F. Moyano, C. Fernandez-Gago, K. Beckers, and M. Heisel, "Enhancing Problem Frames with Trust and Reputation for Analyzing Smart Grid Security Requirements", *Smart Grid Security - Second International Workshop*, LNCS vol. 8448, pp. 166-180, 2014.
http://doi.org/10.1007/978-3-319-10329-7_11
NICS Lab. Publications: https://www.nics.uma.es/publications

Enhancing Problem Frames with Trust and Reputation for Analyzing Smart Grid Security Requirements *

Francisco Moyano[1], Carmen Fernandez-Gago[1],
Kristian Beckers[2], and Maritta Heisel[2]

[1] Network, Information and Computer Security Lab
University of Malaga, 29071 Malaga, Spain
{moyano,mcgago}@lcc.uma.es
[2] paluno - The Ruhr Institute for Software Technology -
University of Duisburg-Essen, Germany
{firstname.lastname}@paluno.uni-due.de

Abstract. Smart grids are expected to scale over millions of users and provide numerous services over geographically distributed entities. Moreover, smart grids are expected to contain controllable local systems (CLS) such as fridges or heaters that can be controlled using the network communication technology of the grid. Security solutions that prevent harm to the grid and to its stakeholders from CLS are essential. Moreover, traditional security approaches such as static access control systems cause a lot of administrative workload and are difficult to maintain in fast growing and changing systems. In contrast, trust management is a soft security mechanism that can reduce this workload significantly. Even though there is not any accepted definition of trust, it is agreed that it can improve decision-making processes under risk and uncertainty, improving in turn systems' security. We use the problem frames notation to discuss requirements for a trust-based security solution concerning CLS.

Key words: Problem Frames, Model-driven Engineering, Security Requirements Engineering, Trust, Reputation, UML4PF

1 Introduction

The concept of trust has been in discussion for a long time and researchers in software engineering still work on clarifying its terminology [1]. In addition, several well known applications rely on trust and reputation mechanisms such as Amazon's product ratings and ebay's seller feedback [2].

In security engineering, current practice is to mitigate potential threats with hard trust mechanisms, which differ from soft trust ones [3]. Hard trust mechanisms aim to define strict rules in order to prevent access to resources without proper authorization.

*This research was partially supported by the EU project Network of Excellence on Engineering Secure Future Internet Software Services and Systems (NESSoS, ICT-2009.1.4 Trustworthy ICT, Grant No. 256980). The first author is funded by the Spanish Ministry of Education through the national F.P.U. program.

These rules are often in the form of permissions associated to roles. Using such a set of static rules implies a high administrative burden, they are hard to maintain in dynamic environments [4], and they only provide limited control prior to the access of users; once users are in the system, hard trust mechanisms cannot detect misbehaviours by themselves. Moreover, any misbehaviour may lead to multiple rules updates, which can lead to missing rules due to small IT staff or to wrong rules due to human errors.

Another example of hard trust mechanism is cryptography-supported trust by means of Public Key Infrastructures (PKIs). However, many challenges must be overcome to accomplish its integration into highly distributed and heterogeneous Future Internet scenarios such as the smartgrid. On the one hand, the tight resource constraints of some devices precludes the use of public-key cryptography [4]. On the other hand, in an open market where different vendors manufacture different devices, it is not realistic (at least in the beginning) to assume that they will agree on the format of certificates or on the Trusted-Third Parties that can play the role of certification authorities.

In contrast to hard trust mechanisms, soft trust mechanisms rely on the characterization of trust relationships based on certain factors that influence these relationships. Examples of these factors are previous experience, membership to a group, reputation or detected strange behaviours. Trust values can be used by the trustor itself (i.e. the entity placing trust) to evaluate if it should engage in an interaction with other entities.

The main difference with the previous schemes is that we are empowering entities to make decisions based on personal judgement of its context and knowledge. Trust is no longer based on a set of strict rules or on statement by a certification authority that is trusted by definition. Trust is based on a subjective evaluation that takes into account a set of factors that may lead entities to trust or distrust other entities, and some of these factors can be monitored autonomously. Trust and reputation are attached to entities and people, providing a better decoupling from the underlying organizational structure compared to the previous mechanisms. This is relevant as according to the European Commission [5]: “Over the period from 2002 to 2010, more than 11000 cases of restructuring were recorded by the European Restructuring Monitor”. Note that we may still need roles to be an important factor of the trust model, and in that case the decoupling would be lower. The drawback of trust-based security solutions is that they entail certain level of subjectivity and uncertainty and do not provide strong guarantees that security concerns will be correctly solved.

Two main challenges arise when we plan to incorporate soft trust mechanisms in the requirements stage of the Software Development Life Cycle (SDLC). First, how to identify the security requirements for which a soft trust approach is a feasible solution; second, how to represent the problem, that is, the security concern, and the elements of trust and reputation that surround this problem. In this paper, we address this second challenge by integrating concepts from trust and reputation in the problem frames approach [6]. We choose this approach because it focuses on describing the system-to-be in its environment. The description of the environment is essential for trust and reputation, because they rely on knowledge about external stakeholders or software entities in order to determine adequate trust or reputation values.

Our contribution is a notation that allows specifying the requirements of a system that includes a trust or reputation model. This notation is an extension over problem

frames that supports the definition of trust and reputation elements and their integration with the rest of elements (i.e. the environment) of the system. Analysts can benefit from our contribution by grasping a better understanding of the system and its interactions with the trust model, whereas designers can obtain a good starting point for planning the architecture and building trust into the architecture.

