M. Kolar, C. Fernandez-Gago, and J. Lopez, "A Model Specification for the Design of Trust Negotiations", *Computers & Security*, vol. 84, pp. 288-300, 2019.
http://doi.org/10.1016/j.cose.2019.03.024
NICS Lab. Publications: https://www.nics.uma.es/publications

A Model Specification for the Design of Trust Negotiations

Martin Kolar[a,*], Carmen Fernandez-Gago[a], Javier Lopez[a]

[a]*Network, Information and Computer Security (NICS) Lab, University of Malaga, Spain*

Abstract

Trust negotiation is a type of trust management model for establishing trust between entities by a mutual exchange of credentials. This approach was designed for online environments, where the attributes of users, such as skills, habits, behaviour and experience are unknown. Required criteria of trust negotiation must be supported by a trust negotiation model in order to provide a functional, adequately robust and efficient application. Such criteria were identified previously. In this paper we are presenting a model specification using a UML-based notation for the design of trust negotiation. This specification will become a part of the Software Development Life Cycle, which will provide developers a strong tool for incorporating trust and trust-related issues into the software they create. The specification defines components and their layout for the provision of the essential functionality of trust negotiation on one side as well as optional, additional features on the other side. The extra features make trust negotiation more robust, applicable for more scenarios and may provide a privacy protection functionality.

Keywords: Trust Negotiation, Trust Model, Software Development Life Cycle, UML, Policy.

1. Introduction

Our modern world is full of entities, such as people, organisations and web sites that form a complicated hierarchy of relationships. They cooperate together and share many resources, such as commodities, estates, jobs and knowledge. In order to collaborate, entities may need to establish a trust relationship between themselves. In the physical world, trust relationships are evolving naturally as the entities interact together and as they are getting to know each other better and gain confidence about the other's expected qualities, such as honesty, reliability, knowledge, etc. For online environments, a different approach is needed. Trust negotiation represents a suitable solution, where a trust relationship is evolved by a mutual exchange of credentials. It is useful for entities willing to build trust that may not know each other well or are total strangers. Trust negotiation allows an entity to obtain enough relevant information about the other one in order to build trust (Winsborough et al., 2000). It is also flexible and can be successfully applied for building trust on demand for various scenarios in a decentralised network. Although trust relationships are dynamic and can change over time, trust negotiation is focused on building the initial trust needed for a given purpose. Once the required trust is established, trust negotiation terminates. It is a process that covers many important aspects, such as security, protection and optionally privacy of exchanged credentials, definition of policies, initialisation, termination and evaluation of trust values and results. Because of that, an entity performing trust negotiation has to manage many actions, such as exchanging credentials with the other side, evaluating trust based on received credentials, comparing it against the required level for reaching a common goal and making credentials disclosure decisions (Winsborough and Li, 2002). However, there is a lack of guidance for developers to include trust negotiation in the service they are building. For this reason, the developers that want to include trust in their software, may find practical to use a trust framework to handle all the needed actions for them. It is then this the main purpose of our paper. In this work, we are suggesting a model specification that will be applicable for trust negotiation. We will use a UML-based notation to outline its concepts as the UML is a widely accepted modelling language and its diagrams are standardised and easy to follow for our purpose. The use cases will be defined by developers according to their needs. Our intention is to create a complete trust negotiation framework that will guide developers to resolve trust related issues in their design and to include trust negotiation into their software. This framework will be applicable for many trust environments and should be as general as possible in the traditional ones as well as in the online ones. We will include our proposed specification into the Software Development Life Cycle (SDLC) as a part of its design phase. SDLC serves developers as a manual for creating a high-quality software that guides them through its particular phases, where each phase solves a different kind

*Corresponding author
Email addresses: kolar@lcc.uma.es (Martin Kolar), mcgago@lcc.uma.es (Carmen Fernandez-Gago), jlm@lcc.uma.es (Javier Lopez)

of problems of the development, but each one is important for the result. Generally, the SDLC does not include trust issues in its guidance. We want to extend the standard SDLC with trust, so that each phase will guide the developers apart from handling the usual problems also through the problems of including trust. It will help developers to create a high-quality trust-aware software from scratch. The specification defines the major trust negotiation components, where each one takes a responsibility for its own part and thus takes it away from the entity. However, the entity, as the owner of resources, should have total control over them. It should define its own policies and the other related issues, such as the decision, whether its goal is only to establish trust or to protect its privacy as well.

The rest of this paper is organized as follows. In Section 2, previous work on trust negotiation, policies and SDLC is presented. Section 3 specifies the UML-based model specification for trust negotiation and Section 4 demonstrates an example application of this specification. The model specification integration into the SDLC is shown in Section 5 and its comparison against other methods is presented in Section 6. Section 7 provides some hints and guidelines for how the validation will be like and Section 8 concludes this paper and outlines the future work.

2. Related Work

Trust is essential for everyday life and it is also an important concept in Computer Science. Entities usually require trust for their communication and cooperation. Without it they may feel uncomfortable, as they cannot rely on the obtained information or be sure about claims and real intentions of the others. However, defining trust is not a straightforward task and therefore there is not a unique definition for it. Gambetta et al. (2000) define trust as a subjective probability, by which an individual A expects another individual B to perform a given action, on which its welfare depends. It is supposed that the trustor is dependent on the trustee, where the trustee is reliable. Falcone and Castelfranchi (2001) claim that having high trust in an entity might not be generally enough to decide to become dependent on that entity. Jøsang et al. (2007) classify trust into two categories, such as reliability trust and decision trust. Reliability trust can be understood as the reliability of something or somebody, while decision trust can be understood as the extent of willingness of an entity to be dependent on another one. Wang and Emurian (2005) claim that multiple definitions of trust exist mainly for two reasons. At first, sometimes it is difficult to clearly distinguish trust from its other related concepts, such as credibility, reliability and confidence. Then, trust is influenced by many aspects, such as cognition, emotions or behaviour. The authors also presents the characteristics of trust and compare them with the ones of the online trust that is used for the online environments. The online trust concept is more specific than the traditional one as it usually represents relationships between a customer and an e-commerce web site. Corritore et al. (2003) present a definition of online trust for an individual person towards a specific website that is: "an attitude of confident expectation in an online situation of risk that one's vulnerabilities will not be exploited". Online trust relationships evince a higher degree of vulnerability as the e-commerce area is complex and anonymous. The participating entities are strangers to each other and their manners can be unpredictable. Therefore, it is important to establish a secure and trusted environment as presented by Tsiakis and Sthephanides (2005). The authors analyse the requirements for electronic payments, such as security and online trust. They characterise a trusted environment by the following features: all entities are uniquely identifiable, they have placed a firm trust onto the others and their trust is not given by default.

