Human-centric Computing and Information Sciences

RESEARCH Open Access

A model-driven approach to ensure trust in the IoT

Davide Ferraris*, Carmen Fernandez-Gago and Javier Lopez

*Correspondence: ferraris@lcc.uma.es
Network, Information and Computer Security Lab, University of Malaga, 29071 Malaga, Spain

Abstract

The Internet of Things (IoT) is a paradigm that permits smart entities to be interconnected anywhere and anyhow. IoT opens new opportunities but also rises new issues. In this dynamic environment, trust is useful to mitigate these issues. In fact, it is important that the smart entities could know and trust the other smart entities in order to collaborate with them. So far, there is a lack of research when considering trust through the whole System Development Life Cycle (SDLC) of a smart IoT entity. In this paper, we suggest a new approach that considers trust not only at the end of the SDLC but also at the start of it. More precisely, we explore the modeling phase proposing a model-driven approach extending UML and SysML considering trust and its related domains, such as security and privacy. We propose stereotypes for each diagram in order to give developers a way to represent trust elements in an effective way. Moreover, we propose two new diagrams that are very important for the IoT: a traceability diagram and a context diagram. This model-driven approach will help developers to model the smart IoT entities according to the requirements elicited in the previous phases of the SDLC. These models will be a fundamental input for the following and final phases of the SDLC.

Keywords: Trust, SysML, UML, Internet of Things (IoT), System Development Life Cycle (SDLC)

Introduction

The Internet of Things (IoT) is a paradigm that permits smart entities (i.e., smart things and humans) to be interconnected anywhere and anyhow [1]. Gartner forecasted that 20.4 billions of devices will be connected by 2020 and, in the same year, the IoT spending will reach almost 3 trillion US$.[1] These numbers show how IoT can grow in the next future. This will bring new opportunities but also new issues. These issues will be related to security and privacy, but also to trust.

Trust is difficult to define. It concerns different aspects and topics ranging from Philosophy to Computer Science [2] and it is strongly dependent on the context, in fact trust "means many things to many people" [3].

However, in a trust relationship, there are basically two actors involved: the trustor and the trustee. The trustor is the one who actively trusts and the trustee is the one who keeps the trust. We can state that this collaboration is necessary when the trustor needs

[1] https://www.gartner.com/newsroom/id/3598917.

the trustee to perform an action or fulfill a goal considering a particular context. This goal is not achievable by the trustor alone. For this reason, the trustee is needed. Trust metrics are necessary to compute a trust level that helps the trustor to decide if a trustee can be trusted [4]. This value must be computed before the two actors start the collaboration. Moreover, the trust level could change over time positively or negatively due to the good or bad behaviour of the trustee [5].

According to Hoffman et al. [6] and Pavlidis [7], trust can be connected to other properties like security and privacy. Then, Ferraris et al. [8] stated that these relations are even more important during an IoT entity development. In fact, as also stated by Mohammadi et al. [9], trust mechanisms can be fundamentals and requires more investigation in this field. For this reason, in our opinion, it is crucial to consider trust since the initial phases of the System Development Life Cycle (SDLC) in order to develop correctly the trust relationships among smart entities. Indeed, if we consider trust through the whole SDLC, this approach could give important rules of behaviour during the interactions with other smart entities. These rules, if implemented during the development of the smart entities, can improve the protection of the same entities and of whom is using them. For example, an important field relating to trust and IoT is related to health monitoring services [10] where the relationships among trusted entities are crucial in order to preserve human lives. Another important field that can be both connected to trust and IoT is related to blockchain technologies [11, 12] in order to protect the IoT systems against attacks.

However, during the SDLC of any system, UML [13] and SysML [14] are widely used by developers. In fact, these diagrams have been created in order to explore the different functionalities of a generic software/system under development. Anyhow, these original modeling languages had no features to implement security, privacy or trust. For this reason, it is necessary to define them in order to help the developers to properly model trust and related domains.

The purpose of this paper is to present a model-driven approach extending UML and SysML in order to implement trust in the SDLC of an IoT entity. Moreover, this language is enriched by trust and related domains. These domains have been presented by Ferraris et al. [8] where the authors proposed a framework that helps developers to consider trust during the SDLC of an IoT entity. We define them as domains because they cover different "areas of knowledge or activity" as defined by Oxford dictionary.[2] Moreover, the word *domain* represents better the fact that they are connected but separated by their different purposes. This framework is composed of a K-Model and transversal activities (i.e., Decision Making, Traceability). The first phase of the K-Model consists of the needs phase, where it is defined the purpose of the IoT entity to be developed. Then, the second phase is related to the requirements phase where the developer elicits the requirements according to the previous needs. The third phase consists of the modeling phase, which is the one that we will extend in this paper.

The structure of the paper is as follows. In "Related work and background" section, we explain the related work and we present our K-Model as background work for this paper.

[2] https://www.oxfordlearnersdictionaries.com/definition/english/domain.

Then, our model-driven approach is presented in "Trust model-driven approach" section. In addition, in "Scenario and diagrams" section, we present the diagrams and a use case scenario to show how the model-driven approach can be implemented. Then, we propose a database simulation in "Trust modeling simulation" section in order to show how the diagrams are connected by traceability. Finally, in "Conclusion and future work" section, we conclude the paper and discuss the future work.

Related work and background

This section is divided into two sub-sections. The first one contains the related work about trust, IoT and the existing system modeling languages that we consider in the development of our model-driven approach. Then, the second sub-section is related to the background of this paper, where we present the K-Model. Our model-driven approach specifies the third phase of the K-Model.

Related work

In the state of the art, the first work that considers trust in the information technology (IT) field has been proposed by Marsh [15] in the nineties. Then, 2 years later, Blaze [16] proposed trust management as a way to merge authentication and access control into a single trust decision. After them, many authors defined other trust models. Anyhow, considering the modeling languages field, several works expand it adding extra features concerning security, but only a few authors have proposed a way to consider and compute trust in UML.

One of these works has been proposed by Uddin et al. [17]. The authors extend UML to UMLTrust considering trust in several UML diagrams (i.e., class diagram). However, this work is not intended for the IoT, so it is needed to add new diagrams (i.e., context diagram) to be able to develop a smart IoT entity. Moreover, they specify only three diagrams that are not enough to model all the relevant aspects related to trust in a software. In fact, without models such as sequence diagram or activity diagram is harder to represent specifically the actions that must be performed by the entity under development. We fill this gap as we will present in "Trust model-driven approach" section.

Before them, only Gorski et al. [18] implemented stereotypes to consider trust in UML use cases. They considered evidence as claims to influence the trust level of the trustor, but they did not implement other trust characteristics. In Ref. [19], we have presented a list of such characteristics (i.e., direct, asymmetric, measurable) and we think that they must be taken into consideration both in the requirements and modeling phase of the SDLC.

Considering security, Jürjens [20] implements security policy validation and encryption extending UML in UMLsec. Furthermore, Basin et al. [21] and Lodderstedt et al. [22] extend UML in secureUML in order to develop access control rules into the models. A limit of these works is that they have not considered scenarios where the access control rules are violated and they have not considered deeply the requirements phase focusing only on the design phase. As we will present in the next sub-section, our work follows the requirements phase that is fundamental in order to develop an IoT entity.

About security and privacy, Mai et al. [23] proposed a modeling method extending only the use case diagrams belonging to UML. Their main motivation was to enrich these diagrams with security and privacy adding a limited number of extensions. An important aspect of this work was the modeling of threat scenarios and their possible mitigation processed in a “structured form”. The main limitation of this work is that extends only the use case diagram, that is the most general UML diagram. For this reason it is difficult to consider this work for complex environments (such as the IoT). In fact, Aufner [24] stated that even if threat models and their implementation to increase security have been deeply researched in social networks [25], there is a lack of research in the IoT field.

Then, other UML extensions consider risk and threats adding features to implement them during the modeling phase. Concerning risk analysis, Vraalsen et al. [26] use CORAS [27] to implement threat and risk modeling. In relation to threats, Hussein et al. [28] extend UML in UMLintr to include intrusion detection into the models. This aspect is very important for the IoT [29]. Anyhow, these works did not consider any other domains and did not considered different contexts.

Regarding SysML, a few works have extended it considering security properties and no one have considered trust. Maskani et al. [30] extended SysML especially in relation to requirement diagrams. The purpose of this work is to add security stereotypes to consider security also during the requirements elicitation process. However, their work does not consider other important properties such as privacy or trust and it is related only to the requirements phase. Apvrille et al. [31] developed SysML-Sec, a framework for embedded systems. They extended SysML to cover requirements, design and validation phases. Regarding the security requirements, they elicited it considering threats and risk assessment. Nevertheless, they do not consider traceability to connect the needs to the requirements and they do not consider other properties.

So far, we noticed also that none of these works have been intended to be used specifically for IoT. Anyhow, Harrand et al. [32] proposed ThingML, a modeling language designed in order to support the development code generation. This modeling language compared to UML can be considered as a domain specific modeling language (DSML). Nonetheless, this work does not consider security, privacy and trust domains. Moreover, Mavropoulos et al. [33] proposed a conceptual model to support the developers during decision making tasks about security analysis of IoT entities. This is an important task that as proposed by the authors must be extended considering also privacy aspects. Anyhow, trust is not considered.

Thus, even if there is at least two works concerning this topic, we believe that in the state of the art there is a lack of modeling languages for IoT and that there is still little effort to consider trust during the modeling phase of the SDLC. Our work aims to fill this gap merging the trust models domain with modeling languages such as UML and SysML. To achieve this goal, we have to consider also trust modeling and how trust has been computed and considered in the state of the art.

For this reason, in this paper, we include the features identified by Moyano et al. [34] to enrich our model-driven approach with trust. Basically, the authors stated that we can distinguish between two types of trust models: evaluation and decision models. Concerning the decision models, the important feature are the credentials, policies and evidences. In addition, it is a particularity of these models to provide a step-by-step authentication that preserves the entities’ privacy. In fact, policies and credentials

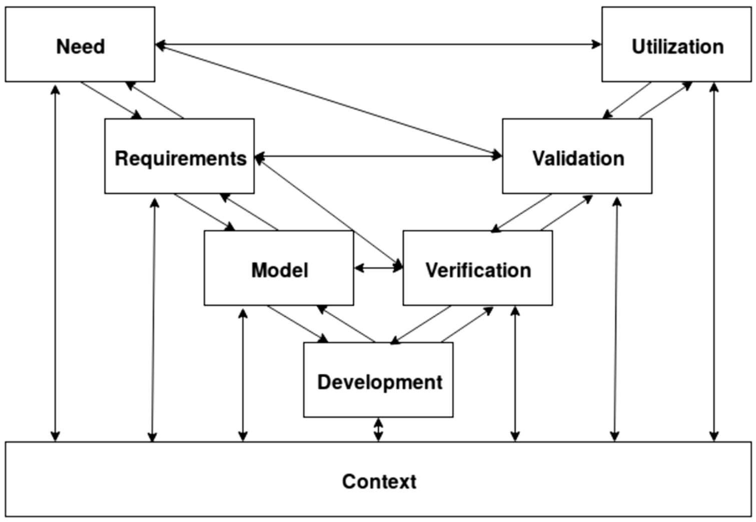


Fig. 1 K-Model [8]

are revealed only when they are required, avoiding the disclosure of extra information when it is not needed. This is an important aspect to take into consideration in order to develop authentication processes for the IoT field [35]. Finally, regarding the evaluation models, Moyano identified that a trust level is always present and it could be uni-dimensional or multi-dimensional. According to Jøsang [36] it might have different degrees of objectivity or scope. To compute these values, trust metrics are necessary and they need attributes to be computed through engines (i.e., simple summation, bayesian).