In this work, we focus on an analysis of trust and reputation relations in the system-to-be. We propose considering trust and reputation in the early phase of software engineering, because the effort for including it in later phases increases. The challenge of such an analysis is to achieve a coverage of all possible trust and reputation relations.

Goal-based methods, e.g., SI* [7] and KAOS [8], investigate the goals and views of all stakeholders of a system. These approaches model stakeholder relations based upon structured goal models. Hence, they consider all goals and relevant software artifacts to these goals. However, they do not consider a complete view of the system-to-be. Other security requirements engineering methods have a similar approach, e.g., the asset-driven risk management method CORAS [9] identifies assets and determines threats to these assets. CORAS models the system-to-be in artifacts that have a relation to an asset and also do not represent the complete system-to-be. Thus, we do not use any of these methods for our trust and reputation analysis.

The Problem Frames [6] method uses an abstraction of the system-to-be and models the environment of the system around it. Thus, this method is our choice to analyze trust relationships in the software and its environment. The method models the *Machine* and its environment in domains with certain characteristics, and we propose a trust and reputation analysis that uses these characteristic to determine trustors, trustees, claims, and other trust-related concepts. We show a structured method that elicits trust and reputation relations for each domain. In the future, we will also provide computer-aided support for consistency, and security reasoning for this method by using OCL [10] queries on the problem frame models. Hence, we use the benefit of having a complete model of the system-to-be and its environment in domains to conduct a threat analysis.

We use the UML representation of the problem frames method called *UML4PF* [11], because this allows us to write OCL expressions to validate the models that will be included in the UML4PF support tool. Moreover, we aim to integrate this analysis into a structured software development process, e.g., an extension of the ADIT [12] process that relies on UML4PF. We choose the UML notation, because software engineers are familiar with it to express software design choices. Moreover, if we express the software analysis and design in UML, we do not need to map the analysis results to a different notation for the software design. This reduces one source of mistakes during software development. Hence, expressing trust and reputation analysis in UML allows for a seamless refinement step to software design, by re-using the UML models created during the analysis phase in the software design phase.

The remainder of our paper is structured as follows. Section 2 explains background on trust and problem frames, as well as some related work. Section 3 shows our UML profile, which illustrates elements of trust, problem frames and their relations. We apply our profile to a smart grid example in Sect. 4, whereas in Sect. 5 we draw some conclusions and give lines of future research.

2 Background and Related Work

We explain trust concepts in Sect. 2.1, problem frames in Sect. 2.2, and related work in software engineering in Sect. 2.3.

2.1 Trust Background and Terminology

There has been a huge amount of definitions of trust over the years. We propose the following definition: *trust is the personal, unique and temporal expectation that a trustor places on a trustee regarding the outcome of an interaction between them*. This interaction usually comes in terms of a task that the trustee must perform and that can (negatively) influence the trustor. The expectation is personal and unique because it is subjective, and is temporal because it may change over time. Tasks belong to a context, in such a way that a trustor may place different trust values on the same trustee depending on the the context where trust is applied. The concept and implications of trust are embodied in so-called *trust models*, which manage trust relationships between trustors (entities that place trust) and trustees (entities onto which trust is placed). Many trust models have been proposed in the literature, but we are particularly interested in evaluation models, as proposed by Marsh in his seminal work [13]. In these models, factors that have an influence on trust are identified, quantified and then aggregated into a final trust score by the trust engine of the trust model. Uncertainty and evaluation play an important role in these models, as the trustor has only limited confidence on a positive output after the interaction with the trustee, and a quantification process is required to evaluate the extent to which one entity trusts another one.

Regarding reputation, the Concise Oxford dictionary[3] defines it as "what is generally said or believed about a person or the character or standing of a thing". The word *generally* implies that reputation is formed by an accumulation of opinions, which makes reputation a more objective concept than trust. A good approximation to the relationship between trust and reputation was suggested by Jøsang [14], who made the following two statements: 'I trust you because of your good reputation' and 'I trust you despite your bad reputation'. Reputation can be considered as a building block of trust but, as stated by the second statement, reputation has not the final say. One could either trust someone with low reputation or distrust someone with high reputation, because there are other factors that may have a bigger influence over the trust decision, such as the trustor's disposition to believe in the trustee, the trustor's feelings, or above all, the trustor's personal experiences with the trustee.

A core concept behind reputation as seen in web reputation models is a *reputation statement*, which can be defined as a claim stated by a source regarding a target. As an example, if Alice says: 'The film Titanic has a good photography', the source is *Alice*, the target is *film Titanic*, and the claim is *to have a good photography*. A source can be human or non-human. Non-human sources include include anti-spam filters, input from other reputation models, log crawlers or recommendation engines. A target can be human, non-human or reputation statements themselves. For instance, a user Alice might claim that the review performed by Bob regarding the film Titanic was useful. In this

[3] http://www.oxforddictionaries.com

case, the target is *Bob's review about Titanic*, that is, a reputation statement where the source Bob expressed its opinion (claim) about the target Titanic. Reputation engines take reputation statements about a given target as inputs, and produce a reputation score for the target.