Trust may be established one-way from an entity to another one or two ways mutually between them. Winsborough et al. (2000) underline two main approaches of establishing trust, such as the identity-based and capability-based approach. The former authenticates an entity based on its public identity and the latter authorizes an entity based on its required attributes. These mainstream approaches are traditional, but they are not suitable for open systems, such as online environments. It is because the entities may not be familiar and their attributes may be unknown. The solution here is to build trust by the mutual exchange of credentials, which is the case of trust negotiation models.

A trust model is an abstraction of the trust relation dynamics between entities, while it examines, defines and evaluates the trust interactions. Trust models can be classified into decision models and evaluation models (Lara, 2015). Decision models facilitate access control decisions by simplifying authentication and authorisation into a single trust decision. They can be further divided into policy models and negotiation models, which is the case of trust negotiation. Evaluation models use a different approach, where trust is evaluated by considering various factors that may influence the trust relationship. They can be further divided into propagation models and reputation models. A trust management system is an abstract system that processes symbolic representations of social trust, usually to support the automated decision-making process. The first trust-based negotiation-model was TrustBuilder and its enhanced version TrustBuilder2 (Winslett et al., 2002; Lee et al., 2009). Both simplify trust negotiation as they allow entities to delegate the whole process of trust negotiation onto negotiators called security agents. Winsborough et al. (2000) use also this concept in their framework. PROTUNE is another trust negotiation model (Bonatti et al., 2010). It is rule-based and makes use of agents too, but in this case they are assigned to roles as the clients and servers. The clients make requests of resources to the servers and the servers respond to them. Trust negotiation has significant benefits compared to the traditional ways of building trust, such as its universality. The negotiat-

ing entities can be anonymous, they do not need to know each other from the past. Trust is built by obtaining and revealing information from the received credentials.

Cassandra (Becker and Sewell, 2004) protects a negotiator by creating a protective layer around his resources and allows the exchange of credentials only through a controlled interface. Cassandra is a trust management system, which is a unified approach for specifying and interpreting security policies, credentials and relationships. Its purpose is making authorizations of security-critical actions. It consists of five basic components, such as a language for describing actions, specifying policies and specifying credentials, a mechanism for identifying principals and a compliance checker. Principals are entities that can be authorized to perform an action. PolicyMaker (Blaze et al., 1996, 1998) and KeyNote (Blaze et al., 1999) are other trust management systems. PolicyMaker acts quite like a database query engine. An entity provides local policies, credentials and a trusted action description, so that the PolicyMaker can evaluate them in order to give a recommendation, if and how the trusted action should be performed. KeyNote is designed for various small and large scale Internet-based applications, while aiming to be simple and flexible. It provides a single language for both policies and credentials and makes use of assertions that are basically small programs describing trusted actions. Saied et al. (2013) present a context-aware and multi-service trust management system for the Internet of Things (IoT). They focus on managing cooperation between heterogeneous nodes with diverse capabilities and establishing a community of trusted elements that can collaborate together. Mármol and Pérez (2009) analyse various security threat scenarios in trust and reputation models and for each one suggest a suitable solution.

A set of rules must be established that would specify, how credentials can be accessed and used. A combination of these rules form a policy. For the security reasons policies should be defined the way that once they grant access to a resource, no additional information should revoke it. A new information should only lead to grant additional privileges (Seamons et al., 2002). Policies define conditions that must be satisfied in order to disclose credentials to the other side. For our specification they can be divided into groups, such as access control policies, purpose policies and optionally privacy policies. Access control policies contain set of rules specifying whether access rights can be granted to entities in the means of security, e.g. which entities and how are authorised to access credentials. Access control policies can use various approaches, for example as the attribute-based access control ($ABAC$) (Winsborough and Li, 2002) or the role-based access control ($RBAC$) (Herzberg et al., 2000; Becker and Sewell, 2004). These can be easily implemented by the general attribute exchange protocols, such as the Extensible Application Markup Language ($XAML$) (MacVittie, 2006) or the Security Assertion Markup Language ($SAML$) (Hughes and Maler, 2005). Once the access is granted, credentials are transferred through messages. The negotiators must agree on the same protocol in order to communicate. The protocol specifies, how the messages are exchanged, which type of information they contain and their ordering (Yu et al., 2001; Winslett et al., 2002). It also specifies the formal way of the initialisation, process and termination of trust negotiation. During building trust each entity gains experience about the other one's attributes, which leads to trust or distrust and in increasing confidence about this opinion after time (Theodorakopoulos and Baras, 2004).

An engineer needs to follow a certain process in order to create a high-quality software that meets complex requirements of a customer. The Software Development Life Cycle represents a suitable approach to develop such software as it contains various models defining the development process, such as the waterfall model, incremental model, b-model, v-model, spiral model, wheel-and-spoke model and unified process model (Ruparelia, 2010). Regardless the model, the SDLC generally divides the development into particular phases, such as requirement analysis, design, implementation, testing and evolution. Rios et al. (2017) present a framework that addresses trust negotiation issues during the early phases, such as the requirement analysis. It is focused on finding the most suitable policies for a system by detecting conflicts between requirements on trust and privacy. The authors define a set of predicates and rules for specifying the trust and privacy policies and for describing the system dynamics. Driver et al. (2017) present a guide, how third-party components can be integrated into a new software using digital trust in the SDLC. Jones and Rastogi (2004) in their work present a way of the security integration into the SDLC. Another work dealing with security is the one from Noopur (2013) that overviews information about developing a secure software. Futcher and von Solms (2008) present a guidelines for a secure software development. The work of Dawson et al. (2010) proposes a methodology for the software assurance integration into the SDLC in order to secure the application layer. Apart from following the SDLC, the engineer should keep in mind the software security goals, such as confidentiality, integrity and availability (Sodiya et al., 2006). Confidentiality prevents an unauthorised disclosure of software resources, integrity prevents their unauthorised modification and availability prevents unauthorised denial of services of them. Generally they help to protect software and its resources, reduce the software design defects and improve its quality, which makes the software more secure at the end.

3. The UML-based Trust Negotiation Specification

In this section we propose a UML-based model specification that will facilitate trust negotiation between two entities. For scenarios, where more entities need to establish trust, this specification can be also helpful. In case of several entities, the required trust relations may be decomposed into a set of one to one relationships and trust

negotiation can be used for each of them. Another option would be to establish the relationships by a mixture of trust negotiation with different methods, such as trust transitivity or trust based on the reputation of an entity. However, our specification is focused on building trust between two entities and the rest is out of scope of this work. This specification includes all the needed resources to carry out this process and will serve developers as a guidance to implement trust negotiation into their software. To do so, we will include this specification into the SDLC to become a part of its design phase and as a result we will obtain the Secure Software Development Life Cycle that will be already trust-aware. Generally, the SDLC represents a traditional way of developing software in which the final product requirements are analysed and defined at the beginning. However, we believe that our proposed specification could be also applicable for agile development methodologies due to its modular, flexible and extensible design. Developers can utilize that parts of the framework that suit their actual needs for trust and later may change their decision. In this work, we refer to the negotiating entities as negotiators.

