Modelling Privacy-Aware Trust Negotiations

Ruben Rios[a,*], Carmen Fernandez-Gago[a], Javier Lopez[a]

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

Abstract

Trust negotiations are mechanisms that enable interaction between previously unknown users. After exchanging various pieces of potentially sensitive information, the participants of a negotiation can decide whether or not to trust one another. Therefore, trust negotiations bring about threats to personal privacy if not carefully considered. This paper presents a framework for representing trust negotiations in the early phases of the Software Development Life Cycle (SDLC). The framework can help software engineers to determine the most suitable policies for the system by detecting conflicts between privacy and trust requirements. More precisely, we extend the SI* modelling language and provide a set of predicates for defining trust and privacy policies and a set of rules for describing the dynamics of the system based on the established policies. The formal representation of the model facilitates its automatic verification. The framework has been validated in a distributed social network scenario for connecting drivers with potential passengers willing to share a journey.

Keywords: Secure Software Engineering, Requirements Engineering, Goal-Oriented Modelling, Trust, Privacy, Policy.

1. Introduction

The development of software systems has traditionally been guided by the fulfilment of functional requirements whereas other important aspects, especially security-related requirements, have been left aside and only considered after the functional requirements were completely satisfied. Fortunately, the trend is changing and nowadays both academia and industry are supporting the use of Secure Software Engineering initiatives (Stavropoulos, 2011), which consider security from the early phases of the Software Development Life Cycle (SDLC), being the requirements phase one of these early phases.

The requirements engineering phase (Mellado et al., 2010) is an important part of the SDLC as subsequent phases of the SDLC build on top of the outcomes of it. At this stage, the software engineer identifies the stakeholders, their goals and the requirements of the system while other low level decisions are obviated. A common approach to requirements engineering is to describe socio-technical systems in terms of the goals pursued by the various actors of the system rather than on programming concepts. Some of these goal-oriented methodologies (Horkoff and Yu, 2013) have started to incorporate security concepts (van Lamsweerde, 2004; Giorgini et al., 2005a), such as delegation or authorisation. However, and despite the importance of security-related concepts such as privacy and trust, requirements engineering methodologies only consider them on a very shallow level (Paja et al., 2015). Often, these methodologies do not reflect all the complexities and the relationships that arise when modelling these intertwined concepts. Typically, trust is based on the availability of information about another entity while privacy is about retaining control over personal data. The more information available, the more accurate and informed are the decisions on the risk associated with establishing a trust relationship. However, this means more information is disclosed, which raises privacy concerns. Modelling these relationships and detecting conflicts between them during the requirements engineering phase is precisely the main goal of this paper.

This paper presents a framework that enables the automatic verification of system models involving trust negotiations[1]. The framework provides requirements engineers with a tool for reasoning about suitable system policies, although they can be changed at run-time by the users. The proposed framework can be regarded as an extension of the SI* modelling language (Massacci et al., 2010) and is capable of identifying trust and privacy conflicts at design time and thus facilitating the tasks of the software engineer. This privacy-aware trust negotiation framework allows the policies for trusting other entities to be graphically defined together with the privacy preferences and sensitivity levels of the resources owned by each of the ac-

*Corresponding author
Email addresses: ruben@lcc.uma.es (Ruben Rios), mcgago@lcc.uma.es (Carmen Fernandez-Gago), jlm@lcc.uma.es (Javier Lopez)

[1]A preliminary version of this paper appeared in (Rios et al., 2016).

tors of the system. The graphical representation of the model can be then automatically translated into a set of formal predicates that, together with a set of rules that define the behaviour of the system, leads to derive stable models and allows conclusions to be drawn.

The rest of this paper is organised as follows. Section 2 introduces the related work in the field while Section 3 provides an overview of the SI* modelling language and presents some extensions that are relevant to our solution. Section 4 describes the various components of our trust negotiation extension paying particular attention to the definition of privacy policies and how privacy degrades with the exchange of information between actors during the negotiation. The model is formalised in Section 5 and then evaluated in a distributed social network scenario in Section 6. A brief discussion about the suitability of the framework is provided in Section 7. Finally, Section 8 presents the conclusions of the paper and outlines some potential lines of future work.