Background

In our previous work, we have presented a framework that holistically considers trust during the SDLC of an IoT entity [8]. It is composed of a K-Model and transversal activities (i.e., Traceability, Risk Analysis). Moreover, because of the dynamicity and heterogeneity of the IoT, context is crucial and strictly related to all the phases of the SDLC. In Fig. 1, we show the K-Model and its phases covering the SDLC of an IoT entity: from cradle to grave. The first phase is about the need phase, where the purpose of the new IoT entity is presented and discussed among all the stakeholders. They are all the actors having an interest in the new IoT entity that is going to be developed. In this phase, it is important to understand where the new IoT entity will be used by the final customers and to know which kind of architecture will be considered in its working environment [37–39].

The second phase is related to the requirements elicitation process where developers must elicit every requirement according to the needs produced in the previous phase. About this phase, we have presented a paper [19] analyzing the requirements elicitation process proposing a requirements elicitation method to help developers in this important and difficult task. In fact, trust management has always been a hard task to be effectively performed during the requirements elicitation process [40].

The third phase is the one proposed in this paper, where we present a model-driven approach for the SDLC of an IoT entity. In fact, this paper is fundamental in order to explain and cover this important phase of the SDLC.

As we show in Fig. 1, the model phase receives inputs from the previous phases and produces output for the central phase of the K-Model (i.e., Development) where the developers will build the IoT entity. Moreover, the models produced in this phase will be verified in the Verification phase. In addition, as we will see in the following sections, another important parameter to be taken into consideration is the context. Especially for trust, context is crucial. In fact trust is different for each individual depending on different situations [41]. According to this aspect, we propose a new diagram: the context diagram.

Finally, as we mentioned before, in the K-Model there are seven transversal activities. In this paper, we especially take into consideration one of them creating a new diagram: the traceability diagram.

Both the traceability and context diagrams will be explained and presented in "Scenario and diagrams" section.

Trust model-driven approach

Our model-driven approach has been developed to ensure trust and related domains into IoT, filling the gap identified in "Related work" section. Important aspects during the modeling task of an IoT entity are to consider its dynamicity and heterogeneity. This can be performed by extending basic diagrams related to UML and SysML and proposing new ones. For this reason, we can state that UML has been developed for software and SysML for general systems. IoT is a system containing software and can be modeled partially either using UML or SysML. Our aim is to create a model-driven approach that can be effective for an IoT entity and, for this purpose, we merge SysML and UML extending their diagrams and creating new ones.

The extended diagrams are use case diagram, class diagram, activity diagram, sequence diagram, state machine diagram and requirement diagram. We have chosen these six diagrams because they allow developers to implement the fundamental aspects of an IoT entity. In fact, the use case diagram allows developers to model the general interactions of the IoT entities; the requirement diagram permits developers to consider the requirements elicited in the previous phase of the K-model in order to model and connect them to the other diagrams; then, the class diagram will be fundamental in order to develop the software of the IoT entity. Finally, the activity, sequence and state machine diagrams allow developers to specify the general interactions under three different perspectives. Moreover, it is important to say that even if the diagrams are different and they explore distinct aspects of the modeled IoT entities, they can be connected. For example, an activity can represent a particular use case. Then, the same activity can be specified through a sequence diagram or a state machine diagram. We will show examples covering this particularity across the paper.

Furthermore, we need two new diagrams: the traceability and the context diagram.

On one hand, the traceability diagram will be used by developers to trace the diagrams and the connections among them. It can be considered a meta-diagram. On the other hand, the context diagram will be used to map the different contexts that will be considered for the IoT entity. This is a diagram useful for the development of all the contexts related to an IoT entity.

We can state that especially the context diagram will improve the effectiveness of our model-driven approach for an IoT purposes. In fact, differently from other systems, IoT entities due to the heterogeneity of its environment can be part of multiple contexts. These contexts can be completely different among them as we will see in "Context diagram" section. For this reason, we can state that due to the specificity of the IoT environment this new diagram allows our model-driven approach to be IoT specific. Anyhow, our model-driven approach can also be used for systems different from the IoT but it can be less effective.

For each diagram that we present in "Scenario and diagrams" section, we briefly describe the basic features (assuming that they are well known by the reader). We will focus more on the improvements respect to the original UML and SysML diagrams.

An important aspect defining the diagrams is the consideration of trust and its related domains. The domains related to trust are usability, security, availability, privacy, identity and safety. Regarding trust, we use features related to the evaluation and decision models proposed by Moyano et al. [34], as we have explained at the end of "Related work" section.

These domains must be considered during the IoT entity modeling. According to the domain and to the selected diagram, the developers must represent the requirements elicited in the previous phase of the K-Model using model-driven approach. Generally, developers must examine the domain related to the modeled requirement considering both its characteristics and the actors involved in that particular case as presented in [19]. In "Scenario and diagrams" section, we will represent for each diagram a general case and a use case scenario in order to show how these tasks must be performed.

Methodology

Now, we present a step-by-step methodology in order to show how the model-driven approach must be considered according to the K-model and the steps that must be performed by the developers in order to perform the model phase.

The steps are shown in Fig. 2.

1. The first step contains the collection of data related to the previous phases of the K-Model (i.e., Need and Requirements). This information is fundamental in order to create the proper diagrams and model the entity according to requirements and needs.
2. Secondly, the diagrams can be drawn by the developers following the model-driven approach and considering the context of the IoT entity. We stated in [19] that the context is strongly dependent on the environment and the scope of the IoT entity. About the diagrams, that will be presented in "Scenario and diagrams" section, they can be drawn in any order, but the preferred order is to consider firstly the requirement diagram according to the elicited requirements of the first step. Then, it is better to draw the use case diagrams collecting all the general actions that must be performed by the actors and the IoT entity. The context must be considered for the whole step, thus the context diagram can be drawn at the beginning and concluded

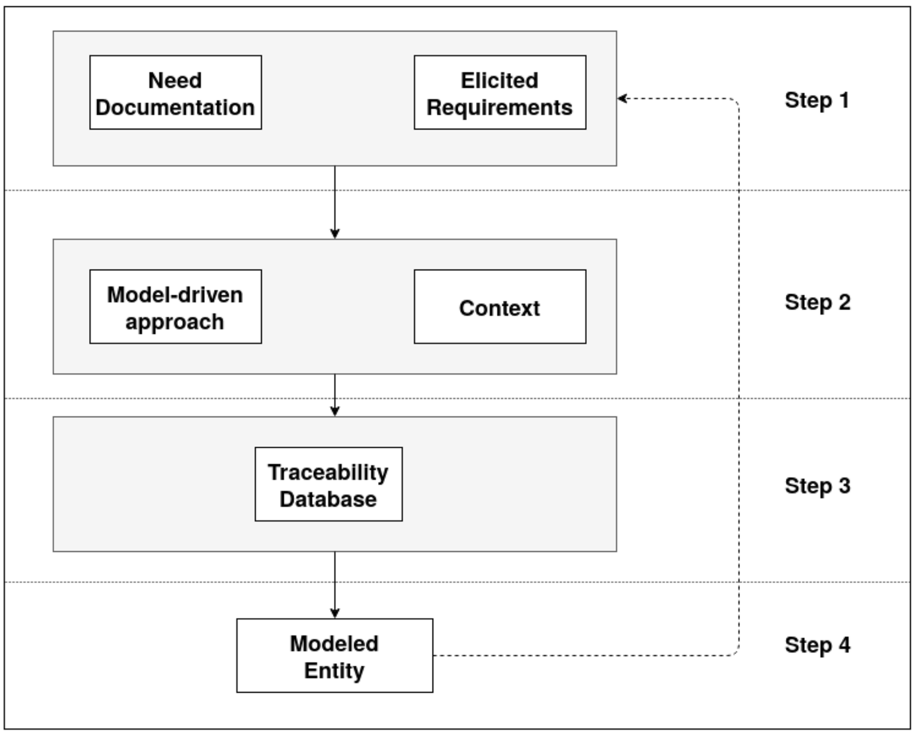


Fig. 2 Model-driven approach: step-by-step methodology

at the end of this step, considering dynamically the modeling process. Then, the developers will use the class, activity, sequence and state machine diagrams in order to specify the general actions modeled by the use case diagram. As for the context, also the traceability diagram can be drawn during the development of the other diagrams in order to map their connections from the beginning, but traceability will be considered in the following step.

3. The third step is fully related to traceability. Considering the traceability diagram created in the previous phase and all the important data related to the other diagrams, it is possible to create a traceability database. It will be fundamental in order to avoid domino effects in the case a diagram must be modified or deleted. This step will be explained in "Trust modeling simulation" section.
4. The fourth is the final step of the methodology where the modeled entity is delivered and it will be developed in the following phase of the K-Model. Anyhow, in the case some modification is needed (for example, because the developers did not address a requirement or a risk was partially or not covered), there is the possibility to come back to step 1 in order to perform these modifications. Certainly, the developers need to consider traceability in order to make these modifications.

This systematic methodology helps the developers to follow a guideline in order to draw the models in a proper way. In the next section, we provide the description of the diagrams and a use case scenario to show how they can be implemented.

Scenario and diagrams

To show a practical example , we introduce a use case scenario that will help us to show how the diagrams work and how it is possible to implement their functionalities.

The first step, as explained in the previous section, is related to the need documentation and the elicited requirements.

In this example, we consider that the stakeholders need to develop an IoT entity that can be used through voice commands (i.e., a device similar to Amazon Alexa Echo Dot[3]). From now on we will use the term device and entity for the same purpose. This device is mainly used in a smart home environment. This means that it must interact with the different actors of a smart home such as humans and other IoT devices (i.e., a smart thermostat). For security and privacy reasons, the device can store personal data of each user. Moreover, through this IoT entity, it is possible to order goods, check the calendar or the bank account and play music. Moreover, this IoT device can interact with other IoT entities belonging to the same smart home. However, all these functionalities must be separated for trust and security reasons, also because it is possible that the device is used by different users that may be able or not to access sensitive information. To cover completely the first step, we show the elicited requirements directly through the requirements diagram in "Requirement diagram—RD1" section.

Then, as suggested in "Methodology" section, for step 2 we will proceed firstly with the requirement diagram, then the use case diagram to show a general action and then the other diagrams in order to specify some functionalities of the device. At the end, we will show a context diagram and a traceability diagram. In order to guarantee traceability through the diagrams and the other phases of the K-Model, each diagram has a unique ID that allows developers to refer to them uniquely. Due to space limitation, we use each diagram proposed, but only once, in order to model different aspects of the IoT entity. From this moment, we refer to the IoT entity as Tvoice. Each diagram will cover different aspects of Tvoice utilization. These models will help the developer in the following phases of the SDLC in order to implement the right functionalities as intended by the previous phases of the SDLC. We will show the different cases in each section in order to give an example of how the diagrams can be used.

In "Trust modeling simulation" section, we will show the third step of the methodology explaining this functionality and why it is important.

Requirement diagram

The Requirement Diagram (RD) helps to map all the requirements being elicited in the previous phase and, through the traceability diagram presented in "Traceability diagram" section, it is possible to map the connections among this and the other diagrams. The use of traceability avoids domino effects in the case of changing or relaxing requirements.

RD is originally used in SysML, where a requirement specifies a capability that shall be satisfied by the system under development. It is expressed by two elements: an ID and a text describing the requirement. In addition, there are operators used to define

[3] https://www.alexa.com/.

the relationships among requirements and other phases of the SDLC (i.e., verification). These operators are the following ones: contain, derive, satisfy and refine.

We extend the RD implementing stereotypes for each domain with the following representation <<*Domain Requirement*>> (i.e., for trust it will be <<*Trust Requirement*>>). Each stereotype identifies a set of requirements. Through this distinction, it is easier for the developer to recognize the domain belonging to each requirement.