The model specification outlined with a UML-based notation is depicted in Figure 1. We decided to illustrate the figure as understandably as possible in a manner that the proposed components and their relations were clear to the reader. For this reason, we use a widely known Unified Modelling Language. The figure reflects our intended scenario with two negotiators, so some degree of redundancy was introduced. Our specification consists of components needed to carry out trust negotiation efficiently. The component *Negotiator* represents an entity willing to establish trust with the other one. There are two negotiators that are connected to the *Goal* component. It defines the common purpose for building trust.

The *Trust Relationship* component represents the trust relationship between the negotiators and it is connected through the *Trust Negotiation (TN)* component that represents the negotiator's part of trust negotiation. These components are connected to the *Negotiation Protocol* component that assures that both negotiators use the same negotiating protocol. The Negotiator component is connected to the *Compliance Checker* component. The compliance checkers are also connected together in order to provide feedback about the use of credentials.

Additionally, two other components are connected to them: the *Credentials / Declarations* component and the *Policy* component. They represent the credentials / declarations and policies of their owning entity. The last one is the *Trusted Authority* that is connected to Credentials / Declarations, as well as to the Negotiator that uses the trusted authority to issue and verify credentials and declarations.

3.1. Analysis of the Specification

We have illustrated the components of the specification and the connections between them. Now we will analyse its functionality.

Two identified and authorised negotiators are needed to carry out trust negotiation. They exchange credentials with each other and follow decisions given by a compliance checker in order to establish the desired trust relationship. It may be built one-way only or two-ways, depending on the requirements of the defined goal and on the agreement of both negotiators. Each negotiator has to specify the required trust level to be placed onto the other one. Also, it has to bear in mind the intended goal and adapt its policies accordingly. Once the built trust is equal or higher to the defined trust level, a trust relationship is established. Depending on the goal, the purpose policies define trust values that will be used for the evaluation of trust within the defined goal-related context. This context is also used for the evaluation of incoming credentials, whether they are context-relevant and whether they will be included in trust negotiation. A credential is assigned a purpose property that defines for what purpose the credential should be used. When a credential is received, the compliance checker compares its purpose against the goal-related context. When they match, the context-related trust value is calculated from the credential's weight and it is added to the overall trust level defining the trust relationship. This applies for both sides, each negotiator evaluates its own trust level independently. Further, each negotiator has to specify its access control policies that define conditions for granting access to its own resources. When the other negotiator requests a credential, it must have been previously identified and authorised and also has to provide additional information, such as the purpose of his request. The requester must agree that the received credential will be used only for trust establishment. Then, the compliance checker inputs all of these information from the requester, checks its own state information from the past and match them against the access control policies. As a result, a disclosure decision is made, whether the request will be satisfied and the credential will be disclosed. However, when a negotiator discloses credentials, reveals its private information that may compromise its privacy. For some negotiating scenarios this may not be an issue, however, our specification supports an optional privacy protection as it can provide an additional security. A negotiator can specify privacy policies that define its accepted maximal exposure level and conditions, under which its privacy can be exposed and to which extent.

The compliance checker helps the negotiators with disclosure decisions. In fact, it can proceed the whole trust negotiation on behalf of the negotiator. It would act as a trusted agent with delegated rights. This depends on the concrete implementation and on the privileges given to the compliance checker. During disclosure, it inputs the demanded credentials and makes a decision based on matching attributes against the defined policies, whilst it may focus on minimisation of disclosures, preservation of privacy or another defined behaviour. Compliance checkers can communicate to each other and exchange infor-