2.2 Problem Frames

Problem frames are a means to describe software development problems. They were proposed by Jackson [6], who describes them as follows: *"A problem frame is a kind of pattern. It defines an intuitively identifiable problem class in terms of its context and the characteristics of its domains, interfaces and requirement."*. It is described by a *frame diagram*, which consists of domains, interfaces between them, and a requirement. We describe problem frames using class diagrams extended by stereotypes as proposed in [15, 11]. All elements of a problem frame diagram act as placeholders, which must be instantiated to represent concrete problems. Doing so, one obtains a problem description that belongs to a specific kind of problem. The class with the stereotype *machine* represents the thing to be developed (e.g., the software). The classes with some domain stereotypes, e.g., *CausalDomain* or *BiddableDomain* represent *problem domains* that already exist in the application environment. Domains are connected by interfaces consisting of shared phenomena. Shared phenomena may be events, operation calls, messages, and the like. They are observable by at least two domains, but controlled by only one domain, as indicated by an exclamation mark. These interfaces are represented as associations, and the name of the associations contains the phenomena and the domains controlling the phenomena. Jackson distinguishes the domain types *CausalDomain*s that comply with some physical laws, *LexicalDomain*s that are data representations, and *BiddableDomain*s that are usually people. The stereotype <<causalDomain >> indicates that the corresponding domain is a *CausalDomain*, and the stereotype <<biddableDomain >> indicates that it is a *BiddableDomain*. In our formal meta-model of problem frames [16], *domains* have *names* and *abbreviations*, which are used to define interfaces. Hence, the class *Domain* has the attributes *name* and *abbreviation* of type string.

Software development with problem frames proceeds as follows: first, the environment in which the machine will operate is represented by a *context diagram*. Like a frame diagram, a context diagram consists of domains and interfaces, but the diagram does not contain requirements. *Domain knowledge diagrams* focus on some domains of the context diagram and document further domain knowledge about them in terms of facts and assumptions. Then, the problem is decomposed into subproblems. Each subproblem is represented by a *problem diagram* containing its domains, phenomena, interfaces, and their relations to at least one requirement that expresses the subproblem. Since the requirements refer to the *environment* in which the machine must operate, the next step consists in deriving a *specification* for the machine (see [17] for details). The specification describes the machine and is the starting point for its construction.

2.3 Related Work

To the best of our knowledge no problem frame extension exists that considers trust and reputation concepts with the purpose of describing requirements concerning trust and reputation concepts.

The software engineering community has focused on specifying traditional security requirements, such as confidentiality or authorization, during the early phases of the Software Development Life Cycle (SDLC). Haley et al. [18, 19] represent security requirements in problem frames. The authors represent security requirements also as trust assumptions, which describe that the security requirement is fulfilled for a particular context, because it is trusted to satisfy the security requirement explicitly. Further examples for modeling notation that consider security are UMLsec [20] and SecureUML [21]. Other notations take relationships between actors and agents into account during the system specification. Mouratidis and Giorgini [22] present Secure Tropos, a notation that extends the Tropos methodology in order to enable the design of secure systems. Actors in Tropos may depend on other actors in order to achieve a goal. Tropos captures the social relationships in the system by specifying the dependencies between actors using the notions of depender, dependum and dependee, and by modeling the actors and agents in the organization. In a similar direction, Lamsweerde and Letier present KAOS [23], a comprehensive goal-oriented method to elicit the requirements of socio-technical systems. Moyano et al. [24] propose a trust model for Si* [7] in order to detect insider threats in an organizational setting during the initial steps of the SDLC. This work proposes setting users permissions on resources or assets, and a level of trust in these permissions. Then, threats, which are implicit wrong permissions, are discovered by examining and navigating through social relationships among actors. All these contributions put forward the idea of capturing social aspects, but the notion of trust and its influence on the information systems are barely explored. This is partially covered by Pavlidis, Mouratidis and Islam [25], who extend the Secure Tropos modeling language in order to include some trust-related concepts. The main difference is that our extension is over problem frames instead of over Secure Tropos. Problem frames are more focused on modelling the system in its environment, which we consider to be useful for trust modelling, and it represents information at a higher level of abstraction.

3 UML Profile for problem-based Trust Analysis

Our profile considers Jackson's domain types (as discussed in Sect. 2.2): *CausalDomain*s, *LexicalDomain*s, *BiddableDomain*s,

Domain Knowledge consist of *Statement*s about domains, in particular *Fact*s that we can prove and *Assumption*s that we consider during software development. A *Requirement* is a specific kind of *Statement* about domains that shall hold after the *Machine* has been built. Requirements ≪constrain≫ at least one domain and can ≪referTo≫ further domains. A *securityRequirement* is a statement about the confidentiality, integrity, or availability concerns of domains and ≪complement≫ at least one functional requirement in this regard.

We extend our UML profile for Jackson's problem frame notation called *UML4PF* [12] and its dependability extension [15] with required elements to describe trust rela-

tionships and trust requirements. We use the profile to create context diagrams, domain knowledge diagrams, and problem diagrams using the elements described in Sect. 2.2. Our trust extension for the UML4PF profile is shown in Fig. 1[4] , where all the contributions of this paper are marked in grey. We define relations between Jackson's domains and the elements of Moyano et al.'s trust framework [1]. Each of these elements are now a kind of domain. *Entity* is a domain and *Human Entity* is a *Biddable Domain*. *Trust Information* and *Reputation Information* are *Lexical Domain*s.