2. Related Work

The design of methods and tools for requirements engineering is a key topic of research and essential for the SDLC, goal-oriented methodologies being a common approach to modelling software systems.

The KAOS framework (van Lamsweerde; R. Darimont ; E. Letier, 1998) is a goal-oriented approach based on temporal logics that includes techniques for resolving inconsistencies or goal conflicts. Tropos (Castro et al., 2005) is another methodology founded on the i* organisational modelling framework (Yu, 1996). i* offers the notions of actor, goal and (actor) dependency. Originally, these frameworks did not take security requirements into consideration but were later extended to consider some security notions. On the one hand, KAOS has been extended to deal with security analysis by considering the notions of obstacle (van Lamsweerde and Letier, 2000) and the notion of anti-goal (van Lamsweerde, 2004). On the other hand, Secure Tropos (Mouratidis and GiorginiI, 2007) extends the Tropos methodology by making explicit who is the owner of a service, who is capable of providing a service and who is entitled to do so. The modelling language for producing diagrams during the requirements acquisition phase in Secure Tropos is SI* (Giorgini et al., 2005a; Massacci et al., 2010). SI* includes a number of security notions, including trust of execution and delegation. Several extensions have been proposed for SI* so as to refine it or to use the extensions to deal with risk or to detect threats (Paci et al., 2013). Our work is also based on SI*. Other goal-oriented methodologies use different modelling languages. For example, STS-ml (Paja et al., 2015) is similar to SI* but places more emphasis on authorisation relationships and introduces the notion of document beyond asset and resource.

Although an extensive body of research exists on security engineering, privacy has traditionally been left out from all the aforementioned frameworks and their corresponding extensions. The only support for privacy in most of these frameworks, including SI* and STS-ml, is in the narrow sense of privacy as data confidentiality and not as data disclosure control (Paja et al., 2015). Privacy issues are tackled in (Notario et al., 2015) by defining a set of best practices at different stages of the SDLC. LINDDUN (Deng et al., 2011) is a privacy engineering method that considers how threats influence privacy requirements and maps threats to software components. Then, threats are used to elicit privacy requirements. Pris (Kalloniatis et al., 2008) models privacy requirements as organisational goals and later privacy patterns are used to identify architectures.

In (Bonatti et al., 2010) the authors introduced a rule-based trust negotiation called PROTUNE that as well as ours use the definition of rules for deriving the negotiation strategy. However, it does not deep into the problem of identifying privacy conflicts. The work that is closest to ours is PP-Trust-$\mathcal{X}$ (Squicciarini et al., 2007), which extends a framework for trust negotiations called Trust-$\mathcal{X}$ (Bertino et al., 2004) with a privacy agreement subphase intended for exchanging privacy policies on the data to be negotiated. The goal of this additional phase is to allow each of the participants to learn about the privacy practices and preferences of their counterpart before releasing their credentials. The main difference with our proposal is that our framework enables detecting privacy conflicts during the requirements engineering phase rather than at runtime. To the best of our knowledge no other works address privacy preserving trust negotiations in the early phases of the SDLC.

Note, that our framework is meant to be a useful tool for the developers of a system that includes trust. Although it includes mechanisms for defining privacy policies, this is left to the designers and developers, in contrast to other works on privacy policies, which are user-centric (McDonald and Cranor, 2008; McDonald et al., 2009; Milne and Culnan, 2004) and focus on strategies for allowing users to compare system privacy policies with their own preferences.

Finally, we want to highlight the differences of this work with a preliminary version of our framework that appeared in (Rios et al., 2016). The current version introduces a significant improvement over the former as it enables the modelling of trust negotiations where the actors can exchange multiple informational assets with various entities. Although this situation is much more realistic, it was not covered in the previous version since it adds complexity to the modelling. It requires to consider the way in which privacy degrades as the entity releases their data to one or more entities. Moreover, this paper includes the validation of the framework with a distributed social network scenario.