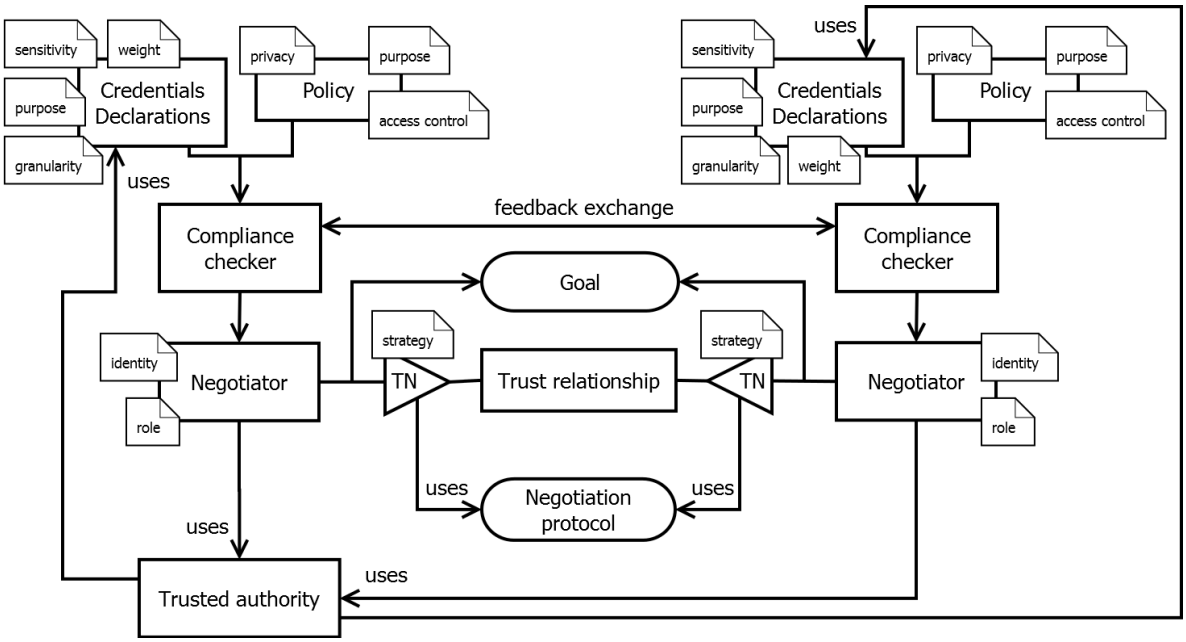

Figure 1: The UML-based model specification for trust negotiation

mation about the disclosed declarations and credentials, which provides them feedback that can be used for future disclosure decisions. Declarations are usually uncertified claims and contain information useful for trust negotiation. They can be certified by a certification authority to add more confidence. The authority can issue, sign and verify credentials, which can give the negotiators additional trust. Credentials are similar as declarations, just are signed. They are characterised by their sensitivity level specifying the contained information confidentiality, weight specifying their importance for building trust and granularity specifying the desired magnitude of information disclosure. For their exchange can be used one of the infinite number of negotiating strategies that significantly influence trust negotiation and its results. The strategy specifies the exact content of the exchanged messages and aims to establish a successful trust relationship. It also defines which credentials will be used, which ones will be prioritised and thus disclosed first, how trust values will be calculated from the incoming credentials and how trust negotiation will be terminated and evaluated. Each negotiator can define and use its own strategy as its primary objective can be different, such as a protection of his privacy or a quick trust establishment. However, the strategies used by both negotiators should be compatible to avoid misunderstanding, conflicts, failures or a privacy abuse. A good approach is to agree on the same strategy as well, but it is not inevitable. However, both negotiators must agree on the same negotiating protocol in order to be able to communicate.

3.2. Specification Matching against Trust Models

In this section we briefly outline the design and functionality of the mentioned trust management systems and trust negotiation models and make a suggestion how they can be instantiated from our proposed specification. The usual approach for trust negotiation models to establish trust is the definition of policies and their evaluation by a compliance checker. From the models we mentioned in Section 2, TrustBuilder and its enhanced version seem to be the most robust ones with a high degree of scalability and reconfigurability for trust negotiation. TrustBuilder and PROTUNE use security agents that carry out trust negotiation on behalf of entities. We chose these models as an example to be matched against our proposed specification representing a general trust negotiation concept that can be applied to the same or similar scenarios as these models use. The Negotiator component may act as a security agent, if the entity is willing to delegate its access rights and trust negotiation control to the system. Our specification can be instantiated by applying more detailed application-specific requirements on the intended scenario and by following its concept, such as components topology. The instantiation will make it usable for a concrete trust negotiation application.

TrustBuilder allows the negotiators to define their own negotiating strategy. They use so called local strategy that specifies the resources to be disclosed next and the conditions to continue or terminate the process. PROTUNE does not allow a specification of own strategy and instead provides a so called cooperative default strategy. All the resources are disclosed at each step that can be released and that seem to be valid to build trust. Winsborough et al. in their framework define two other strategies, such as the eager and the parsimonious one. The former is a simple strategy, where a negotiator discloses all his unlocked credentials. When these are received, locked credentials

become unlocked and will be disclosed too. This strategy is efficient as it does not restrict the exchange process, however, it does not protect privacy. The latter is more sophisticated and secure. It avoids redundant disclosures by using requests and aims to disclose only a minimal set of credentials that are really required. In our specification, a developer can define the most convenient strategy that suits his needs. He may also let the negotiators choose the strategy they prefer or he may restrict them to use only one given strategy. The strategy is defined by rules that, depending on scenario, can be implemented as "strategy policies" in a policy language. A combination of the strategy policies and the negotiator's policies, such as the privacy and access control ones, determines the behaviour of the negotiator.

4. The Model Specification Example

The UML-based model specification introduced in the previous section is useful to specify all the trust negotiation models. In this section we illustrate a suitable example of possible applications that is depicted in Figure 2. Let us assume a small company that outsources network services. The company offers a quality work and various services for a customer, such as a network planning and design. The company wants to sell its services, so it has to find potential customers. In our example, a customer that wants to make a network design for his offices, is considering the company offer.

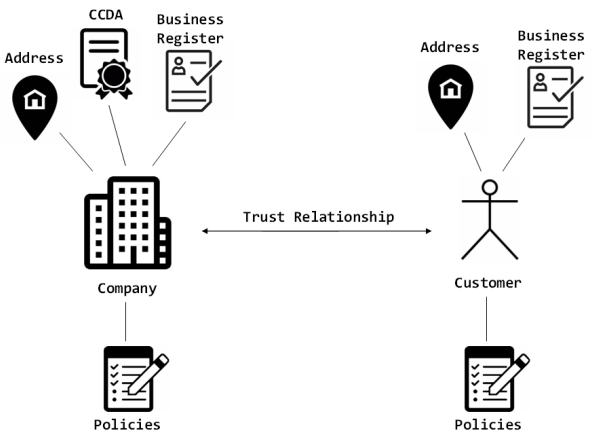

Figure 2: The example of our trust negotiation scenario

In our specification, the company will be represented by one negotiator and the customer by the other one. Trust must be established both ways in order to successfully negotiate. The company needs to make sure that the customer will pay for the provided services and the customer needs to make sure that the delivered solution will be of high quality and working well. The common goal is to make a successful business to satisfy both parties, which means that the company will create a complete network solution for the customer and will be paid for it. Each party plays a different role in order to accomplish this goal. The company plays a role as a networking expert and the customer plays a role as an ordering party.

4.1. Definition of Policies

Before trust negotiation can be carried out, the company and the customer must define their policies. They will specify goals and rules for accessing the negotiators' data in order to preserve their security and privacy. The company and the customer are recorded in the national business register that manages various information about its members for commercial purposes. When we are referring to the business register in this example, we mean records containing sensitive information about its owner, such as its trade name, business plan, identification number, legal form, etc. At first, the company has to define its objectives in purpose policies, such as selling its services:

- The company carries out a design and implementation of a complete network solution for a customer.
- The company carries out a maintenance of a network in the customer's area.
- The company defines a restrictive policy that it will offer its services only to a customer that has his own offices and that is located in a range of 100 km at most due to logistic issues.

Then, the company has to define policies that describe and condition the selling process itself, such as a specification of the price list:

- The network design costs 50€ per one man-hour.
- The network implementation costs 30€ per one man-hour.
- The maintenance of the network costs 20€ per each network equipment a month.
- If the customer agrees with a long-term maintenance contract and orders more than 200 hours in total, the company provides a discount of 20% of the future maintenance.

The company also requested the Cisco Systems, a trusted authority to issue the CCDA[1] certification stating its expertise in networks design, which should help the company to be more attractive and reliable to a customer. The company defines also its access control policies combined with privacy policies:

- The company business register as well as its CCDA certification is publicly available to anyone.

[1]The CCDA stands for the Cisco Certified Design Associate and certifies the knowledge and skills of a professional to design routed and switched Cisco converged network infrastructures and services.

- The price list of services is available to potential customers on demand.
- Payment information is protected and it will become available to a customer that has been contracted.