We aim to build s specific set of *Machine*s in order to integrate trust and reputation mechanisms into a system-to-be. These are *Computation Engine*s, which in turn can be *Trust Engine*s or *Reputation Engine*s, depending on whether they calculate trust or reputation, respectively. *Trust Engines* are in charge of calculating *Trust Value*s for *Trust Relationship*s among *Entit*ies. These engines take *Trust Factor*s, associated to *Entity* as input, which may be *Objective Factor* or *Subjective Factor*. Factors can be assigned explicitly or can be obtained by some sort of monitoring; in any case, they are responsible for some other *Entity* playing the role *factor producer*. *Computation Engine*s can have different mathematical mechanics, including *belief* or *fuzzy* logics. *Uncertainty* estimates the probability of a trust or reputation value being accurate. The *Time* states when a trust relationship or reputation related statement or information was defined.

*Entit*ies playing the role *Source* can make *Claim*s about other *Entit*ies with role *Target*. This information is aggregated in the form of *Reputation Statement*s, which are used by *Reputation Engine*s to compute reputation scores. A *SourceEntity* can make *Claim*s *after an interaction* or just at *any moment*. The model considers *Human Entities*, who have an implicit *trust disposition* and who value their *Asset*s and wish to minimize *Risk* to these *Asset*s. *Countermeasure*s reduce the risk to *Asset*s. Finally, *Event*s are circumstances in the system that trigger a trust or reputation update. These events can be visualized by behavioral diagrams, such as sequence diagrams, as depicted and further discussed in Fig. 6.

4 Applying the Trust-extension of the UML4PF Profile to a Smart Grid Example

We use the protection profile for the smart metering gateway [26] as an example for our approach. The gateway is a part of the smart grid. This is a commodity network that intelligently manages the behavior and actions of its participants. The commodity consists of electricity, gas, water or heat that is distributed via a grid (or network). The benefit of this network is envisioned to be a more economic, sustainable and secure supply of commodities. Smart metering systems meter the production or consumption of energy and forward the data to external entities. This data can be used for billing and steering the energy production. The protection profile defines security requirements for a smart metering gateway [26] and we use the UML profile as the source for the following example.

[4]Note that for readability purposes we simplified the profile and several domains are not illustrated in Fig. 1, e.g., display domains and assets.

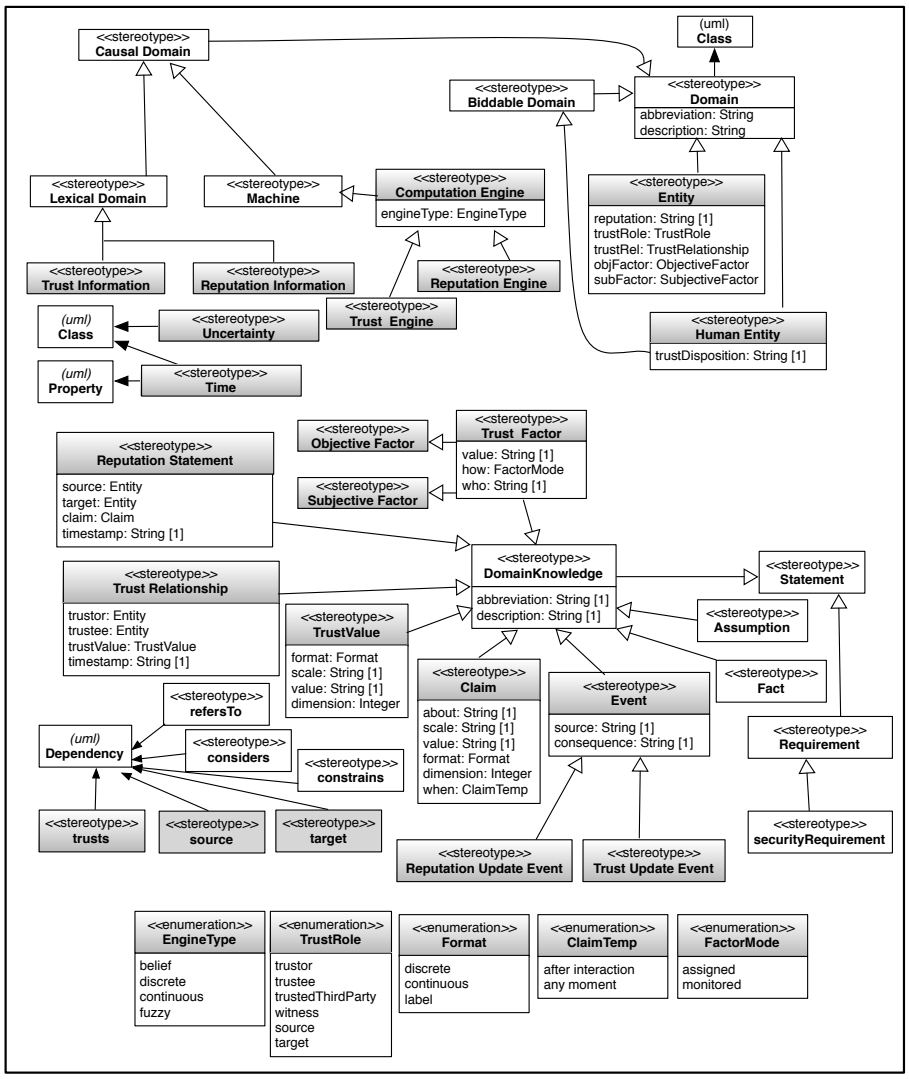

Fig. 1. A Trust extension of the UML4PF Profile

The context diagram shown in Fig. 2 describes the machine to be built in its environment. It is part of the overview description of the security target. The ≪Machine≫ is the *SmartMeteringGateway*, which serves as a bridge between the Wide Area Network ≪wan≫ and the Local Network ≪physical≫ of the *Consumer*. The *Meter* is connected to the machine via a Local Metrological Network ≪lmn≫. The Meter is an in-house equipment that can be used for energy management. The Controllable Local System *CLS* can be, for example, an air conditioning unit or an intelligent refrigera-