It is important to underline that according to [19], a requirement could have one or more sub-requirements. Usually, the sub-requirements specify additional information that must not be included in the main requirement.

For this reason, we propose additional and optional information. An optional information is used by the developers if they need to better specify the description of a requirement. The new information is related to the stakeholders and to the IoT entity or its subsystems and it is extended by using the following optional elements:

- Verify: It is a link showing which kind of parameter must be verified during the verification phase.
- ExpressedBy: With this element, we consider the stakeholder that expressed the need satisfied by the requirement. Usually, the stakeholders are the ones that have an interest in the entity under development (i.e., vendors, customers).
- ExpressedFor: This element is related to the IoT entity as a whole or as a part of it (i.e., a subsystem) expressed by the requirement.
- NeedSatisfied: This element represents which need is satisfied by the requirement.
- ModelConnected: This element is related to which diagrams are used to fulfill the requirement. It is a type of traceability. It could be used together with *Satisfy*.
- RiskCovered: It represents the risk mitigated by the requirement.
- ThreatMitigation: This element represents which known threat is mitigated by the elicited requirement.

In "Requirement diagram—RD1" section, we propose an example to show how requirements must be written and connected among them.

A graphical view could be complicated in case of a huge number of requirements. For this reason, we use the database proposed in [19] that also enhances traceability among requirements.

Requirement diagram—RD1

In this example, we show how a Requirement Diagram is drawn and how it can be used to represent and connect different diagrams.

This RD is connected to UCD1 and CD1, in fact, in Fig. 3 we can see requirements connected to the privacy action represented in UCD1 (see "Use case diagram example—UCD1" section) and voice interaction analysed in CD1 (see "Class diagram—CD1" section), as we will see later.

RD1 is composed of three main requirements. The Ids and the texts of the requirements are shown in Fig. 3. We assume that the trust requirement (id: TRST-01) satisfies a need expressed by the vendor. Moreover, this requirement mitigates the risk of unauthorized use. The Security Requirement (id: SEC-01) derives from TRST-01 and

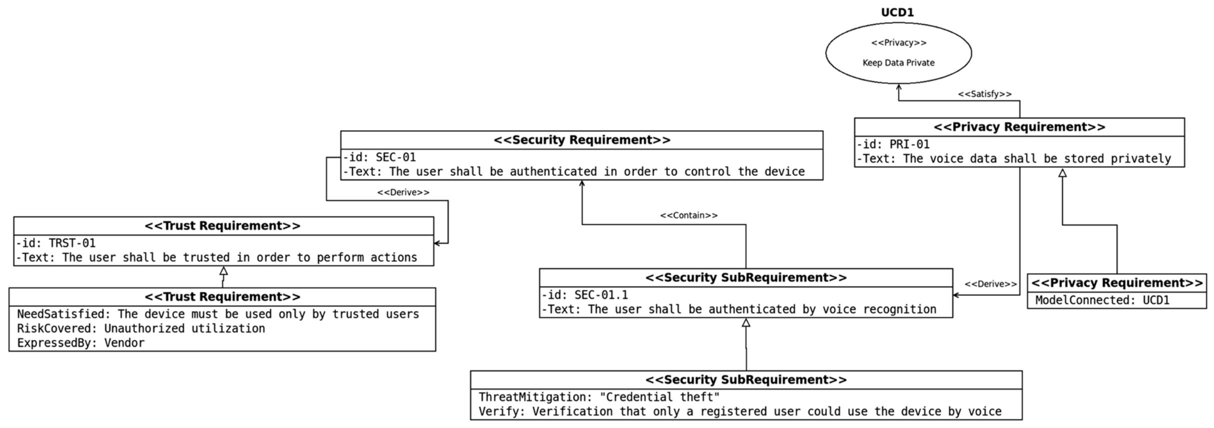

UCD1
<<Privacy>>
Keep Data Private
<<Satisfy>>
<<Privacy Requirement>>
-id: PRI-01
-Text: The voice data shall be stored privately
<<Security Requirement>>
-id: SEC-01
-Text: The user shall be authenticated in order to control the device
<<Derive>>
<<Trust Requirement>>
-id: TRST-01
-Text: The user shall be trusted in order to perform actions
<<Trust Requirement>>
NeedSatisfied: The device must be used only by trusted users
RiskCovered: Unauthorized utilization
ExpressedBy: Vendor
<<Contain>>
<<Security SubRequirement>>
-id: SEC-01.1
-Text: The user shall be authenticated by voice recognition
<<Derive>>
<<Privacy Requirement>>
ModelConnected: UCD1
<<Security SubRequirement>>
ThreatMitigation: "Credential theft"
Verify: Verification that only a registered user could use the device by voice

Fig. 3 RD1—trust, security and privacy requirements

is related to the fact that a user shall be authenticated in order to be trusted (decision trust). SEC-01 has a sub-requirement (id: SEC-01.1) that specializes the authentication process through voice authentication. The sub-requirement must be verified in the verification phase. Furthermore, it mitigates the threat related to the "Credential Theft", because only the voice of a legitimate user can be accepted in order to perform actions. An attack to break this control could be performed registering the voice of the legitimate user, but this is a hard task to achieve without having access to the same room of the user. To enhance the protection, the privacy requirement (id: PRI-01) deriving from SEC-01.1 states that the voice data shall be stored privately. This requirement is connected to the UCD1 element "Keep Data Private" through a satisfy connection. This connection is represented also by the element ModelConnected.

Use case diagram

The UCD is an existing UML and SysML diagram representing a user interaction with a system or an entity. This interaction is represented through actions. In its original form, the diagram is represented by actors and circles (or ellipses) containing the actions. These actions can be general or specific. Other types of diagrams may expand this diagram in order to show how an action is fulfilled (i.e., a sequence diagram specifies the actions).

Actors are represented by name and they are tagged specifying which role they have in the following format:<<***tag***>>. For example, possible tags for the actors involved in a trust action could be: <<*Trustor*>> or <<*Trustee*>>.

As an extension, we include this feature also into the Use Cases. Thus, we can represent them including a tag related to the domain and their name. The name of the use case represents an action. In addition, in our version of UCD, the same actor may be involved in different domains (i.e., trust and privacy). In this case, there are "parallel" use cases that are connected among them. To model this aspect, we use a set of connections related to the actions. These connections are shown with the following tags: <<Dependence>> when the line has a direction or <<Interdependence>> when the line is not oriented. For example, if there is a <<Dependence>> pointing from action A to action B, it means that action B is generated because of action A. In case of an <<Interdependence>>, it means that both actions have the same importance.

Use case diagram example—UCD1

Tvoice provides various functionalities. One of them, that we consider very important from a trust, privacy and security perspective, is the possibility to store user's private data in order to access them when needed.

Using our UCD, we model the possibility that Tvoice stores the data of the users. In this case, the device should be allowed to store locally the private data of the user. In order to proceed, the user must approve the action, otherwise the data will be asked when needed.

For this use case diagram, we need to take into consideration three domains (privacy, security and trust) that might affect the privacy of the data, the security of the storage and the trust of the user providing his/her personal data.

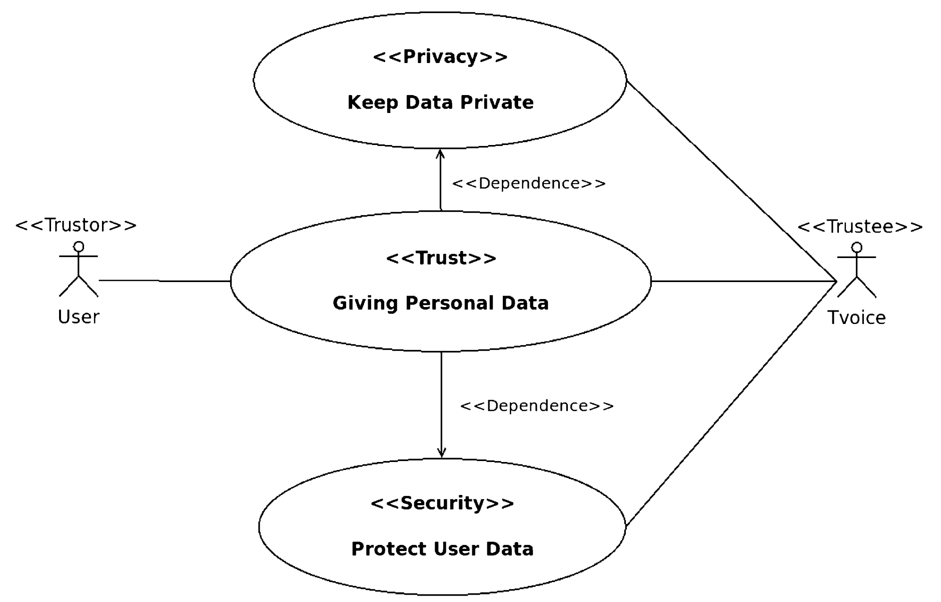


Fig. 4 UCD1—user data

Figure 4 shows the use case for our example. As we can see, there are two actors involved. One is the user (i.e., the owner of the device) who is considered, from a trust perspective, as the trustor. The other one is Tvoice, that in this case is considered as the trustee. We model three use cases that are related to three different domains: privacy, trust and security. The trust use case is named "Giving Personal Data". It is connected through a <<Dependence>> connection to the privacy use case named "Keep Data Private" and to the security use case named "Protect User Data". In fact, we can state that to share private data, the user must trust the device. This trust relationship is strongly dependent on the fact that the data are kept private and secure (i.e., an encrypted and protected database). In addition, we can see that the user is connected to the trust use case. We assume that the user gives the data as a trust action. He is not connected directly to the privacy or security use case, but only through the trust use case. The motivation is that the privacy and security actions are only provided by Tvoice and for the user, they are important but transparent. On the other hand, Tvoice is involved in all the use cases. As a trustee in order to keep the trust of the trustor. Moreover, Tvoice needs to store the data privately and securely.

To conclude, the user can be everyone (i.e., the owner of the device or a malicious user). Using this diagram, it is more important to model *what* the system shall do more than *how* the system implements the functionalities. In fact, the use case is a general diagram and the implementation of the rules must be implemented through other diagrams (such as an activity diagram or a sequence diagram).

Class diagram

IoT entities are usually developed with a software that controls their behaviour according to the context and the environment where they will be placed. For this reason, we propose an extension of the class diagram (CD), which is widely used in UML and

Software Engineering. Through the CD, the developers can manage careful planning of the software itself and define the entities and the actions that they can perform.

In our CD version, we reuse three canonical boxes originally developed in [13] representing the name of the class, the attributes and the methods (or operations) plus an extra box related to the context. It can be empty, in the case a class is used in every context or it can contain the classification of the contexts related to the context diagrams (see "Context diagram" section). Because the contexts could be more than one, we use an array to represent this field.

We extend the name box adding the domain that is considered for the class. We implement an array because a class could belong to different domains. Thus, the name of the classes is represented in the following way:

$$<< class\_name[Domain(1), ..., Domain(n)] >>$$

The boxes related to attributes and methods are designed according to the domain of the class. For example, in the case that the domain is related to trust, it will be fundamental to specify attributes and methods belonging to the evaluation or decision models specified by Moyano et al. [34]. In fact, depending on which models are considered, the methods will be used to implement a software containing decision or evaluation rules.

Class diagram—CD1

As we defined earlier, Tvoice must allow the legitimate and trusted users to order any goods through an E-commerce service. Anyhow, in this case, we assume that this action can be performed only by the owner of the device. In order to recognize the owner, the device must register the owner data (i.e., birth date, voice, credit card) and keep them private. Other recognized users can be guests and children. Anyway, they cannot perform actions according to this particular case. Finally, the available services must be trusted by the owner and the device in order to proceed with the transactions.