The customer is in a disadvantaged position against the company as he pays and may not be confident about the company deliveries. The issued certification may convince him, even when the customer does not require any certification to be possessed by the company. The customer must also specify his policies. At first, the purpose policies are defined:

- The customer's main objective is to purchase a custom-made network design and to implement it in his offices.
- The customer is willing to sign a long-term contract about maintenance of the new installed network equipment.
- The customer wants to negotiate with networking companies in order to obtain enough trust towards a potential contractor.
- The customer is decided to accept a common market price for the bought network services.

Just like the company, the customer needs to specify his access control policies that will regulate access to his credentials during trust negotiation:

- The customer is more worried about his privacy, so he does not disclose any information to public.
- The customer's business register can be only disclosed to an entity that has provided its register in return.
- The customer will disclose a brief information about his office infrastructure that are needed for trust negotiation, only to a potential contractor. Full-detailed information can be provided only to the approved contractor.

All the defined policies are summarised in Table 1.

4.2. Trust Negotiation

When all the policies are defined, the customer and the company may proceed with trust negotiation. We suppose that they have already agreed to do so based on their common goal of creating a network solution defined in their purpose policies. The customer has found a potential designer of his network and the company has found a potential client. Now they need to build mutual trust in order to sing a contract. They have to use the same protocol for communication, but this is more a technical issue and principally is not relevant for the result. More importantly, each has to choose its negotiating strategy. The company chooses an eager-like strategy as it wants to disclose all releasable credentials. It does not care much about its privacy since it wants to be as transparent to the customer as possible in order to appear trustworthy and reliable. The customer is more conservative, so he chooses a parsimonious-like strategy that covers his needs for privacy too. A more detailed view on how trust negotiation is carried out in this case is depicted in Figure 3, while a simplified, but complete version is depicted in Figure 4.

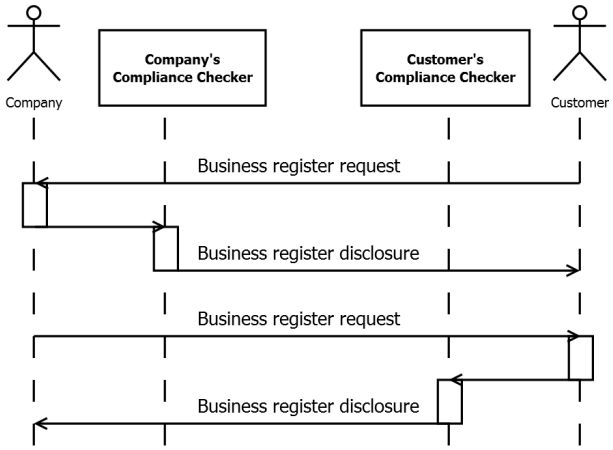

Figure 3: Trust Negotiation Requests Sequence

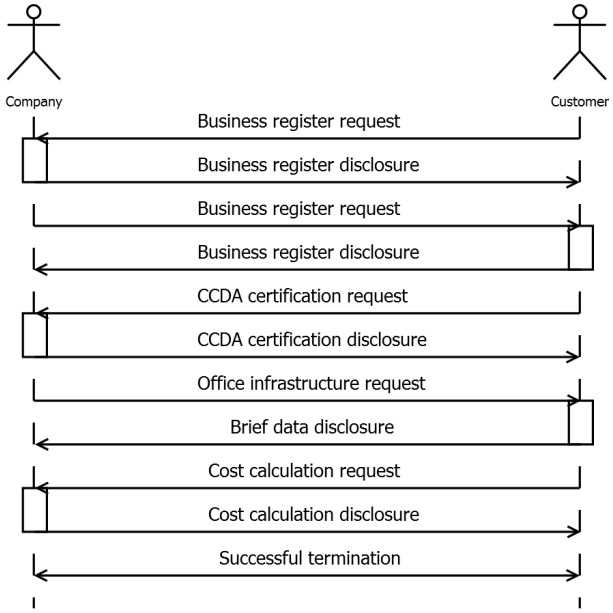

Figure 4: Simplified Trust Negotiation

The company and the customer are represented by two negotiators. Trust negotiation is initiated by the customer's request of business register to the company. The company passes this request to its compliance checker for checking the access control policies, whether the request can be satisfied. As the policies do not restrict access for

Table 1: Defined Policies of the Company and the Customer

Policies	
Company	**Customer**
Purpose Policies • To make a network solution design • To implement a network solution • To carry out network infrastructure maintenance	Purpose Policies • To purchase a network solution design • To purchase a network solution implementation • To purchase the network maintenance
Services Price List • A network design 50 € / man-hour • A network implementation 30 € / man-hour • A network maintenance 20 € / equipment monthly • At least 200 hours contracted: 20% discount for maintenance	Services Price List • Accept common market prices
Access Control Policies • Business register: free access • CCDA certification: free access • Price list: available on demand • Payment information: available based on contract	Access Control Policies • Business register: access for similar information in return • Brief infrastructure information: on demand • Full infrastrucure information: only the approved contractor
Other Policies • Requirement for owned offices • Requirement for maximum radius 100 km	

the business register, it is disclosed. The same action happens for the other side, when the company asks the customer for his register. The customer's compliance checker inspects its access control policies and the register is disclosed, since a similar information has been provided in the previous step from the company. At this point two credentials have been exchanged and trust has been increased in both negotiators. Then, the customer makes another request to the company and ask for the CCDA certification as he wants to identify the company's qualities. Policies assure free access, so the request is satisfied and builds additional trust in the customer. The company needs to know the customer's office infrastructure in order to propose the network solution and make a cost estimation, so requests the customer for such information. However, the customer has defined granularity restrictions for provisioning this information in his policies, where the level of details depends on the current relationship towards the company. As a result, the customer partially satisfies the request by provisioning a less-detailed infrastructure information than demanded with a note that more details can be provided later. As a last step, the customer makes a request to the company for obtaining a cost estimation. The company checks the provided information and matches them against its policies. The customer's offices are located in the range of 30 km, which satisfies the maximum range policy of 100 km. The company makes the following cost estimation: 200 man-hours (10.000€) for design of the network and 150 man-hours (4.500€) for the implementation. Since the order is more than 200 man-hours, the company offers a discount of 20% for a long-term maintenance. The customer is satisfied with this estimation and his trust to the company has increased sufficiently. He is ready to sign a contract with the company as all provided information are in compliant with his policy. The same applies for the company. More importantly, a strong-enough trust relationship was established between both parties, so trust negotiation was successful and is terminated. However, the exchange of information between both negotiators continue. This time not for building trust, but for reaching the common goal. Once the contract is signed, the customer can provide according to his policies all the necessary information about the infrastructure to the company.