3. The SI* Modelling Language

This section provides an overview of the SI* modelling language. First, we describe the methodology and provide some insight into its functionality. Then, we present the core elements of the language and their formalisation. Finally, we introduce some extensions to the language that are relevant to our framework.

*3.1. Introduction to SI**

SI* (Massacci et al., 2010) is a goal-oriented modelling language that was proposed to capture the functional and security requirements of socio-technical systems, that is, systems that take into account social interactions. For that reason, the SI* modelling language is an attractive tool when dealing with security notions such as delegation and trust. There are other predicates in SI* that denote social relationships, such as supervision, dependency or distrust, but we have not included the details of them in this paper as we do not use them for our purpose. Moreover, this language is the basis for Secure Tropos (Mouratidis and GiorginiI, 2007), a requirements engineering methodology that, besides modelling, provides model instantiation, model verification and support for the application of security patterns.

Graphical SI* models can be translated into Answer Set Programming (ASP) specifications to enable the analysis and validation of the models. ASP (Brewka et al., 2011) is a form of declarative programming language, which is well suited for solving hard computational problems. This is done by expressing a problem as a set of facts and rules, which are inputted into an inference engine to derive new facts and eventually reach a solution to the problem. Therefore, SI* models are encoded as a set of facts while the semantics are expressed by means of rules (or axioms) using Horn clauses. Then, the specification is fed into a system like DLV (Alviano et al., 2011) for detecting inconsistencies in the model. This is exactly the approach we follow in this paper.

3.2. Core Elements

The SI* modelling language defines some fundamental concepts that are necessary to represent socio-technical systems, including the actors[2], their goals and entitlements, and the relationships between them. An *agent* is an actual entity of the system that plays an active role in it. Thus, an agent of the system may be a particular individual, a specific organisation or even a software agent. A *role* represents the behaviour of an agent within the system. This enables modelling situations in which a particular agent has various roles and to characterise how the actor is expected to behave depending on its role.

Table 1: Core ASP predicates in SI*

Goal model
actor(Actor: a)
agent(Agent: a)
role(Role: r)
service(Service: s)
goal(Goal: g)
task(Task: t)
resource(Resource: r)
Actor properties
play(Agent: a, Role: r)
own(Actor: a, Service: s)
request(Actor: a, Service: s)
provide(Actor: a, Service: s)
Goal refinement
subgoal(Service: s_1, Service: s_2)
AND_decomp(Service: s, Service: s_1, Service: s_2)
OR_decomp(Service: s, Service: s_1, Service: s_2)
means_end(Service: s_1, Service: s_2)
Social relations
del_perm(Actor: a_1, Actor: a_2, Service: s)
del_exec(Actor: a_1, Actor: a_2, Service: s)
trust_perm(Actor: a_1, Actor: a_2, Service: s)
trust_exec(Actor: a_1, Actor: a_2, Service: s)

The language also includes the *play* relationship to indicate that an agent plays a particular role. It also defines some other relationships between roles but these are not described here since they are not relevant for our purposes.

The language also includes the notion of *service*, which is a generalisation of the goal, task and resource concepts. A *goal* is a desirable situation or interest expressed by an entity, while a *task* is a set of actions that can be executed as a means to fulfil a goal. Finally, a *resource* is a physical or informational artefact, which is produced or used by another goal or task. Connecting services to actors is done via three relationships: *own* denotes the authority of entities over resources and goals; *provide* represents the ability of an actor to accomplish a goal or to provide a resource; and *request* denotes the interest of an entity over a goal or resource. In the graphical representation of the model, these relationships are depicted with arcs labeled with O, R, and P, respectively. There are some additional relationships to denote that a goal can be achieved by satisfying a set of subgoals. In the case that the goal can only be fulfilled by satisfying all of its subgoals, this is represented by using an *AND_decomposition* relation, but if a single subgoal is sufficient to reach the goal, then this is represented by means of an *OR_decomposition* relation. A *means_end* relation is also introduced to identify goals that are necessary to achieve another goal or resources that are used or produced by a goal. Although these goal refinement relationships are core to SI* they are only described here for the sake of completeness.