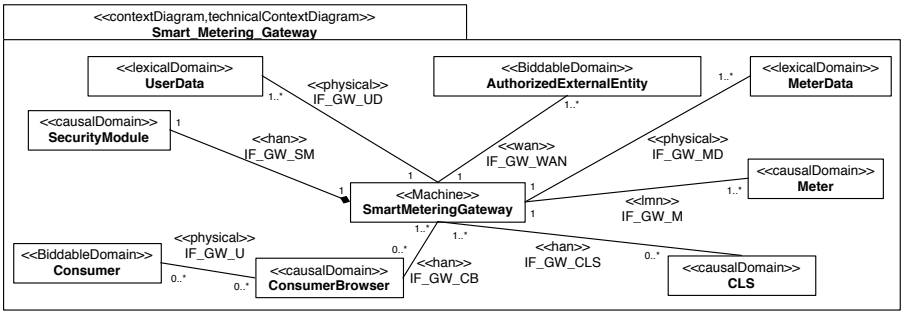

Fig. 2. The Context Diagram of the Smart Metering Gateway

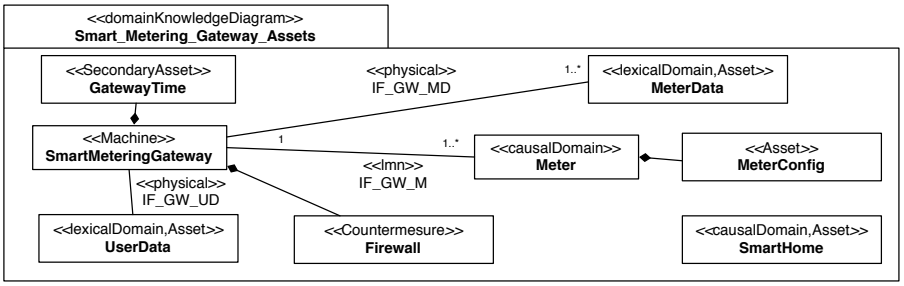

Fig. 3. Domain Knowledge Diagram Concerning Assets and Existing Countermeasures

tor. The *Consumer* can also access the *Machine* [26] via a *ConsumerBrowser*. We extended the description with the following phenomena. The *Meter* sends meter data to the *SmartMeteringGateway*. The *SmartMeteringGateway* stores this data. The *Meter* can also receive updates from the *AuthorizedExternalEntity* forwarded via the *SmartMeteringGateway*. The *AuthorizedExternalEntity* retrieves sent meter data in fixed intervals from the *SmartMeteringGateway*. The *SecurityModule* provides cryptographic functionalities for the *SmartMeteringGateway* such as key generation and random number generation. The *Consumer* can retrieve meter data via the *SmartMeteringGateway* and the *ConsumerBrowser*. The *Consumer* can also configure the *SmartMeteringGateway*, send commands to the *CLS*, receive status messages from the *SmartMeteringGateway* and store *UserData* in it.

We iterated over the domains in Fig. 2 and identified the *MeterData* as an ≪asset≫. Figure 3 presents a domain knowledge diagram that contains the description of this asset. The meter data has value for the *Consumer*, because his/her billing depends upon it and a behavior profile about the *Customer* can be created from it. Integrity, authenticity, and confidentiality of this data need to be protected. Another asset of the *SmartMeteringGateway* is the *GatewayTime*. The asset is revealed via investigating assumptions about the *SmartMeteringGateway*, namely that the meter data is recorded with a correct time stamp. The time is used in *MeterData* records that are sent to *AuthorizedExter-*

nalEntity for, e.g., billing. Its integrity and authenticity have to be protected and especially the time adjustment using an externally referenced time is critical.

Some functional requirements of the smart metering gateway are:

R 1 The *CLS* can receive energy consumption data from the *Meter*
R 2 The *CLS* can communicate with an *External Entity* using the WAN
R 3 The *Consumer* can communicate with the *CLS*

We model the trust relationships relevant for the aforementioned considerations in a domain knowledge diagram (see Fig. 4). We focus on the trust relationship *Consumer_CLS_Trust* between the trustor *Consumer* and the trustee *CLS*. This relationship expresses the trust that the trustor has in the trustee concerning the integrity of its configuration and the preservation of confidentiality of private billing information.

We also illustrate in the figure that *OtherConsumer*, *AuthorizedExternalEntity*, and *SmartMeteringGateway* are trust entities (i.e. ≪Entity≫) in the sense that they are sources of reputation for the *CLS*. Concretely, *OtherConsumer*s can report their experience with the *CLS* after interacting with it by using a continuous number between 0 and 1. For example, *OtherConsumer* could be another home user (not the main user), who after asking the fridge for a list of food that is running out, could physically check whether the information was accurate and evaluate with a value between 0 and 1 accordingly. *AuthorizedExternalEntity*, who may represent administrators or technicians, can report the same information after a check-up of the *CLS*, but this time by using a discrete value between 0 and 10. Finally, the *SmartMeteringGateway* can report a claim about the behavior of the *CLS* in terms of privacy awareness. Each time the *CLS* sends some information outside the home environment, the gateway analyses the information and issues a claim in a labeled scale between *very good* and *very bad*, according to the sensitivity and quantity of the information sent.