5. Integration of the Trust Negotiation Specification into the SDLC

The Software Development Life Cycle (SDLC) is an overall name for various models covering the life cycle of software. It helps developers to create and maintain a complex high-quality software. By the life cycle is meant a series of stages in form and functional activity through which a software passes during its lifetime. Therefore, as the name suggests, the SDLC is composed of clearly defined phases, where a succeeding one follows the preceding one. The exact number and interpretation of the phases depends on the actual model used, however, generally we can identify the following ones: requirement analysis, design, implementation, testing and evolution.

These phases describe the exact actions and the order that developers have to follow, however, they lack support for decision-based systems, in particular, for trust negotiation. A common today's problem is that software is being developed the old-fashioned way and if trust-awareness is needed, it is included additionally into the software. It is an underestimation of the importance of the trust-related issues. We find them very important as they determine the software security and other factors. Putting trust support additionally is not a reliable way of doing it. We will address the trust-related issues and integrate our trust negotiation specification into the SDLC resulting in a creation of the Trust-Aware Software Development Life Cycle (TASDLC) depicted in Figure 5. Once it is done for all phases, developers will be guided to create a secure, reliable and trust-aware high-quality software. In this work

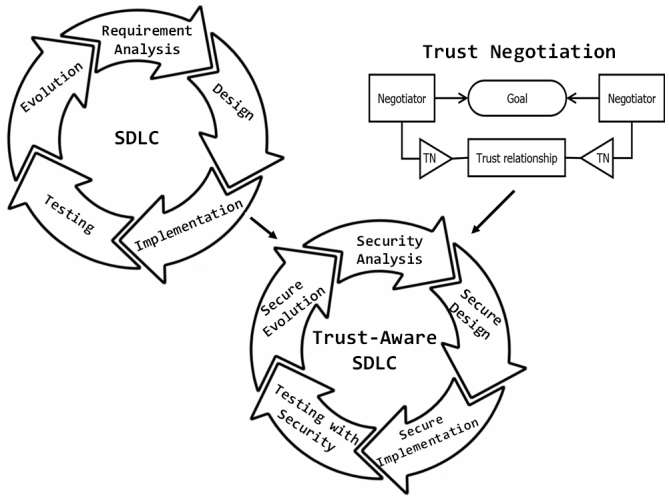

Figure 5: The Secure Software Development Life Cycle

we will focus on the integration of our proposed specification into the design phase of the SDLC.

5.1. Integration of the Specification into the Design Phase

The requirement analysis phase is the first one. It consists of tasks determining the needs and conditions that must be accomplished. We do not focus on the first phase now, however, it is also important for a secure and efficient software development (Rios et al., 2017). When all requirements are clearly defined and validated during the first phase, we can proceed to the design phase. It is focused on problem solving and finding solutions. During planning and designing, the layout is considered from more points of view, such as classifications based on granularity or diverse areas involved in design. For example, this phase can be divided into a low-level design involving the data structure, algorithm and low-level component design and into a high-level design involving the architectural, interface and high-level module design.

The software should be divided into modules that encapsulate a functional unit performing desired operations. They will have specified trust relationships between them, while they can be divided into the implicit and the established relationships. The former are given by the architectural design and are fixed. Actually, this trust was given to the module by the designer. If the trusted module is not behaving as expected, the impacts can affect the trustor. To prevent an unexpected or unwanted behaviour, designers have to consider possible module failures and design a fail-safe and fail-secure recovery functionality. A safer approach is to not trust the module unconditionally, but rather to check its verifiable functionality, such as outputs on a defined interface to verify their correctness. The established relationships are not initially given and will have to be built.

In this case, two different scenarios must be distinguished: The first one is establishing trust between modules themselves that may be strangers or they come from different parties, so their attributes must be inspected and confirmed first. Such module can represent a library coming from an unverified source. If the source code is available, programmers can make a code audit to verify its proper functionality and check for bugs or possible backdoors. A positive audit outcome can increase the placed trust onto the module. If the source code is proprietary and not available, designers carry out trust negotiation with the third party in order to establish trust to their library.

The other scenario supposes that the software modules are coherent, secure and reliable. There are no contradictions within the software itself and the modules trust each other. However, the purpose of the software will be building trust on demand between entities, where they will be instantiated from given modules. This scenario requires a direct application of a trust model into the designed trust-aware software. This is the case we are proposing in this section. Then, trust negotiation will be carried out during the software deployment.

Software Conceptual Model

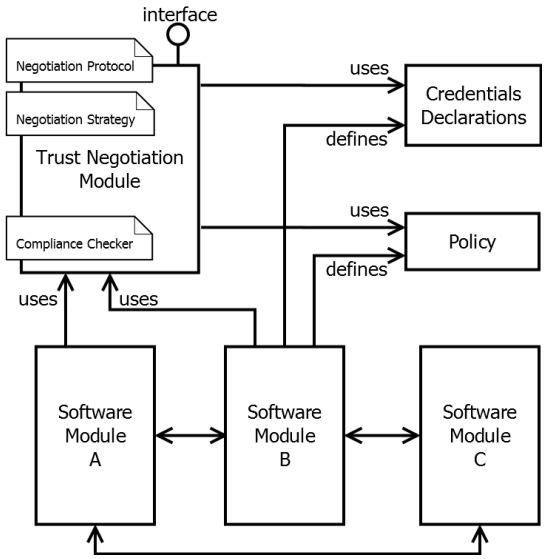

Figure 6: Integration of Trust Negotiation into a Software Conceptual Model

The concept of the integration of trust negotiation into software during the design phase is illustrated in Figure 6. The usual way of designing a quality software is to divide it into modules, where each one of them encapsulates a specific functionality and communicates with the other modules or users through a defined interface. For illustration, we depicted three software modules in Figure 6, namely the *Software Module A, B* and *C* that represent a given functionality specified by the software designer. The exact topology of the modules can be arbitrary and depends on the software design and the concrete implemen-

tation. The developer should follow the standard software development methodologies in order to design secure and extensible modules. He can choose any suitable development model from the SDLC. Apart from them, we include another one named *Trust Negotiation Module* that integrates trust negotiation into the software. It connects to the other modules and it's purpose is to establish trust relationship with a desired party.

The trust negotiation module can remotely connect through a defined interface to an external entity, such as a user, system or another trust negotiation module in order to exchange credentials needed for building trust. This module encapsulates all the sub-components needed to carry out trust negotiation, such as the negotiating protocol, the negotiating strategy and the compliance checker. *Credentials*, *Declarations* and *Policies* represent a user and application data that are stored separately from the module in a data storage. Physically they can be maintained in a database. The software modules access them through secure interfaces, while their data structure and access restrictions are given by design. The trust negotiation module will not modify any existing data as it will be accessed only to read policies and send credentials to the other entity. The received ones will be saved into the storage place too. Depending on the trust negotiation scenario and the use-case of credentials, some of them can be stored only temporarily to build trust, while the others may be preserved for other purposes. The credentials, declarations and policies are specified by their owning entity through software modules that are entitled to do so, such as in our case the Software Module B.