We summarize all these concepts in the class diagram (CD1) shown in Fig. 5.

From the "context" field it is possible to see that some classes are related to the context number 5. This context is shown later in "Context Diagram—XD1" section). The classes related to it are **Order** and **Service**. The other three classes are important for every context, so the context box is empty. At the center, we have the **User** class. We can see from the stereotypes that this class is related to the trust, privacy and security domains. These domains are chosen because the user is a *trustor* of the services. In fact, he/she has a role that concerns security constraints and the recording of the voice can raise a privacy issue.

Now, we focus only on two classes in order to show which trust models we have considered for them.

We focus on the following parameter belonging to the class **Order**: **trustedRole**. The only trusted user role is the owner. The other users cannot perform orders (at least in this scenario). For this reason, in this class trust is a *boolean* value.

In this case, to model trust, we consider decision model rules and also in this case the considered parameter is related to the user's role.

Then, considering the class **Service**, we focus on the **Trust** attribute. It represents the trust level of the service. In this case, trust is represented by a *float* value because each

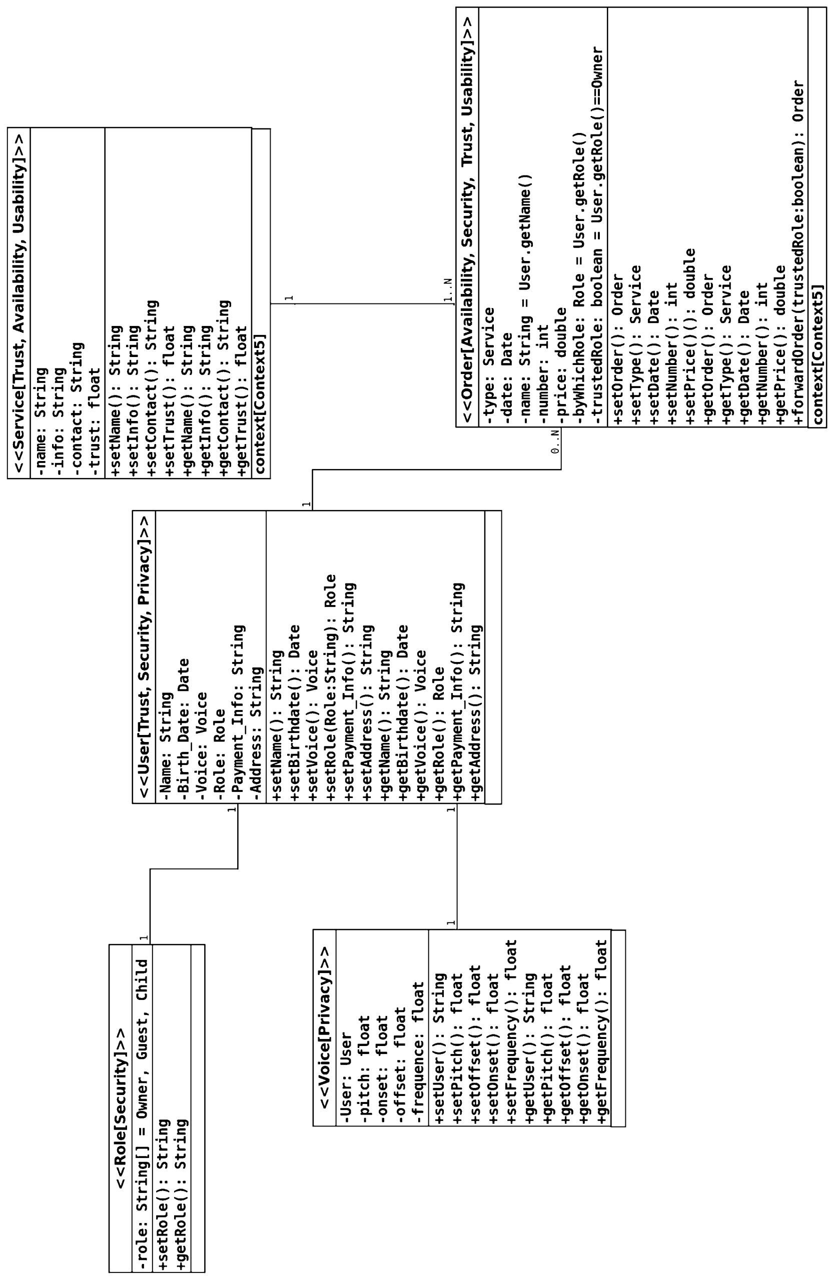


Fig. 5 CD1—E-commerce service classes

service could have different trust levels in order to be trusted. Anyway the value is considered between 0 and 1 where the former refers to no trust and the latter to the maximum trust level.

In this case, we model trust following evaluation model rules. In fact, trust is dependent on the different trust levels of the service organized as reputation values. We decided to consider them as floats in this example, but they can be considered also as double or integer parameters. The chosen metric can create different trust levels. We decide to consider them with the following ranges.

If x (i.e., the trust value) is lower than 0.5 the service is not trusted. On the other hand, if x is higher or equal to 0.5 the service is trusted. The ranges are between 0 and 1. Anyhow, possibilities and ranges could be numerous, but we decide to consider this simple case where the service is simply not trusted or trusted. After the outcome of the order, the user can change the value using the setTrust() method.

The connection between the class User and Voice is 1 to 1 because each user has a unique pitch and voice [42]. Then, the connection between User and Role is 1 to 1 because we assume that for every context the role does not change. It should be possible to have a different role for each possible context, but we do not model this case in this scenario. The connection between the User and the Order is 1 to 0/N because a user could perform zero orders or more. Finally, the connection between the Service and the Order is 1 to 1/N because a service is used for at least one to more orders.

Activity diagram

The generic Activity Diagram (AD) is basically an advanced flow chart. It models the workflow from an activity to another. An IoT environment can be represented by the AD in order to specify which activities are important to be developed. An AD is composed of activities, a pre-condition and one or more post-conditions. There are also optional elements: decision points, merge nodes, fork, join and swimlanes.

Regarding the swimlanes, we enrich them to model the domains, rather than the boundaries among actors. So, for example, a swimlane related to privacy means that every activity belonging to it will be related to privacy concepts. The connection between swimlanes belonging to different domains could represent the connection between UCD actions that we presented in "Use case diagram" section.

Considering the workflow, we implement new elements enriched by trust. We propose a new element called "*trust trigger*". It is used to pass from an activity to another considering a minimal trust level as a trigger. If the trust level is enough, then the workflow continues. Otherwise, the workflow could end with a post-condition or it could continue with an alternative workflow. It is also possible to have a flow to increase the trust level (i.e., giving more information). After that, the flow may return to the trust trigger to check the new trust level. The criteria to continue the flow after the trust trigger could be related to trust decision model aspects (it is basically an access control decision) or to trust evaluation model aspects (it considers a computation of trust parameters like reputation into a single trust level). It is possible to use a decision point after the trust trigger. Considering the computed level of trust, the decision point considers this value as an input to allow the workflow to proceed to a determined path. For example, it is possible to continue with a subset of activities in a secondary path (i.e., a less trusted path in

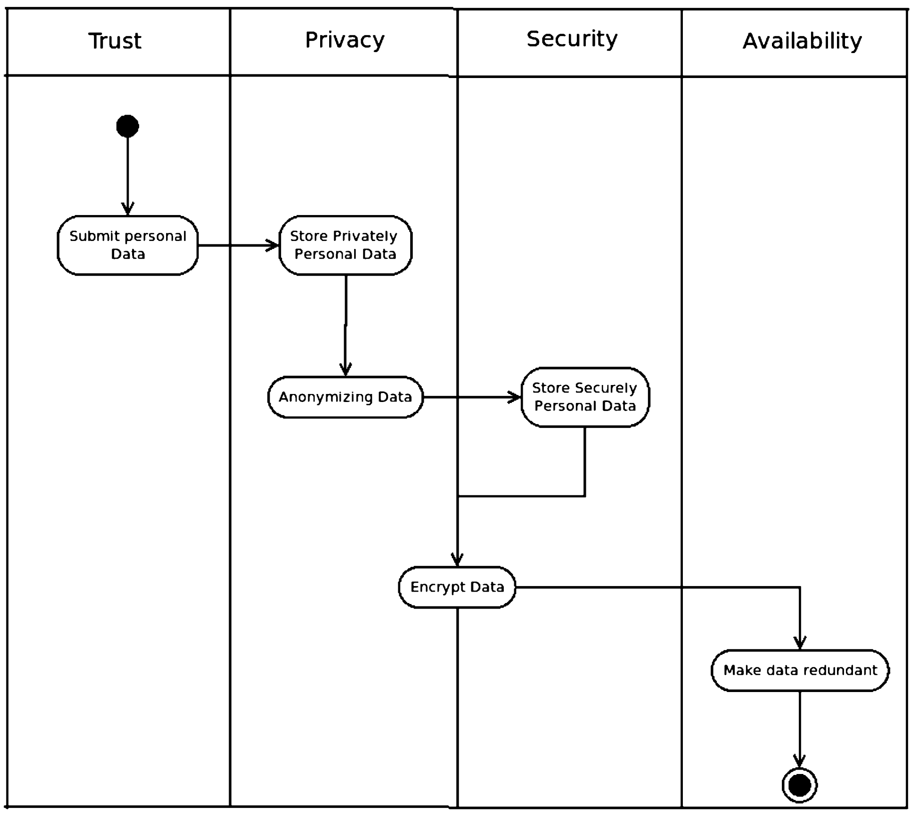


Fig. 6 AD1—securing user data

case of a low trust level). Regarding the trust trigger, we will show an example in "State machine diagram" section.

Activity diagram—AD1

With this example, we model an AD to implement the possibility for the users to store their personal data to Tvoice.

In Fig. 6, we show how an AD related to UCD1 could be implemented. Specifically, we implement the possibility to secure user data. As we have shown in Fig. 4, there are at least three domains involved: trust, security and privacy. These activities are modeled to specify the use case actions. In addition to these domains, in this diagram, we consider also the availability domain.

We use the swimlanes to separate the activities belonging to different domains. About the trust domain, we model the activity "Giving personal Data". In fact, only if the user trusts Tvoice, it is possible to perform this activity. It belongs to the trust domain because the user must trust the device in order to reveal personal data. Then, the following activity is part of the privacy domain and it is called "Store privately personal Data". This means that Tvoice must store these data considering privacy aspects. The following activity is "Anonymizing Data" to enhance the privacy of the data. Hence, there is an activity belonging to the security domain: "Store securely personal Data". This is a generic activity and the developer will decide how to store the data securely. The following activity is related to the encryption of the data. It belongs to the privacy and security domain because through encryption it is possible to enhance the security of the data (i.e., avoiding unauthorized use of it) and also the privacy of the data. Finally, the last activity belongs to the availability domain and it requires that the data could be made redundant. Through the redundancy, the possibilities to lose the data are minimized. There is no written rule on how this redundancy will be implemented. Through this diagram it is more important to model the *what* rather than the *how*.

Sequence diagram

The sequence diagram (SD) illustrates the interactions among actors related to the execution of a particular task or process. The first purpose of an SD is to model all the steps performed during a particular interaction. The SD could specify also a particular UCD action or an AD activity. For example, it is possible to have an AD activity called "Check the Smart Thermostat Temperature" and several steps to perform this activity shown in the SD (i.e., steps to connect to the thermostat and a step to ask information about the temperature level). Basically, the SD represents the messages exchanged by the actors or by the systems involved in the interaction through arrows. Moreover, there can be frames representing the type of interaction.

