could be the basis for future directions related to the automation of the requirement elicitation process, as proposed by Mavropoulos et al. [20].

The Scenario that we have proposed has some limitations related to his complexity and on the number of requirements proposed, but it is useful to understand how our methodology is used. According to this simple scenario, the lessons learned are that the methodology helps to elicit requirements according to the stakeholder needs, avoiding mistakes related to the skip of this phase and traceability helps to avoid domino effects when some requirement must be changed. In addition, JSON will be helpful also for later phases of the K-Model (i.e. the verification phase) giving the possibility to automatize the verification. Moreover, the JSON template will be the basis to automatize also the process of requirement elicitation to mitigate the subjectivity issue.

VI. Conclusion and Future Work

In this paper, we have proposed TrUStAPIS, a JSON based requirement elicitation method that helps the developer to elicit the proper requirements during the SDLC of an IoT entity. Considering the K-Model we have proposed in our previous work, this methodology is used in the requirements elicitation phase. The requirements considered are related to seven domains: Trust, Usability, Security, Availability, Privacy, Identity, Safety. To write the proper requirements, we have highlighted for each of these domains a set of characteristics that must be taken into consideration. An important feature of the method is *traceability*. To have a holistic view of the developed IoT entity, it is fundamental to connect the requirements between them. This connection guarantees control and avoids domino effects in case of relaxing requirements. Finally, to show how the method works, we have presented a scenario related to a Smart Cake Machine. Thus, we have shown how to elicit the requirements and how the traceability works.

As a future work, we will expand the work by using TrUStAPIS considering a real and more complex use case scenario to demonstrate the validity and usefulness of the proposed method. In addition, we are working on a decision-making algorithm based on Analytic Hierarchy Process (AHP) [43] to classify the possible conflicts among requirements, in order to decide which requirement could be released or modified. This approach will permit developers and stakeholders to decide which requirement to keep during the requirements elicitation process. Furthermore, we will develop a tool to elicit

and store requirements according to TrUStAPIS method using our JSON template. In this way, the requirement elicitation process will be automatized mitigating the subjectivity issue discussed before. Then, we will present a survey analyzing our work compared to other requirement elicitation existing methodologies.

ACKNOWLEDGEMENT

This work has received funding from the NeCS project by the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 675320, the CyberSec4Europe project under SU-ICT-03 programme grant agreement 830929, and the SMOG project founded by the Spanish Ministry of Economy and Competitiveness (TIN2016-79095-C2-1-R).

This work reflects only the authors' view and the Research Executive Agency is not responsible for any use that may be made of the information it contains.

COMPLIANCE WITH ETHICAL STANDARDS

All authors declare that they have no conflict of interest. This article does not contain any studies with human participants or animals performed by any of the authors.

REFERENCES

[1] R. Roman, P. Najera, J. Lopez, Securing the internet of things, Computer **44** (9), 51-58 (2011).

[2] C. Fernandez-Gago, F. Moyano, J. Lopez, Modelling trust dynamics in the internet of things, Information Sciences **396**, 72-82 (2017) doi:https://doi.org/10.1016/j.ins.2017.02.039

[3] C. Haskins, K. Forsberg, M. Krueger, D. Walden, D. Hamelin, Systems engineering handbook, INCOSE (2006).

[4] D. Mellado, C. Blanco, L. E. Sanchez, E. Fernandez-Medina, A systematic review of security requirements engineering, Computer Standards & Interfaces **32** (4), 153-165 (2010).

[5] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, J. Mylopoulos, Tropos: An agent-oriented software development methodology, Autonomous Agents and Multi-Agent Systems **8** (3), 203-236 (2004).

[6] F. Massacci, J. Mylopoulos, N. Zannone, Security requirements engineering: the si* modeling language and the secure tropos methodology, Advances in Intelligent Information Systems, Springer, 147-174 (2010).

[7] H. Mouratidis, P. Giorgini, Secure tropos: a security-oriented extension of the tropos methodology, International Journal of Software Engineering and Knowledge Engineering **17** (02), 285-309 (2007).

[8] E. S.-K. Yu, Modelling strategic relationships for process reengineering, Ph.D. thesis, University of Toronto (1995).

[9] E. Paja, F. Dalpiaz, P. Giorgini, Modelling and reasoning about security requirements in socio-technical systems, Data & Knowledge Engineering **98**, 123-143 (2015).

[10] L. J. Hoffman, K. Lawson-Jenkins, J. Blum, Trust beyond security: an expanded trust model, Communications of the ACM **49** (7), 94-101 (2006).

[11] M. Pavlidis, Designing for trust, CAiSE (Doctoral Consortium), 3-14 (2011).

[12] R. Rios, C. Fernandez-Gago, J. Lopez, Modelling privacy-aware trust negotiations, Computers & Security (2017).

[13] D. Ferraris, C. Fernandez-Gago, J. Lopez, A trust by design framework for the internet of things, NTMS'2018 - Security Track (NTMS 2018 Security Track), Paris, France (2018).

[14] Z. Yan, P. Zhang, A. V. Vasilakos, A survey on trust management for internet of things, Journal of network and computer applications **42**, 120-134 (2014).

[15] A. Jøsang, R. Ismail, C. Boyd, A survey of trust and reputation systems for online service provision, Decision support systems **43** (2), 618-644 (2007).