Software Component Layout

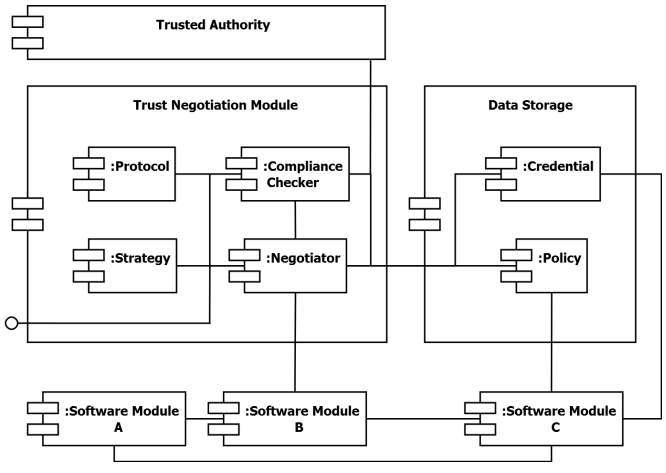

Figure 7: Integration of Trust Negotiation into a Software Component Layout

Figure 7 depicts a more detailed view, how the modules can be organised into particular components. The software modules A, B and C are not decomposed, since their design and inner structure are unknown. The trust negotiation module consists of the *Protocol*, *Strategy*, *Compliance Checker* and *Negotiator* components. The Negotiator component carries out trust negotiation with the external entity through the defined interface. It can quickly identify the entity by possessing some of its information. The protocol component maintains the defined negotiating protocol and similarly, the strategy component maintains the chosen negotiating strategy. They will be used by the negotiator to perform trust negotiation. The compliance checker component demands credentials and policies from the *Data Storage* module. Based on them and on the chosen strategy this component makes disclosure decisions and sends the result to the negotiator in order to control the credentials disclosures. Also, the compliance checker may query the *Trusted Authority* module, if present, to verify incoming credentials or sign new ones.

The developer can design each component a class. These classes will be connected to each other through the association relationship and will communicate by sending messages through the members function calls. However, the trust negotiation module should be isolated, so its interfaces can be designed as exported functions from the classes. The module can be designed also as a standalone library that is dynamically linked to the other modules and that encapsulates and thus protects its inner functionality well.

6. The Model Specification Comparison

To the best of our knowledge, our model specification integration into the SDLC is unique as it was not proposed before. The model specification represents a general trust framework that serves for the inclusion of all trust negotiation models into the SDLC. Apart from our work, Lara (2015) presents an engineering framework dealing with trust. The author deals with the secure system engineering and trust management and combines them together. He proposes a general framework for the inclusion of trust evaluation models. Additionally, Driver et al. (2017) present a similar work dealing with trust in the SDLC. The authors claim that the majority of code in new software comes from external third-party channels. They analyse these channels according to their trustworthiness and a potential risk they may introduce to the new software. They do not focus on creating a software that will handle trust in its application, but rather on creating a trustworthy software that will minimise risks by introducing a reliable third-party components.

Other works deal with security that is a topic closely related to trust, since security can be considered as the essential requirement for building trust. Jones and Rastogi (2004) present a work that deals with secure coding practices and building security. They focus on the creation of the secure SDLC, which means that they analyse each phase of the SDLC in order to recognize their security issues and to propose a suitable solution. The security context is added to each phase and is important in

order to create a trustworthy and reliable software. Another work dealing with security is the one from Noopur (2013). He presented a survey that overviews methods for the secure software development in existing processes, standards, life-cycle models, frameworks, etc. This work is suitable to acquire a basic knowledge about the topic and the related terminology. Yet another work dealing with the secure software development is the one from Futcher and von Solms (2008). The authors describe security standards and practices that simplify the implementation of security controls. They guide software engineers and developers in the process of including and solving security issues in their software. Dawson et al. (2010) also analyse in their work secure coding practices and their integration into the SDLC. In addition, a methodology is proposed for integrating software assurance into the Department of Defence Information Assurance Certification & Accreditation Process *(DIACAP)*. The software assurance integration facilitates to secure the application layer, where the majority of the system vulnerabilities are located, as the authors claim. Sodiya et al. (2006) present a work that focus on secure software development by avoiding security breaches and design defects. They introduce a unified model, named Secure Software Development Model *(SSDM)* that combines the security engineering and the software engineering. It focuses on the effective development of secure software products.

Our model specification seems to be the only one work that deals with the problem of including trust negotiation into the SDLC. The other works solve different trust or security issues.

7. Validation of the Model Specification

Our proposed specification is intended as a guidance for creating a customised model of trust negotiation, where two entities aim to establish a trust relationship. We will validate it by implementing a specific trust negotiation scenario similar to the one presented in Section 4. Two entities from a commercial area want to make a business together and need to find out, whether they can trust each other.

In order to do so, we will design a complete scenario with defined goals, credentials and policies. We will design appropriate software modules as described in Section 5, their layout and interfaces between them. The core of the software is represented by a trust negotiation module. Each entity owns one its instance that will carry out trust negotiation on behalf of the entity. Other modules would be a control module for supervising trust negotiation and an end-user interface module for communicating with a user. Once this proposal is done, we will design classes belonging to the modules and define their methods and attributes. For example, the trust negotiation module will contain the Negotiator and ComplianceChecker classes, where the former includes methods for exchanging credentials and the latter for their supervising and providing a feedback. Then, we will implement the classes and fill data structures with the defined data, such as the credentials and policies. We will also define several negotiating strategies for each entity. Finally, we will launch a set of trust negotiations, each with a different configuration in terms of the defined policies and used strategy. This way, we will confirm the validity of our proposed specification and will be able to collect and evaluate results of the implemented trust negotiation scenario.

8. Conclusion

In this work we presented a UML-based model specification that facilitates the implementation of trust negotiation for two entities willing to build a trust relationship. The aim of it is to help developers to solve trust-related issues and to guide them to include trust negotiation into their software in each phase of development in order to make it more secure and reliable. Then, we illustrated a suitable example, how this specification can be applied in a real scenario in order to build trust between unknown entities. The specification due to its generality can be further extended in case of need. We also described how the specification could be integrated into the design phase of the Software Development Life Cycle (SDLC). It was designed with the consideration to represent the design phase of the SDLC and later to continue with the others. We compared our proposed model specification against the other existing methods.