We draw a problem diagram for each requirement in order to refine it. We present a problem diagram for **R1** in Fig. 5. **R1** ≪constraints≫ the *CLS* in such a way that it can receive ≪MeterData≫. Moreover, we use the information in Fig. 4 to devise a trust-based security treatment. The ≪securityRequirement≫ *CLS Protection* describes that we must preserve both the integrity of the configuration data of *CLS* and the confidentiality of the *MeterData* of the *Consumer* when it is used by a *CLS*. *CLS Protection* ≪complements≫ **R1**. We require a ≪TrustEngine≫ called *CLS-TrustEngine* to calculate the trust values, which in turn relies on a ≪ReputationEngine≫ called *CLS-ReputationEngine*. The *CLS-TrustEngine* considers both the reputation score and the explicit trust of the consumer when calculating trust values, whereas the *CLS-ReputationEngine* uses the claims made by *AuthorizedExternalEntity*, *SmartMeterGateway*, and *OtherConsumer*. *CLS-TrustEngine* considers the trust factors and the *CLS-ReputationEngine* the claims illustrated in Fig. 4. For simplicity's sake, we do not show these again in Fig. 5, but they have to be considered during the refinement of the problem diagram into an implementable software specification of the trust and reputation engines.

Figure 6 illustrates the use of trust in the system, and concretely, an event that causes the update of a trust relationship between the *Consumer* and the *CLS*. We consider that the *SMG* has a built-in firewall that allows controlling the information flowing to and

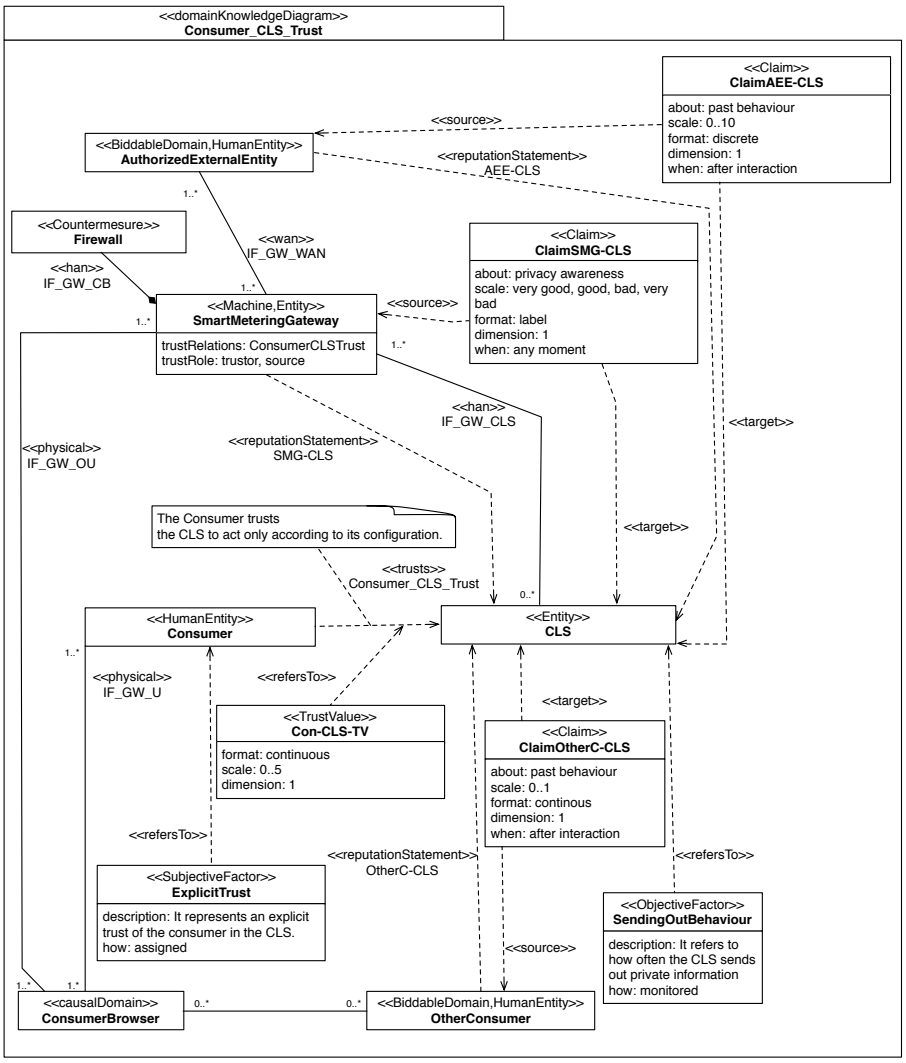

Fig. 4. Domain knowledge diagram concerning trust relations and reputation

from the *CLS* as well as preventing changes to the *CLS* configuration. The Common Criteria protection profile [26] states that the gateway has already a firewall to protect the *Meter* functionality. We propose to extend this firewall to protect the *CLS* functionality, as well.