[16] D. H. McKnight, N. L. Chervany, The meanings of trust, Technical Report MISRC Working Paper Series 96-04 (1996).

[17] R. Baharuddin, D. Singh, R. Razali, Usability dimensions for mobile applications: a review, Res. J. Appl. Sci. Eng. Technol **5** (6), 2225-2231 (2013).

[18] P. Mahalle, S. Babar, N. R. Prasad, R. Prasad, Identity management framework towards internet of things (iot): Roadmap and key challenges, International Conference on Network Security and Applications, Springer, 430-439 (2010).

[19] R. Rios, C. Fernandez-Gago, J. Lopez, Privacy-aware trust negotiation, International Workshop on Security and Trust Management, Springer, 98-105 (2016).

[20] O. Mavropoulos, H. Mouratidis, A. Fish, E. Panaousis, C. Kalloniatis, Apparatus: Reasoning about security requirements in the internet of things, International Conference on Advanced Information Systems Engineering, Springer, 219-230 (2016).

[21] IEEE Computer Society. Software Engineering Standards Committee, IEEE-SA Standards Board. IEEE Recommended Practice for Software Requirements Specifications. Institute of Electrical and Electronics Engineers. (1998).

[22] A. Alonso-Nogueira, H. Estevez-Fernandez, I. Garcia, Jrem: An approach for formalising models in the requirements phase with json and nosql databases, World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering **11** (3), 353-358 (2017).

[23] W. Abdelghani, C. A. Zayani, I. Amous, F. Sedes, Trust management in social internet of things: a survey, Conference on e-Business, e-Services and e-Society, Springer, 430-441 (2016).

[24] T. Beth, M. Borcherding, B. Klein, Valuation of trust in open networks, European Symposium on Research in Computer Security, Springer, 1-18 (1994).

[25] J. Chang, H. Wang, Y. Gang, A dynamic trust metric for p2p systems, 2006 Fifth International Conference on Grid and Cooperative Computing Workshops, IEEE, 117-120 (2006).

[26] B. Christianson, W. S. Harbison, Why isn't trust transitive?, in: International workshop on security protocols, Springer, 171-176 (1996).

[27] T. Grandison, M. Sloman, A survey of trust in internet applications, IEEE Communications Surveys & Tutorials **3** (4), 2-16 (2000).

[28] S. P. Marsh, Formalising trust as a computational concept, Ph.D. thesis, Department of Computing Science and Mathematics, University of Stirling (1994).

[29] M. Nitti, R. Girau, L. Atzori, Trustworthiness management in the social internet of things, IEEE Transactions on knowledge and data engineering **26** (5), 1253-1266 (2014).

[30] Z. Yan, S. Holtmanns, Trust modeling and management: from social trust to digital trust, IGI Global, 290-323 (2008).

[31] R. Mahmoud, T. Yousuf, F. Aloul, I. Zualkernan. Internet of things (IoT) security: Current status, challenges and prospective measures. In 2015 10th International Conference for Internet Technology and Secured Transactions (ICITST) (pp. 336-341) (2015).

[32] M. U. Farooq, M. Waseem, A. Khairi, S. Mazhar. A critical analysis on the security concerns of internet of things (IoT). International Journal of Computer Applications, 111(7) (2015).

[33] M. Bauer, M. Boussard, N. Bui, J. De Loof, C. Magerkurth, S. Meissner, J.W. Walewski. IoT reference architecture. In Enabling Things to Talk (pp. 163-211). Springer, Berlin, Heidelberg (2013).

[34] A. Pfitzmann, M. Hansen. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management (2010).

[35] K. Ligett, S. Neel, A. Roth, B. Waggoner, S. Z. Wu. Accuracy first: Selecting a differential privacy level for accuracy constrained erm. In Advances in Neural Information Processing Systems (pp. 2566-2576) (2017).

[36] M. Lesk. Safety risks-human error or mechanical failure?: Lessons from railways, IEEE Security & Privacy **13** (2), 99-102 (2015).

[37] S. Singh, N. Singh. Internet of Things (IoT): Security challenges, business opportunities & reference architecture for E-commerce. In 2015 International Conference on Green Computing and Internet of Things (ICGCIoT) (pp. 1577-1581) (2015).

[38] Q. Gou, L. Yan, Y. Liu, Y. Li. Construction and strategies in IoT security system. In 2013 IEEE international conference on green computing and communications and IEEE internet of things and IEEE cyber, physical and social computing (pp. 1129-1132) (2013).
Internet of Things. In 2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC) (pp. 1-6) (2019).

[40] S. Friedenthal, A. Moore, R. Steiner: A practical guide to SysML: the systems modeling language. Morgan Kaufmann (2014).

[41] R. L. Kissel, K. M. Stine, M. A. Scholl, H. Rossman,J. Fahlsing, J. Gulick. Security considerations in the system development life cycle (No. Special Publication (NIST SP)-800-64 Rev 2) (2008).

[42] M. Geisser, T. Hildenbrand. A method for collaborative requirements elicitation and decision-supported requirements analysis. In IFIP World Computer Congress, TC 2 (pp. 108-122). Springer, Boston, MA (2006).

[43] T. L. Saaty. Analytic hierarchy process. Encyclopedia of Biostatistics, 1 (2005).