In our extension of the SD, we include features to consider trust. Moreover, in a trust domain, it is important to model the difference between decision and evaluation models. We add labeled frames related to these models to distinguish between trust decision or evaluation models [34]. These frames are useful to implement the steps related to a particular trust model using the right parameters and metrics. Thus, if the frame is related to an evaluation model, the sequence diagram will be drawn considering messages about the computation of a reputation or a trust value. On the other hand, if the frame is belonging to a decision model, the messages will be used to implement the process of giving credentials in order to be trusted. Furthermore, we add frames concerning other domains. Thus, we can model actions related to other diagrams and belonging to a particular domain (i.e., security). Considering the previous phases of the SDLC, this separation in the same sequence diagram helps developers in modeling the proper requirements according to the development process taking into consideration the connections among domains.

Sequence diagram—SD1

Tvoice is built to offer many services to the user through voice interaction. One of them is the IoT interaction with the smart thermostat belonging to the same smart home.

Thus, using the SD, we model the interaction between the user and Tvoice referring to the check and modification of the smart thermostat temperature.

In the diagram that is shown in Fig. 7, the interaction is started by the user asking Tvoice to check the temperature. To proceed, Tvoice checks if the voice matches the owner's registered voice. In this interaction, there is a privacy issue to be taken into consideration related to the voice parameters of the owner. Moreover, there is a decision trust computation, because the voice match can be considered as an authentication process. In this diagram, we model that the user voice is matched, so Tvoice asks the smart thermostat for the temperature. The smart thermostat replies with the temperature.

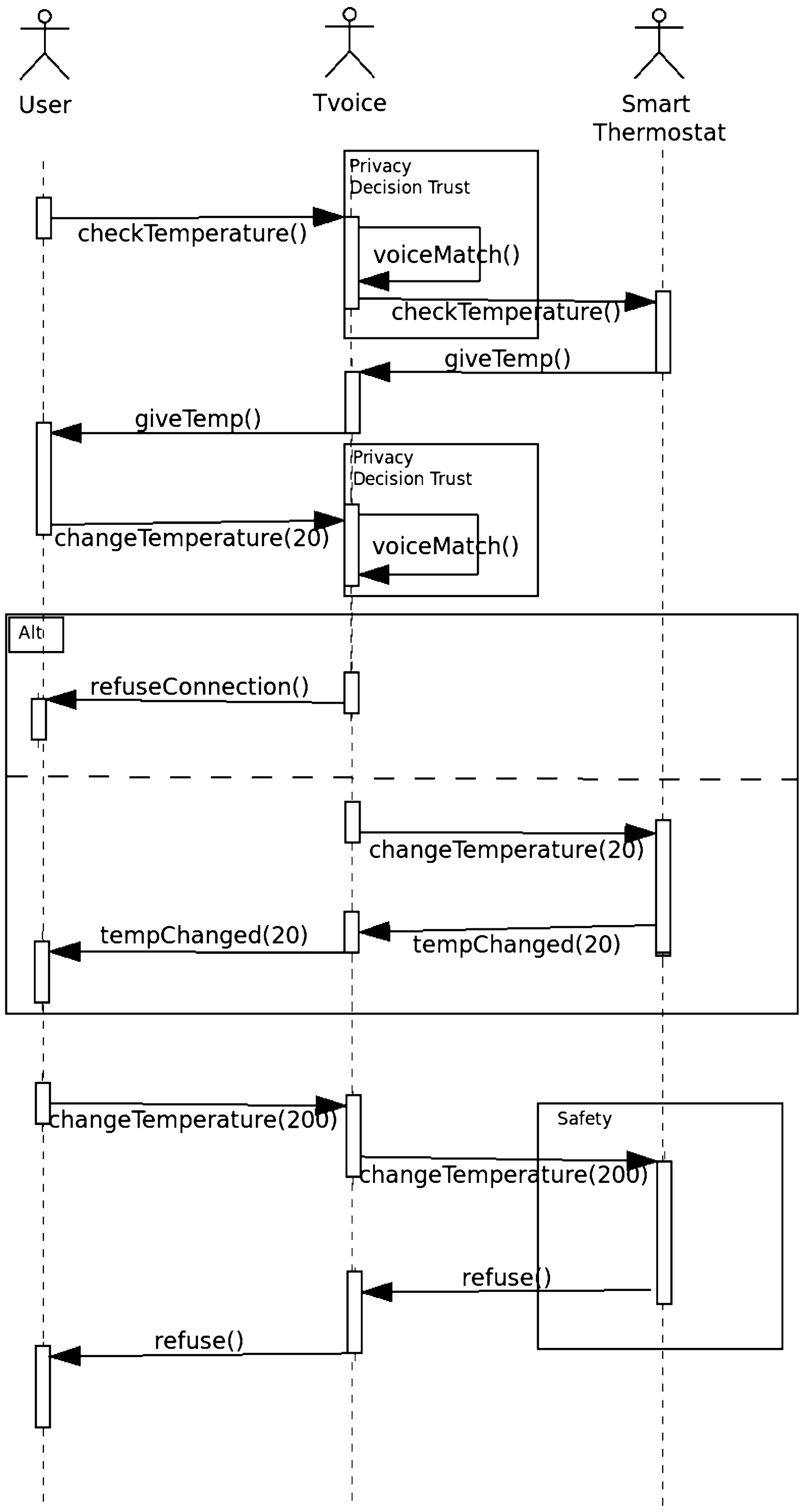


Fig. 7 SD1—interaction between Tvoice and the Smart Thermostat

We assume that Tvoice is trusted by the smart thermostat. In the interaction between Tvoice and the thermostat, Tvoice is the trustor and the smart thermostat the trustee. The interaction ends with Tvoice giving the temperature information back to the user.

After this step, we model another possible interaction. In fact, this time the user wants to change the temperature. Tvoice checks the user voice and then we modeled the following steps as mutually exclusive (alt tag), in fact, only one of them can be executed. The first alternative is related to the failure of the request because the user is not allowed to change the temperature. The second alternative considers the success of the request. In this case, Tvoice interacts with the Smart Thermostat in order to change the temperature to 20 °C. The thermostat replies with the command *tempChanged(20)* and Tvoice informs the user about the change.

Finally, we model another situation. Now, the user wants to change the temperature to 200 °C. This time, we can have a safety domain issue and the request is refused by the smart thermostat. In this interaction, we did not model the *voiceMatch()* check in order to avoid redundancy.

State machine diagram

The state machine diagram (SMD) is used basically to represent every possible state that is reached by the modeled entity. The original SMD is mandatory composed of a starting and a final state. In the middle of the execution there could be also one or more transition states.

A final state could be reached whether the process fails or ends correctly. Moreover, a state could be simple or composite. In the simple case, the transition to the next state is automatic. In the composite state, there are some functions or activities that must be performed inside the state.

In our extended version, we design the SMD to help developers to consider each state that the smart IoT entity could reach. During the iteration, we consider a flow representing the exact state of the entity moving from the starting state to a final state.

Moreover, as for our AD version, we consider using trust triggers to pass to the next state. Thus, in the case that the trust level is not enough to continue the iteration, a final state is reached. In addition, after the trust trigger, it is possible to have an alternative path according to the triggered trust level or useful to increase the trust level.

It is important to underline that these triggers enrich the diagram with more control. In fact, without these triggers, the SMD could end directly or proceed to the next state. Therefore, using these triggers, we increase the modeling possibilities of this diagram. Moreover, we enrich these diagrams with this element in order to avoid the use of multiple elements to represent this one. For this reason, with these new elements, we have increased the effectiveness of the modeling representation in order to immediately understand the element and its purpose.

In the case the trust trigger is developed following decision model rules, it allows the flow to proceed only if the right credentials are provided. Otherwise, it is possible to have a subset of states that are developed in order to allow the collection of more information (i.e., in the case a password is needed, in the case the password is not provided it is possible to create a new one or to provide other information in order to be accepted). On the other hand, if the trust trigger is implemented using evaluation model rules, the

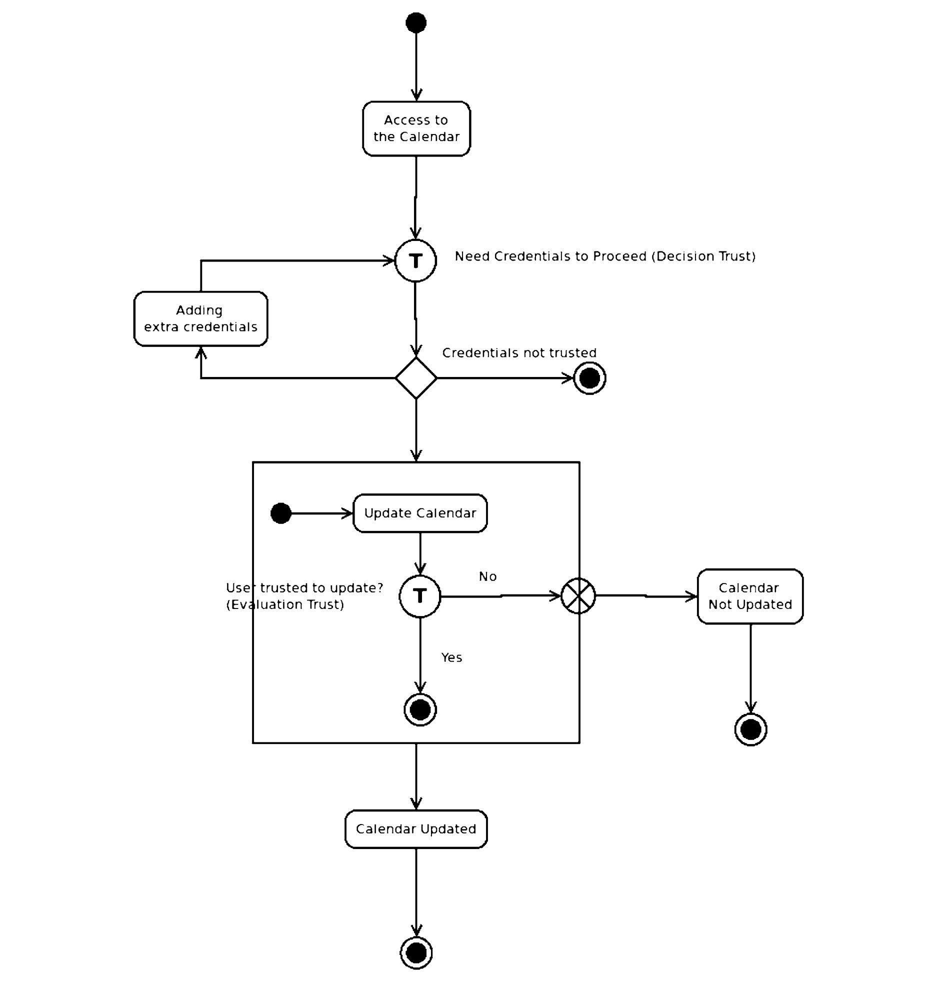


Fig. 8 SMD1—update the user calendar

flow will be allowed to continue only if the trust level is higher than the trust trigger level. If not, there are two possibilities: the flow ends or the flow continues to a sub-state in order to increase the trust level.

State machine diagram—SMD1

One of the potentialities of an IoT entity, moreover if it is a voice assistant, is to be used for many different contexts. Through this diagram, we model another important service of a home assistant: update the user calendar.

Through the SMD that is shown in Fig. 8, we model the states related to the calendar update.