The *SMG* detects that the *CLS* is leaking private information that (in its understanding and according to some policy) should not be passed through. In addition to blocking the information, the *SMG* sends a claim, which triggers a reputation update. Once reputation is updated, *SMG* requests an update of the trust relationship between the

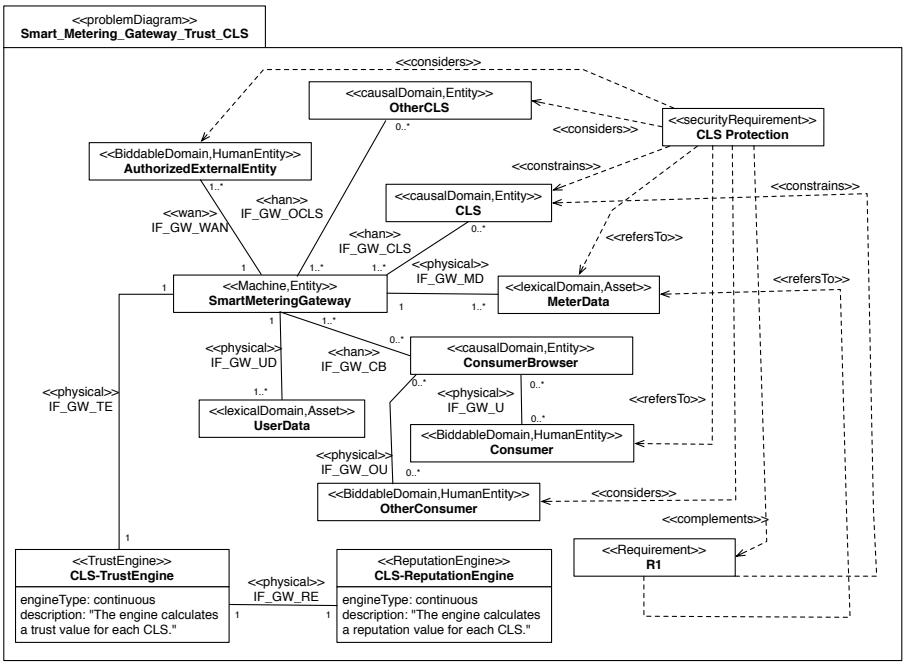

Fig. 5. Problem diagram considering a trust and a reputation engine

Consumer and the *CLS*. In order to compute the new trust value, the *CLS-TrustEngine* needs the explicit trust value defined by the *Consumer* for that particular *CLS* as well as the reputation of the *CLS*[5]. Upon receiving the new trust value, the *SMG* updates the firewall rules. For example, in case the *Consumer* does not trust the *CLS* above a given, configurable threshold, all requests from the *Consumer* to the *CLS* are blocked by the *SMG*, and all the messages flowing out from the *CLS* are also blocked, isolating this device from the rest of the system.

5 Conclusion

We have extended UML4PF, which is a UML-profile based on Jackson's Problem Frame notation, with concepts of trust and reputation. In particular, we related these concepts to Jackson's domains and gave some hints on how to describe security requirements that consider trust and reputation. We applied the extended UML4PF profile with trust and reputation concepts to a smart grid example and illustrated the following:

[5]We are assuming a trust model consisting of two factors: an explicit trust assigned by the user and the reputation of the trustee, which is computed by aggregating different claims of *OtherConsumer*s, *AuthorizedExternalEntity* and *SmartMeteringGateway*. However, any other kind of trust model that considers other factors can be specified.

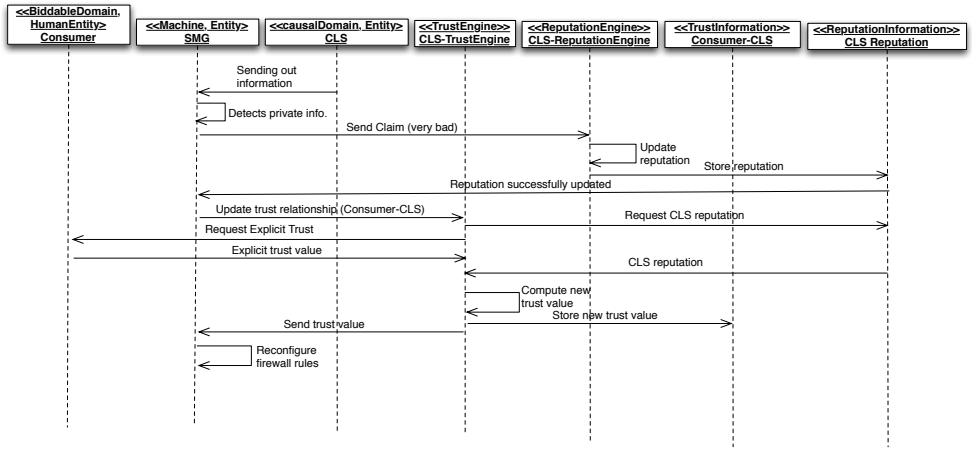

Fig. 6. Sequence Diagram for Trust Update Event

- How to elicit trust and reputation information for a specific context.
- Explicit documentation of domain knowledge in terms of trust relations, reputation relations, claims, trust values, etc.
- Describing security requirements that consider trust and reputation domain knowledge with the purpose of building trust and reputation engines to protect assets.
- Refine the static descriptions of trust and reputation engines in problem diagrams with descriptions of their dynamic behavior in UML sequence diagrams.

In the future, we will create a structured method with tool support for creating trust and reputation engines. In particular, we will focus on supporting the modeling and we will provide OCL-based consistency checks of the models. In addition, we will analyze the relations of security controls of the ISO 27001 [27] standard to trust and reputation concepts. We assume the results will provide insights into which ISO 27001 controls can benefit from trust and reputation engines.