As SI* is intended for socio-technical systems, the language deals with social relationships such as delegation and trust. These notions have security implications as trusting other entities or delegating permission over them

[2]The notion of *actor* is inherited from i*. Although this notion is no longer part of the language, it is used as a generalisation of the concepts of agent and role.

for achieving a goal pose some risk. The language can be used to represent the transfer of responsibility (*execution delegation*) and authority (*permission delegation*) from one actor to another to achieve a goal or to provide a resource. These relationships are labelled De and Dp, respectively. Similarly, it is possible to model the expectations that one actor has of another actor regarding his capabilities to perform a task (*trust execution*) or with respect to his behaviour concerning a given permission (*trust permission*). Trust relationships are labelled Te and Tp, respectively.

The formalisation of the aforementioned elements is done by using ASP predicates. Table 1 provides a summary of the most relevant predicates. These are only a subset of the predicates available in SI*. Also, note that these are extensional predicates that are used to translate graphical diagrams in formal specification. There are also some intensional predicates, which are necessary to model the actual willingness to achieve services, to provide them, and so on. These predicates are the result of applying rules and we will not describe them here.

3.3. SI Extensions*

Despite the expressiveness of the language, some authors have proposed extensions in order to support the modelling of complex scenarios. Some of these extensions are useful to our framework, which will be described in Section 4.

The first of the extensions, which are relevant for our purposes is due to Asnar et al. (Asnar et al., 2011), who extended SI* to introduce different levels of permissions on resources and the relationships between them to enable the identification of threats at the organisational level. In general, the resources of an information system can be characterised by means of three types of relationships that identify their composition, allocation and dependency. A *stored_in* relationship identifies that an informational resource is allocated in a physical resource. The *part_of* predicate denotes a compositional relationship, which denotes that a resource is composed of several other resources. The *require* relationship indicates that a resource needs another resource to function. Moreover, resources are marked with a *security requirement* label that indicates the security property (confidentiality, integrity and availability) that must hold for it. The security requirement is also related to the *permission type* granted to agents on resources. Actors can be provided with three different types of permissions on resources, namely, access, modify or manage permission. As such, a permission type might lead to a violation of a security property if the agent misuses its permission on a resource. This situation is covered with the *threat* predicate that holds when an agent poses a threat to a particular security property on a resource.

Later, Paci et al. (Paci et al., 2013) introduced two additional extensions to the language (an asset model and a trust model) to identify insider threats. An *asset* is a service with a particular *sensitivity level* that has a *security property* associated with it. These labels identify the level of protection for the resource demanded by its owner. This model distinguishes two types of assets. Direct assets are those for which a security property and sensitivity level are modelled explicitly, while indirect assets are services for which these values are derived from rules based on their relationships with other entities. The *trust* model proposed by the authors allows to specify the trust level[3] that an actor places in another actor with respect to a given permission for a particular asset. Similarly to what is proposed in the asset model, a series of rules are defined to derive indirect trust values. Also, a set of rules is introduced to resolve potential conflicts caused by applying derivation rules and to transform numerical trust values into ordinal trust categories.

Table 2: Predicates in SI* Extensions

Resource model
stored_in(Resource: r, Resource: r_1)
part_of(Resource: r, Resource: r_1)
require(Resource: r, Resource: r_1)
Permission model
permission(Actor: a, Resource: r, PType: pt)
del_perm(Actor: a, Actor: a_1, Resource: r, PType: pt)
trust_perm(Actor: a, Actor: a_1, Resource: r)
Security and Threat model
secure_req(Resource: r, SProperty: sp)
secure_req(Goal: g, SProperty: sp, Resource: r)
threat(Actor: a, Resource: r, SProperty: sp)
threat(Actor: a, Goal: g, SProperty: sp, Resource: r)
Asset model
asset(Service: s, Actor: a)
sensitivity(Service: s, SLevel: sl, Actor: a)
secure_req(Service: s, SProperty: sp, Actor: a)
Trust model
trust_perm(Actor: a, Actor: a_1, Service: s, PType: pt)

A summary of the predicates proposed by the aforementioned extensions is presented in Table 2. Again, for the sake of simplicity, the table only covers extensional predicates. The interested reader is referred to the original papers for the definition of intensional predicates and derivation rules. Also, note that we have slightly adapted some of these predicates in order to follow a common notation throughout the paper.