Firstly, there is a state related to access to the calendar. To pass to the following state (that is a composite state) a trust trigger is needed. In fact, without the proper credentials, it is not possible to access the calendar. Here, we have two possibilities: the state machine ends or it is possible to provide other credentials in order to gain trust to access the calendar. We do not model how and which credentials are needed, we model the possibilities to have an extra step to access the calendar (i.e., using a

Table 1 Context diagram—database template

Context_ID	Context_Name	(Domain_1, Domain_2,…, Domain_N)

password, a secret question). If the extra credentials are provided or the first credentials are enough, the state machine passes to the next state, which is a composite state. A composite state has another starting point and it is like a state machine inside a single state. In this composite state, Tvoice checks that the user is enabled to update the calendar. If not, the calendar could not be updated, the state machine exits from the composite state, it enters in the state "Calendar not updated" and then it ends. If the user is enabled to update the calendar, the state machine exits from the composite state and it enters in the state "Calendar updated", then it ends.

Context diagram

As presented in "Background" section, according to the dynamicity and heterogeneity of the IoT environment it is important to consider context in every phase of the K-Model. Especially in the model phase, it is important to provide developers with a diagram to implement it.

For this reason, in our model-driven approach, we create a new diagram called context diagram (XD) to model the context related to the developed IoT entity. By this diagram, the developers could map the dynamic behaviour of the IoT entities considering all the contexts related to them. Anyhow, the developer can choose which context to model in each diagram according to its task.

The XD shows the different contexts that an IoT entity could be part of. In addition, it helps in understanding the connections and weak points raised as the consequence of sharing different contexts within the same IoT entity. For example, if an IoT entity is used to check both the calendar and the bank account, it is important to guarantee the separation of these two contexts. This separation mitigates the risk of sharing information between the different contexts. In fact, it is possible that an entity could be involved in the calendar context but not in the bank context.

Each context could belong to a particular domain (i.e., trust, privacy). For each of them, the developer must consider the actors involved and the type of information that is shared. Furthermore, it should be possible that more domains are involved in a particular context. In this case, we can model the context using the characteristics related to the domains. These characteristics were analyzed in [19] and they are useful to determine how the context is composed and how the IoT entity should be developed.

It is possible that the XD related to the IoT entity could have more contexts and its graphical representation can be difficult. For this reason, it is possible to represent the contexts using a database notation. The database table is composed of three columns. The first one refers to the context ID, it is unique and considered as the key element. The second one contains the context name. Then, the third one refers to the domains belonging to the context. We can see the database template in Table 1.

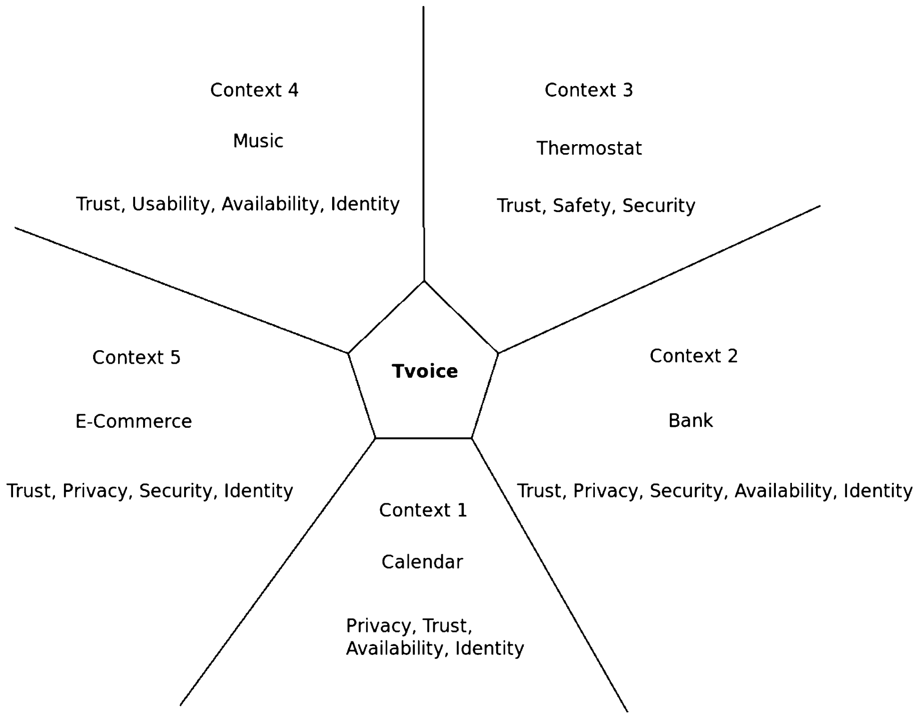


Fig. 9 XD1—contexts: calendar, bank, thermostat, Music, E-commerce

We show an example for this new diagram in the next sub-section.

Context diagram—XD1

In Fig. 9, we consider five contexts that can be implemented in Tvoice.

Some of them have been presented in the previous diagrams. The contexts are: calendar, bank, thermostat, music and E-Commerce. For each of these contexts, there are one or more domains related to it. The contexts are defined as follows:

- Context 1 (Calendar): In this case, the domains are related to privacy, trust, availability and identity. In fact, it is important to have the service available for the device. Moreover, the information must be kept private and available only for the identified trusted user.
- Context 2 (Bank): We select five domains. Trust is crucial to allow a user to check the bank account or to make transactions through Tvoice. Security is important to guarantee that the service is protected and not accessible to other users (malicious or not). Privacy is important to keep the information safe and not shared with other users. Availability is important because the service must be always available. Then identity is strongly related to trust, privacy and security because it enforces that the authenticated user is the only one that must manage bank information.
- Context 3 (Thermostat): For this context, we select the trust, safety and security domains. Because this context is related to a very important service for the security of the smart home, we select both safety and security. Safety strongly related

Table 2 XD1—database view

Context 1	Calendar	[Privacy, Trust, Availability, Identity]
Context 2	Bank	[Trust, Privacy, Security, Availability, Identity]
Context 3	Thermostat	[Trust, Safety, Security]
Context 4	Music	[Trust, Usability, Availability, Identity]
Context 5	E-commerce	[Trust, Privacy, Security, Identity]

to the environment and it is fundamental in order to avoid malfunctions that can harm the humans involved. Security is more related to the IoT. It is needed to protect the basic functionalities of the device by malicious users. Trust is important to allow the interaction between Tvoice and the smart thermostat.

- Context 4 (Music): For this context, Tvoice needs to interact with other external services (i.e., Youtube, Spotify). In this case availability of the external service is needed. Usability is fundamental to allow opportunity to the user to create playlist according to their preferences. Identity is important to provide the correct user with the right playlists. Trust is important because the user and the streaming service must be trusted.
- Context 5 (E-Commerce): This context is related to an external service that must be trusted to provide E-commerce services. In this context also privacy is important considering the personal data of the owner (i.e., credit card, address). Security is important to keep these data protected. Identity guarantees that only authorized users can perform actions.

As we said earlier, if there are many contexts it would be a problem to use a graphical view. So, we can see in Table 2 the database view related to XD1.

Traceability diagram

In order to cover traceability, that is an important transversal activity of the K-Model proposed in "Background" section, we create a second new diagram: the Traceability Diagram (TD). This diagram is important to trace connections among all the diagrams that we have presented so far. The traceability diagram can be considered as a diagram of diagrams (i.e., a meta-diagram). Moreover, we can state that when two or more diagrams are connected among them they compose a cluster.

The developers by using the TD have a holistic view of the models considering their connections during the development of the IoT entity. In addition, it is possible to mitigate domino effects in the case a modification or deletion of a diagram is needed. Because each diagram has a unique identifier, it is possible to map the connections among diagrams uniquely and store these data in a proper database. This means that this diagram could be graphical (showing the connections between diagrams) or it could be represented by a traceability database. In fact, as for the XD, in the case TD grows, a graphical notation can be complicated to be represented. For this reason, we can represent this diagram using also a database notation as we will see in "Trust modeling simulation" section.

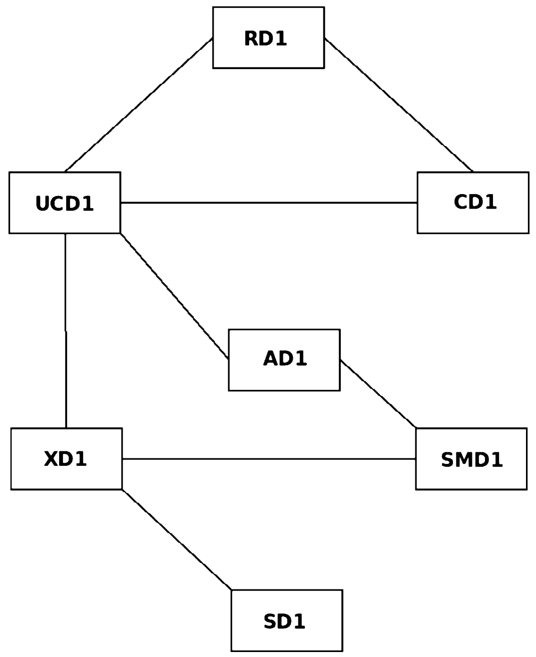


Fig. 10 TD1—traceability among diagrams

Traceability diagram—TD1

Considering our modeling scenario, the correspondent TD is presented in Fig. 10.

We can see that UCD1 is connected to four diagrams: RD1, CD1, AD1 and XD1. In fact, the use cases presented in UCD1 are modeled in different ways using the other diagrams. Then, it is possible to see a connection between CD1 and RD1 because the requirements specified in RD1 are developed in CD1. XD1 is also connected to SD1 because the smart thermostat context is considered in both of them. Finally, AD1 is connected to SMD1 because both the diagrams are related to store personal information. For this reason, SMD1 is connected also to XD1 where the calendar context is proposed as the context number 1.

Trust modeling simulation

In this section, we present the third step of the step-by-step methodology presented in "Methodology" section. Here, we show how the traceability database must be structured. We will consider the diagrams presented in the previous section plus other dummy diagrams in order to show how traceability works with a larger number of diagrams. In fact, the more complex the scenario is, the more elements will be connected among them.

Even if there are other possibilities in order to show how the diagrams are connected among them (i.e., visual goal diagrams), we have chosen the database visualization because we think that it is the most effective way to show how the diagrams are connected among them. Moreover, creating the database tables, we can represent also important aspects related to any diagram (i.e., the domains) or to a specific one (i.e., activities for the AD).

Each database table is related to a particular diagram (i.e., CD or SMD) where the primary key is the ID of the related diagram (i.e., CD1 for a CD diagram). The traceability diagram is represented by a table related to the connections among diagrams. In the rows of this table, we have all the IDs of the connected diagrams. If there is a connection among requirements they will be represented in the same row. In the relational databases is represented as a multi-multi relationship database.

In Fig. 11, we can see the database chart where we have a traceability database connected to all the databases of the other diagrams. In this scenario, we assume that there is no connection among diagrams related to the same type (i.e., UCD1 cannot be connected to UCD2).

As shown in Fig. 11, each diagram is stored in their table where the diagrams belonging to the same type are allocated. Each of them is connected by their ID to the Traceability table. In each row of the Traceability table, we have only the diagrams connected among them. For example, if we have a connection between two diagrams, we will have only two diagrams represented in a single row. On the other hand, if we have a connection among three or more diagrams, we will have a row containing three or more IDs. The missing diagrams are represented by a "na" that means "Not Available". For example, referring to the case proposed in Fig. 10, we can see that there is a triple connection among RD1, UCD1 and CD1. In fact, they are all connected among them, RD1 is connected with both CD1 and UCD1, UCD1 is connected with both CD1 and RD1 and, finally, CD1 is connected with RD1 and UCD1. One consideration is needed related to the diagrams UCD1, AD1, SMD1 and XD1. They are connected in circle among them, but there is no direct connection between AD1 and XD1 or SMD1 and UCD1. For this reason, they cannot be represented all together in a single row. In Fig. 12, we can see how the traceability table is populated according to the rules that we have mentioned earlier.