References

1. Moyano, F., Fernandez-Gago, C., Lopez, J.: A conceptual framework for trust models. In Fischer-Hübner, S., Katsikas, S., Quirchmayr, G., eds.: 9th International Conference on Trust, Privacy & Security in Digital Business (TrustBus 2012). Volume 7449 of Lectures Notes in Computer Science., Vienna, Springer Verlag, Springer Verlag (Sep 2012 2012) 93–104
2. Kirtland, Alex and Schiff, Aaron: On A Scale of 1 to 5: Understanding Risk Improves Rating and Reputation Systems. `http://boxesandarrows.com/on-a-scale-of-1-to-5/` (Jun 2008)
3. Rasmusson, L., Jansson, S.: Simulated social control for secure internet commerce. In: Proceedings of the 1996 workshop on New security paradigms. NSPW '96, New York, NY, USA, ACM (1996) 18–25

4. Roman, R., Zhou, J., Lopez, J.: On the features and challenges of security and privacy in distributed internet of things. Computer Networks **57** (July 2013) 2266–2279
5. European Commission: Restructuring in Europe 2011: Restructuring and anticipation of change, what lessons from recent experience? `http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=SEC:2012:0059:FIN:EN:PDF` (Jan 2012)
6. Jackson, M.: Problem Frames. Analyzing and structuring software development problems. Addison-Wesley (2001)
7. Massacci, F., Mylopoulos, J., Zannone, N.: Security requirements engineering: The si* modeling language and the secure tropos methodology. In Ras, Z., Tsay, L.S., eds.: Advances in Intelligent Information Systems. Volume 265 of Studies in Computational Intelligence. Springer Berlin / Heidelberg (2010) 147–174
8. van Lamsweerde, A.: Requirements Engineering: From System Goals to UML Models to Software Specifications. 1st edn. John Wiley & Sons (2009)
9. Lund, M.S., Solhaug, B., Stølen, K.: Model-Driven Risk Analysis: The CORAS Approach. 1st edn. Springer (2010)
10. UML Revision Task Force: OMG Object Constraint Language: Reference (February 2010)
11. Côté, I., Hatebur, D., Heisel, M., Schmidt, H.: UML4PF – a tool for problem-oriented requirements analysis. In: Proceedings of the International Conference on Requirements Engineering (RE), IEEE Computer Society (2011) 349–350
12. Côté, I.: A Systematic Approach to Software Evolution. Deutscher Wissenschafts-Verlag (DWV) Baden-Baden (2012)
13. Marsh, S.: Formalising Trust as a Computational Concept. PhD thesis, University of Stirling (April 1994)
14. Jøsang, A., Ismail, R., Boyd, C.: A survey of trust and reputation systems for online service provision. Decision Support Systems **43**(2) (March 2007) 618–644
15. Hatebur, D., Heisel, M.: A UML profile for requirements analysis of dependable software. In Schoitsch, E., ed.: Proceedings of the International Conference on Computer Safety, Reliability and Security (SAFECOMP) (LNCS 6351), Springer (2010) 317–331
16. Hatebur, D., Heisel, M., Schmidt, H.: A formal metamodel for problem frames. In: Proceedings of the International Conference on Model Driven Engineering Languages and Systems (MODELS). Volume 5301., Springer Berlin / Heidelberg (2008) 68–82
17. Jackson, M., Zave, P.: Deriving specifications from requirements: an example. In: Proc. 17th Int. Conf. on Software Engineering, Seattle, USA, ACM Press (1995) 15–24
18. Haley, C.B., Laney, R.C., Nuseibeh, B.: Deriving security requirements from crosscutting threat descriptions. In: Proceedings of the 3rd International Conference on Aspect-oriented Software Development. AOSD '04, ACM (2004) 112–121
19. Salifu, M., Yu, Y., Nuseibeh, B.: Specifying monitoring and switching problems in context. In: Requirements Engineering Conference, 2007. RE '07. 15th IEEE International. (2007) 211–220
20. Jürjens, J.: UMLsec: Extending UML for Secure Systems Development. In: Proceedings of the 5th International Conference on The Unified Modeling Language. UML '02, London, UK, UK, Springer-Verlag (2002) 412–425
21. Lodderstedt, T., Basin, D.A., Doser, J.: SecureUML: A UML-Based Modeling Language for Model-Driven Security. In: Proceedings of the 5th International Conference on The Unified Modeling Language. UML '02, London, UK, UK, Springer-Verlag (2002) 426–441
22. Mouratidis, H., Giorgini, P.: Secure Tropos: a Security-Oriented Extension of the Tropos Methodology. International Journal of Software Engineering and Knowledge Engineering **17**(2) (2007) 285–309
23. van Lamsweerde, A., Letier, E.: Handling Obstacles in Goal-Oriented Requirements Engineering. IEEE Trans. Softw. Eng. **26**(10) (October 2000) 978–1005

24. Paci, F., Fernandez-Gago, C., Moyano, F.: Detecting insider threats: a trust-aware framework. In: 8th International Conference on Availability, Reliability and Security, Regensburg, Germany, IEEE, IEEE (Nov 2013 2013) 121–130
25. Pavlidis, M., Mouratidis, H., Islam, S.: Modelling Security Using Trust Based Concepts. IJSSE **3**(2) (2012) 36–53
26. BSI: Protection Profile for the Gateway of a Smart Metering System (Gateway PP). Version 01.01.01(final draft), Bundesamt für Sicherheit in der Informationstechnik (BSI) - Federal Office for Information Security Germany (2011) https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/SmartMeter/PP-SmartMeter.pdf?__blob=publicationFile.
27. ISO/IEC: Information technology - Security techniques - Information security management systems - Requirements. ISO/IEC 27001, International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) (2005)