4. Trust Negotiation Extension

This section presents the main elements of our framework, which allows the incorporation of trust negotiations into the definition of socio-technical systems. First, we provide a general overview and then concentrate on each of its elements, namely, the trust negotiation relationship and the informational resources that are exchanged during

[3]In the original SI* language, trust was a binary relationship; an actor was either trusted or distrusted.

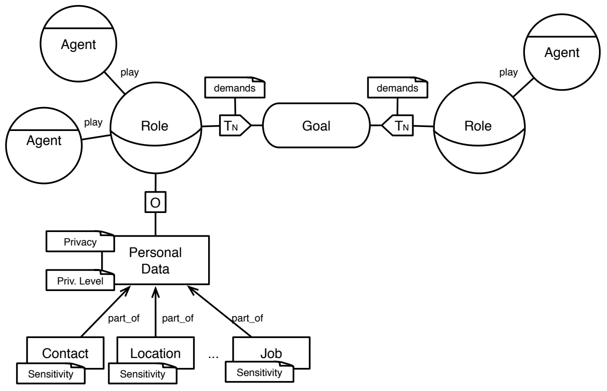
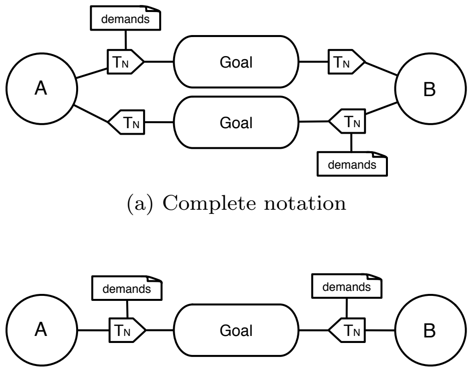


Figure 1: Trust Negotiation Representation in SI*



(a) Complete notation

(b) Simplified notation

Figure 2: Graphical notation for trust negotiations

the trust negotiation process. Since the exchange of informational assets is core to trust negotiations, we provide details on the privacy policy definition and the privacy exposure due to the release of multiple data resources.

4.1. Overview

A trust negotiation (Lee et al., 2009; Winslett et al., 2002) is a process in which various entities exchange (accredited) information in order to establish a base of trust as a means to achieve a goal. Therefore, a trust negotiation is a *dual relationship* in which the participants demand and offer information to one another and, as such, information about the participants is revealed.

This process can be modelled during the requirements engineering phase as shown in Figure 1. Note that at this stage it is not relevant to decide on the underlying mechanisms used for exchanging data or the format of the data itself. Thus, our framework abstracts from these issues and concentrates on transactions at a high level. Negotiation strategies, data exchange protocols and other relevant decisions can be made later, at design time.

The figure presents two main components that play a fundamental role in the modelling of trust negotiations. First, the dual relationship that considers the data being demanded by each of the agents and the goal to be accomplished. Second, it contemplates the informational resources (i.e., data) owned by the entities, which are important to establish the base of trust but at the same time need to be under control. For that reason, these are marked with a privacy requirement label and a sensitivity level to indicate the risk of sharing these data. Additionally, the sensitivity label is associated with the level of detail that an entity is willing to reveal about this informational resource.

This representation is based on existing features of the SI* modelling language and some existing extensions, as shown in Section 3. Next, we provide a detailed description of the elements that are necessary to model trust negotiations during the requirements engineering phase at the beginning of the SDLC.