In Fig. 12, we have represented all the diagrams (AD1, RD1, CD1, XD1, SD1, SMD1 and UCD1) related to the use case proposed along "Trust model-driven approach" section. Now, we add other dummy diagrams in order to show how the traceability relationship works. These dummy diagrams are the following:

- Activity diagrams: from AD2 to AD9;
- Requirement diagrams: from RD2 to RD9;
- Class diagrams: CD2 to CD5;
- Context diagrams: XD2 to XD3;
- Sequence diagrams: SD2 to SD9;
- State machine diagrams: SMD2 to SMD8
- Use case diagrams: UCD2 to UCD8

We do not represent the tables related to the single diagrams for space limitations. Anyhow, their connections are represented in order to show how the traceability database works avoiding the loss of important information after deleting a connected diagram.

Some of the dummy diagrams are injected in the Traceability Table in order to simulate the connection among them. The new table is shown in Fig. 13.

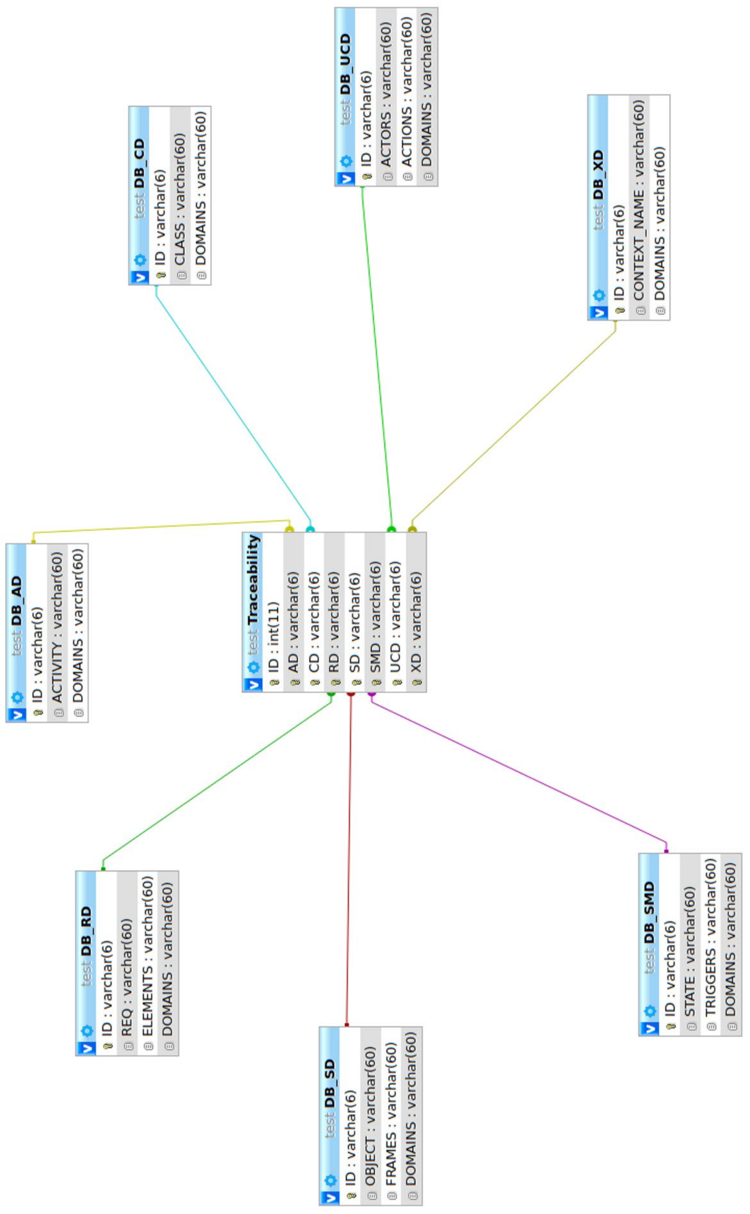
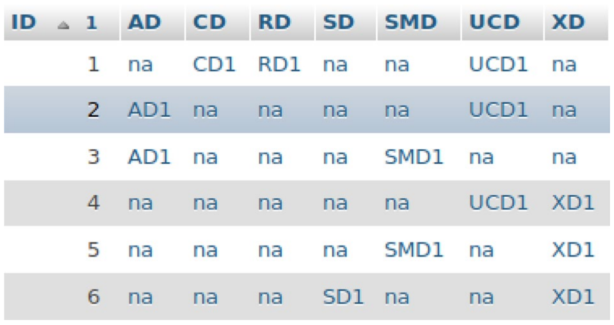
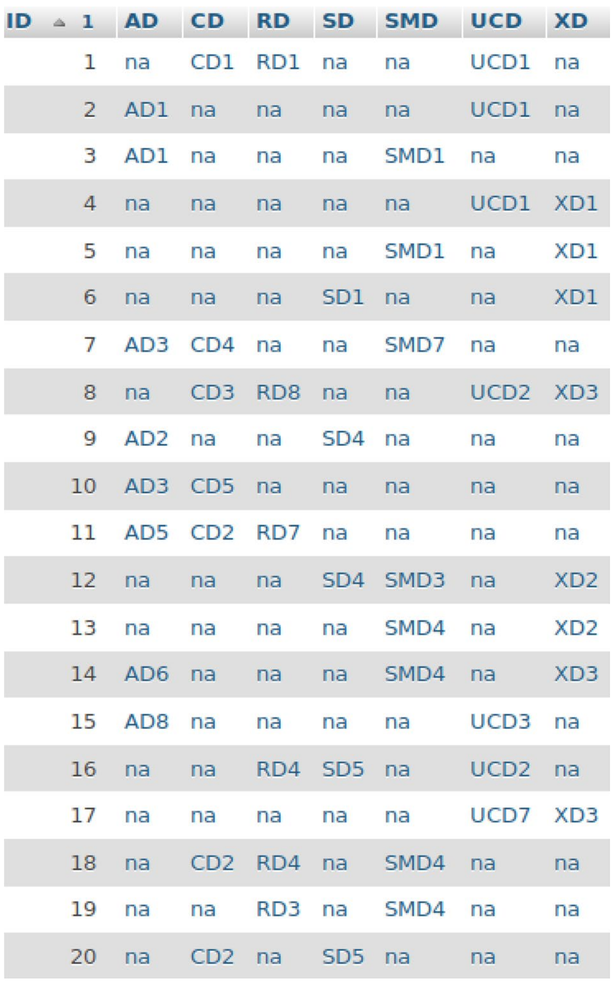


Fig. 11 Database chart



ID ▲ 1	AD	CD	RD	SD	SMD	UCD	XD
1	na	CD1	RD1	na	na	UCD1	na
2	AD1	na	na	na	na	UCD1	na
3	AD1	na	na	na	SMD1	na	na
4	na	na	na	na	na	UCD1	XD1
5	na	na	na	na	SMD1	na	XD1
6	na	na	na	SD1	na	na	XD1

Fig. 12 Traceability table



ID ▲ 1	AD	CD	RD	SD	SMD	UCD	XD
1	na	CD1	RD1	na	na	UCD1	na
2	AD1	na	na	na	na	UCD1	na
3	AD1	na	na	na	SMD1	na	na
4	na	na	na	na	na	UCD1	XD1
5	na	na	na	na	SMD1	na	XD1
6	na	na	na	SD1	na	na	XD1
7	AD3	CD4	na	na	SMD7	na	na
8	na	CD3	RD8	na	na	UCD2	XD3
9	AD2	na	na	SD4	na	na	na
10	AD3	CD5	na	na	na	na	na
11	AD5	CD2	RD7	na	na	na	na
12	na	na	na	SD4	SMD3	na	XD2
13	na	na	na	na	SMD4	na	XD2
14	AD6	na	na	na	SMD4	na	XD3
15	AD8	na	na	na	na	UCD3	na
16	na	na	RD4	SD5	na	UCD2	na
17	na	na	na	na	na	UCD7	XD3
18	na	CD2	RD4	na	SMD4	na	na
19	na	na	RD3	na	SMD4	na	na
20	na	CD2	na	SD5	na	na	na

Fig. 13 Traceability table (extended)

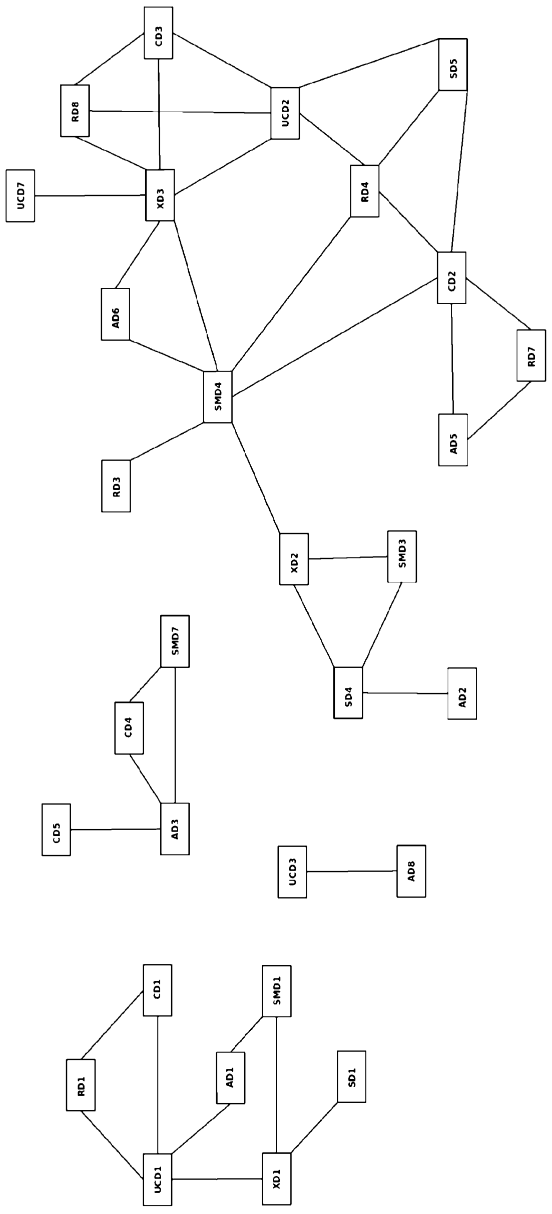
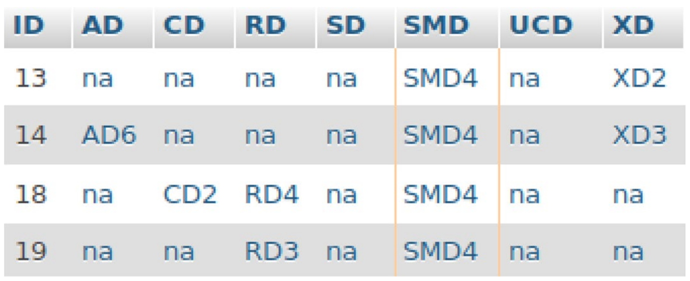


Fig. 14 Graphical view of the traceability table (extended)



ID	AD	CD	RD	SD	SMD	UCD	XD
13	na	na	na	na	SMD4	na	XD2
14	AD6	na	na	na	SMD4	na	XD3
18	na	CD2	RD4	na	SMD4	na	na
19	na	na	RD3	na	SMD4	na	na

Fig. 15 Traceability table related to SMD4

As we can see, the diagrams related to the first six rows of the traceability table are the same presented in the previous section and Fig. 12. Moreover, we represent the dummy diagrams from rows seven to twenty. The diagrams not represented in the table are the ones not connected to the others. This means that they do not contain any information in common with other diagrams and they can be deleted or modified without representing any domino effect issue for the other diagrams.