In the future work we will include our specification in the implementation phase and we will validate it in the manner described in Section 7. Later, we will cover all the phases of the SDLC and our specification will become a part of a trust negotiation framework. Then we will validate the whole framework in a real software implementation. Additionally, the specification can be analysed and evaluated for its use with agile methodologies that are currently more widely used for software development. They emerge from the traditional approaches, such as the SDLC, so a research in this area is very suitable.

Acknowledgements

This research has been supported by the European project "European Network for Cyber-security (NECS)" - the European Unions Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 675320 and by the Spanish Ministry of Economy and Competitiveness through the SMOG (TIN2016-79095-C2-1-R) project.

References

Becker, M.Y., Sewell, P., 2004. Cassandra: Distributed access control policies with tunable expressiveness, in: 5th IEEE International Workshop on Policies for Distributed Systems and Networks

(POLICY 2004), 7-9 June 2004, Yorktown Heights, NY, USA, pp. 159–168.

Blaze, M., Feigenbaum, J., Ioannidis, J., Keromytis, A.D., 1999. The keynote trust-management system version 2. RFC 2704, 1–37.

Blaze, M., Feigenbaum, J., Lacy, J., 1996. Decentralized trust management, in: Proceedings of the 1996 IEEE Symposium on Security and Privacy, IEEE Computer Society, Washington, DC, USA. pp. 164–.

Blaze, M., Feigenbaum, J., Strauss, M., 1998. Compliance checking in the policymaker trust management system, in: Financial Cryptography, Second International Conference, FC'98, Anguilla, British West Indies, February 23-25, 1998, Proceedings, pp. 254–274.

Bonatti, P.A., Coi, J.L.D., Olmedilla, D., Sauro, L., 2010. A rule-based trust negotiation system. IEEE Trans. Knowl. Data Eng. 22, 1507–1520.

Corritore, C.L., Kracher, B., Wiedenbeck, S., 2003. On-line trust: Concepts, evolving themes, a model. International Journal of Human-Computer Studies 58, 737 – 758. Trust and Technology.

Dawson, M., Burrell, D.N., Rahim, E., Brewster, S., 2010. Integrating software assurance into the software development life cycle (sdlc). Journal of Information Systems Technology & Planning 3, 49–53.

Driver, M., Gaehtgens, F., O'Neill, M., 2017. Managing digital trust in the software development life cycle ID G00326944.

Falcone, R., Castelfranchi, C., 2001. Social trust: A cognitive approach, in: Trust and Deception in Virtual Societies. Springer, pp. 55–90.

Futcher, L., von Solms, R., 2008. Guidelines for secure software development, in: Proceedings of the 2008 Annual Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries, SAICSIT 2008, Wilderness, South Africa, October 6-8, 2008, pp. 56–65.

Gambetta, D., et al., 2000. Can we trust trust. Trust: Making and Breaking Cooperative Relations 13, 213–237.

Herzberg, A., Mass, Y., Mihaeli, J., Naor, D., Ravid, Y., 2000. Access control meets public key infrastructure, or: Assigning roles to strangers, in: 2000 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 14-17, 2000, pp. 2–14.

Hughes, J., Maler, E., 2005. Security assertion markup language (saml) v2.0 technical overview. OASIS SSTC Working Draft sstc-saml-tech-overview-2.0-draft-08 , 29–38.

Jones, R.L., Rastogi, A., 2004. Secure coding: Building security into the software development life cycle. Information Systems Security 13, 29–39.

Jøsang, A., Ismail, R., Boyd, C., 2007. A survey of trust and reputation systems for online service provision. Decision Support Systems 43, 618–644.

Lara, F.M., 2015. Trust Engineering Framework for Software Services. Ph.D. thesis. Universidad de Málaga.

Lee, A.J., Winslett, M., Perano, K.J., 2009. Trustbuilder2: A reconfigurable framework for trust negotiation, in: Trust Management III, Third IFIP WG 11.11 International Conference, IFIPTM 2009 , West Lafayette, IN, USA, June 15-19, 2009. Proceedings, pp. 176–195.

MacVittie, L.A., 2006. XAML - in a Nutshell: A Desktop Quick Reference. O'Reilly.

Mármol, F.G., Pérez, G.M., 2009. Security threats scenarios in trust and reputation models for distributed systems. Computers & Security 28, 545–556.

Noopur, D., 2013. Secure software development life cycle processes. Software Engineering Institute, Carnegie Mellon University, (2013). .

Rios, R., Fernandez-Gago, C., Lopez, J., 2017. Modelling privacy-aware trust negotiations. Computers & Security .

Ruparelia, N.B., 2010. Software development lifecycle models. SIGSOFT Softw. Eng. Notes 35, 8–13.

Saied, Y.B., Olivereau, A., Zeghlache, D., Laurent, M., 2013. Trust management system design for the internet of things: A context-aware and multi-service approach. Computers & Security 39, 351–365.

Seamons, K.E., Winslett, M., Yu, T., Smith, B., Child, E., Jacobson, J., Mills, H., Yu, L., 2002. Requirements for policy languages for trust negotiation, in: 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY 2002), 5-7 June 2002, Monterey, CA, USA, pp. 68–79.

Sodiya, A.S., Onashoga, S.A., Ajayĩ, O., 2006. Towards building secure software systems. Issues in Informing Science & Information Technology 3.

Theodorakopoulos, G., Baras, J.S., 2004. Trust evaluation in ad-hoc networks, in: Proceedings of the 3rd ACM Workshop on Wireless Security, ACM, New York, NY, USA. pp. 1–10.

Tsiakis, T., Sthephanides, G., 2005. The concept of security and trust in electronic payments. Comput. Secur. 24, 10–15.

Wang, Y.D., Emurian, H.H., 2005. An overview of online trust: Concepts, elements, and implications. Computers in Human Behavior 21, 105–125.

Winsborough, W.H., Li, N., 2002. Towards practical automated trust negotiation, in: 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY 2002), 5-7 June 2002, Monterey, CA, USA, pp. 92–103.

Winsborough, W.H., Seamons, K.E., Jones, V.E., 2000. Automated trust negotiation, in: DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings, IEEE. pp. 88–102.

Winslett, M., Yu, T., Seamons, K.E., Hess, A., Jacobson, J., Jarvis, R., Smith, B., Yu, L., 2002. Negotiating trust in the web. IEEE Internet Computing 6, 30–37.

Yu, T., Winslett, M., Seamons, K.E., 2001. Interoperable strategies in automated trust negotiation, in: CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6-8, 2001., pp. 146–155.