A graphical view of the diagrams connections is shown in Fig. 14. As we mentioned before, the diagrams represented here are the ones connected with at least another diagram.

Analyzing the structure of the diagrams, we can notice that there are four clusters.

On the left, we can see the same cluster presented in Fig. 10. Then, we can see a small cluster composed of only two diagrams (AD8 and UCD3) and it is related to the row 15 of the Traceability Table presented in Fig. 13. The third cluster is composed of four diagrams. Three of them are connected in a circle and they are represented in the row number 7 of the Traceability Table, the fourth diagram is CD5 and it is only connected to AD3. Finally, on the right, we can see that there is a fourth cluster. Considering all the elements, we can state that one of the most connected diagrams is SMD4. In fact, if we make the following query in the extended traceability table, we can check which diagrams are directly connected with it:

SELECT ∗ FROM ‘Traceability‘ WHERE SMD =″ SMD4″

The result is shown in Fig. 15 and it means that SMD4 is connected to AD6, CD3, RD3, RD4, XD2 and XD3. In fact, if we try to cancel SMD4 we will receive an error message telling that it is not possible to cancel the diagram because of existing external references.

It is possible to cancel or modify the diagram only after relaxing the existing connections. This is a powerful measure that avoids domino effects after the deletion of important pieces of data. After this step, the modeling phase is concluded and, in the case no further actions are needed (i.e., modify a model or delete it). It is possible to proceed to the following phase of the K-Model: the development phase.

Conclusion and future work

In this paper, we have presented a model-driven approach extending UML and SysML diagrams. The aim of this work is to provide developers with a tool helping them to consider domains such as trust and security during the SDLC of an IoT entity. About trust, we consider the distinction between evaluation and decision models in order to model the proper features related to trust stereotypes. We extend existing diagrams (i.e., class and sequence diagram) adding new features and stereotypes. Moreover, we present two new diagrams: the context and traceability diagrams. The first diagram helps developers highlight each possible context and its related domains in order to consider the different functionalities of an IoT entity. The second one is needed in order to control the connection among the other diagrams and it helps developers in avoiding domino effects due to the modification of diagrams that are connected to others. Finally, we have shown how the traceability diagram database works and how it can be used to control the connections among the other diagrams. The graphical view of the traceability table helps developers recognize which clusters are present and which diagram is more critical. The lessons learned are that the methodology allows developers to consider the needs and requirements elicited in the previous phase of the K-model avoiding the possibility to skip these crucial tasks. Then, the diagrams can be drawn in an order preferred by the developers. We however suggest one in order to cover all the possible aspects of the modeling development especially considering context and traceability.

As future work, we will develop a tool to draw the proposed diagrams in order to provide developers a proper way to implement them. Considering the traceability diagram, we will propose a way to recognize if two or more diagrams should be connected considering proper keywords related to the elements of the diagrams. Moreover, we will develop a tool support for the analysis of clusters, in order to improve the effectiveness of traceability. Finally, we will validate our model-driven approach in a real and complex scenario avoiding the utilization of dummy elements.

Abbreviations

UCD: Use case diagram; CD: Class diagram; AD: Activity diagram; SD: Sequence diagram; SMD: State machine diagram; RD: Requirement diagram; XD: Context diagram; TD: Traceability diagram.

Acknowledgements

This work has been supported by the EU project H2020-MSCA-RISE-2017 under grant agreement No 777996 (Sealed-GRID) and the EU H2020-SU-ICT-03-2018 Project No. 830929 CyberSec4Europe (cybersec4europe.eu). 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.

Authors' contributions

DF was a major contributor in writing the manuscript as a 1st Author and Corresponding Author. CFG and JL helped him with suggestions and modifications for the whole paper. All the authors read and approved the final manuscript.

Funding

This work has been supported by the EU project H2020-MSCA-RISE-2017 under grant agreement No 777996 (Sealed-GRID).

Data availability statement

The datasets related to the Use Case Scenario are available from the corresponding author on reasonable request.

Competing interests

The authors declare that has no competing interests.

Received: 20 July 2020 Accepted: 5 December 2020
Published online: 14 December 2020

References

1. Roman R, Najera P, Lopez J (2011) Securing the internet of things. Computer 44(9):51–58
2. Fernandez-Gago C, Moyano F, Lopez J (2017) Modelling trust dynamics in the internet of things. Inf Sci 396:72–82. https://doi.org/10.1016/j.ins.2017.02.039
3. Erickson J (2009) Trust metrics. In: International symposium on collaborative technologies and systems, 2009. CTS'09, IEEE, New York, pp 93–97
4. Levien RL (2002) Attack resistant trust metrics. PhD thesis, University of California at Berkeley
5. Grandison T, Sloman M (2000) A survey of trust in internet applications. IEEE Commun Surv Tutorials 3(4):2–16
6. Hoffman LJ, Lawson-Jenkins K, Blum J (2006) Trust beyond security: an expanded trust model. Commun ACM 49(7):94–101
7. Pavlidis M (2011) Designing for trust. In: CAiSE (Doctoral Consortium), pp. 3–14
8. Ferraris D, Fernandez-Gago C, Lopez J (2018) A trust by design framework for the internet of things. In: NTMS'2018—Security Track (NTMS 2018 Security Track), Paris, France
9. Mohammadi V, Rahmani AM, Darwesh AM, Sahafi A (2019) Trust-based recommendation systems in internet of things: a systematic literature review. Human Centric Comput Inf Sci 9(1):21
10. Shayesteh B, Hakami V, Akbari A (2020) A trust management scheme for IOT-enabled environmental health/accessibility monitoring services. Int J Inf Secur 19(1):93–110
11. Bordel B, Alcarria R, Martin D, Sanchez-Picot A (2019) Trust provision in the internet of things using transversal blockchain networks. Intell Autom Soft Comput 25(1):155–170
12. Lee Y, Rathore S, Park JH, Park JH (2020) A blockchain-based smart home gateway architecture for preventing data forgery. Human Centric Comput Inf Sci 10(1):1–14
13. Rumbaugh J, Jacobson I, Booch G (2004) Unified modeling language reference manual. The Pearson Higher Education, London
14. Friedenthal S, Moore A, Steiner R (2014) A practical gide to SysML: the systems modeling language. Morgan Kaufmann, Burlington
15. Marsh SP (1994) Formalising trust as a computational concept. PhD thesis, Department of Computing Science and Mathematics, University of Stirling
16. Blaze M, Feigenbaum J, Lacy J (1996) Decentralized trust management. In: Proceedings 1996 IEEE symposium onSecurity and privacy, 1996. IEEE, New York, pp 164–173.
17. Uddin MG, Zulkernine M (2008) Umltrust: towards developing trust-aware software. In: Proceedings of the 2008 ACM symposium on applied computing, ACM, New York, pp 831–836
18. Górski J, Jarzębowicz A, Leszczyna R, Miler J, Olszewski M (2005) Trust case: justifying trust in an it solution. Reliabil Eng Syst Saf 89(1):33–47
19. Ferraris D, Fernandez-Gago C (2019) Trustapis: a trust requirements elicitation method for IOT. Int J Inf Secur 19:111–127
20. Jürjens J (2005) Secure systems development with UML. Springer, Berlin
21. Basin D, Doser J, Lodderstedt T (2003) Model driven security for process-oriented systems. In: Proceedings of the eighth ACM symposium on access control models and technologies, ACM, New York, pp 100–109
22. Lodderstedt T, Basin D, Doser J (2002) Secureuml: a uml-based modeling language for model-driven security. In: International conference on the unified modeling language. Springer, Berlin, pp 426–441
23. Mai PX, Goknil A, Shar LK, Pastore F, Briand LC, Shaame S (2018) Modeling security and privacy requirements: a use case-driven approach. Inf Softw Technol 100:165–182
24. Aufner P (2020) The iot security gap: a look down into the valley between threat models and their implementation. Int J Inf Secur 19(1):3–14
25. Rathore S, Sharma PK, Loia V, Jeong Y-S, Park JH (2017) Social network security: issues, challenges, threats, and solutions. Inf Sci 421:43–69
26. Vraalsen F, Lund MS, Mahler T, Parent X, Stølen K (2005) Specifying legal risk scenarios using the coras threat modelling language. In: International conference on trust management. Springer, Berlin, pp 45–60
27. Dimitrakos T, Ritchie B, Raptis D, Stølen K (2002) Model-based security risk analysis for web applications: the coras approach. In: Proceedings of the EuroWeb. Citeseer
28. Hussein M, Zulkernine M (2006) Umlintr: a uml profile for specifying intrusions. In: 13th annual IEEE international symposium and workshop on engineering of computer based systems, 2006. ECBS 2006, IEEE, New York, pp 8
29. Sicato JCS, Singh SK, Rathore S, Park JH (2020) A comprehensive analyses of intrusion detection system for iot environment. J Inf Process Syst 16(4):975–990
30. Maskani I, Boutahar J, El Houssaïni SEG (2018) Modeling telemedicine security requirements using a sysml security extension. In: 2018 6th international conference on multimedia computing and systems (ICMCS), IEEE, New York, pp 1–6
31. Apvrille L, Roudier Y (2013) Sysml-sec: A sysml environment for the design and development of secure embedded systems. APCOSEC, Asia-Pacific Council on Systems Engineering, pp 8–11
32. Harrand N, Fleurey F, Morin B, Husa KE (2016) Thingml: a language and code generation framework for heterogeneous targets. In: Proceedings of the ACM/IEEE 19th international conference on model driven engineering languages and systems, pp 125–135
33. Mavropoulos O, Mouratidis H, Fish A, Panaousis E, Kalloniatis C (2017) A conceptual model to support security analysis in the internet of things. Comput Sci Inf Syst 14(2):557–578
34. Moyano F, Fernandez-Gago C, Lopez J (2012) A conceptual framework for trust models. In: 9th international conference on trust, privacy and security in digital business, TrustBus 2012, vol. 7449 of lectures notes in computer science. Springer, Berlin, pp 93–104
35. Kou L, Shi Y, Zhang L, Liu D, Yang Q (2019) A lightweight three-factor user authentication protocol for the information perception of iot. Comput Mater Continua 58(2):545–565
36. Jøsang A, Ismail R, Boyd C (2007) A survey of trust and reputation systems for online service provision. Decis Support Syst 43(2):618–644

37. Ferraris D, Daniel J, Fernandez-Gago C, Lopez J (2019) A segregated architecture for a trust-based network of internet of things. In: 2019 16th IEEE annual consumer communications & networking conference (CCNC) (CCNC 2019), Las Vegas, USA
38. Gafurov K, Chung T-M (2019) Comprehensive survey on internet of things, architecture, security aspects, applications, related technologies, economic perspective, and future directions. J Inf Process Syst 15(4):797–819
39. Park J-H, Salim MM, Jo JH, Sicato JCS, Rathore S, Park JH (2019) Ciot-net: a scalable cognitive iot based smart city network architecture. Human Centric Comput Inf Sci 9(1):29
40. Giorgini P, Massacci F, Mylopoulos J, Zannone N (2006) Requirements engineering for trust management: model, methodology, and reasoning. Int J Inf Secur 5(4):257–274
41. Yan Z, Holtmanns S (2008) Trust modeling and management: from social trust to digital trust. IGI Global, Hershey, pp 290–323
42. Hershey JR, Chen Z, Le Roux J, Watanabe S (2016) Deep clustering: discriminative embeddings for segmentation and separation. In: 2016 IEEE international conference on acoustics, speech and signal processing (ICASSP), IEEE, New York, pp 31–35

Publisher's Note