4.2. Trust Negotiaton Relationship

There is a natural trade-off between privacy and trust in trust negotiation models due to their conflicting objectives. On the one hand, trust is founded on the availability of information about other entities and, for that reason, the parties involved in trust negotiations request information to their counterparts. The more information an entity has about another entity, the more accurate the decisions made will be about the trustworthiness of this entity. On the other hand, privacy refers to the ability to keep control of sensitive information and thus it is important to prevent the indiscriminate disclosure of information. As a result, trust negotiations are ruled by two parameters that determine the amount of information that each participant *demands* and the amount of data that each participant is willing to *offer* in order to establish a base of trust.

The trust negotiation relationship can be represented with a notation that is consistent with the structures and components available from the SI* modelling language, as shown in Figure 2a. In this figure, actors A and B are represented by circles, the goal to be accomplished is represented by a squared oval, and these three elements are connected by a labelled arc. The label T_N is contained within a pentagon that is further parameterised with the information being requested from the other actor. Since a trust negotiation is a dual relationship it is necessary to have one arc in each direction.

However, and for the sake of simplicity, we propose an alternative notation with a single arc, as depicted in Figure 2b. Now, the pentagons at each side of the relationship are associated with their closest actor and point to the actor from whom information is being requested. Each of the pentagons have demand labels attached to them. These labels indicate the particular informational assets of interest to the actor as well as the level of detail that needs to be satisfied in order to establish the trust relationship.

4.3. Privacy Policy Definition

Privacy violations are usually associated with a loss of control over personal information and thus it is necessary to define privacy policies. To limit the release of data, our framework takes into consideration the ownership of data[4] and on top of that defines a set of labels that are attached to resources. The labels indicate the level of privacy that a particular resource must retain as well as the sensitivity of the resources. The privacy policy of each actor is defined based on these labels. Note that these are not the policies that the user will be assigned at run time, they are just a tool for the engineer to reason about suitable default policies for the system.

In previous extensions to the SI* modelling language (Asnar et al., 2011; Paci et al., 2013) the concept of *security requirement label* was defined. These labels are attached to assets (i.e., goals and resources) in order to specify the security properties that must hold for a particular asset. The security properties considered in these works include confidentiality, integrity, and availability. Similarly, we propose the use of a *privacy requirement label* to indicate that a particular resource must maintain a specific level of privacy.

Data resources may be additionally labelled with a *sensitivity level* to indicate how valuable this information is. The sensitivity level thus provides a natural indicator of the severity of leaking this informational asset. In other words, the sensitivity level helps to quantify the cost of a privacy violation. Moreover, sensitivity is related to the level of accuracy that the owner of the data is willing to reveal in such a way that the higher the sensitivity, the lower the precision of the data being disclosed. They are inversely proportional to each other.

The level of granularity of these labels depends on the application, and the requirements engineer is responsible for defining them. The choice of these labels will depend on the complexity of the application and the granurality needed for establishing the trust negotiation process. For the sake of simplicity, we will consider a discrete set of 3 levels, namely, *Low*, *Medium*, and *High*. Since the level of sensitivity is related to the accuracy of the data, we will also have 3 levels of granularity for each data type. The level of granularity of data depends on the type of data being considered. Suppose we are dealing with location information, we can provide these data at 3 different levels of detail, say, Low: city, $Medium$: zip code, and $High$: exact address. Just as the engineer is responsible for deciding the number of labels to use, it is also his role to decide the level of detail of the data associated with each label depending on the requirements of the application. These decisions are beyond the scope of this paper.

[4]Data ownership is a convoluted term that lacks a universal definition. Here, we define data ownership as the possession of information coupled with the ability to control the disclosure of that information, which is typically not known to others. For our purposes, the data owner is either the creator of an informational asset or the individual to whom the information refers.

4.4. Privacy Exposure

As actors release personal information, their privacy degrades. The disclosure of various pieces of information accumulates in a very particular way depending on the types of data being released. This is related to the notion of quasi-identifier (Vimercati and Foresti, 2011), which was introduced in the area of micro-data release protection to designate a set of attributes that are apparently innocuous but, in combination, can be used to re-identify individuals[5]. Therefore, it is paramount to keep track of the amount of data that may be released by each actor in order to control their privacy exposure. However, knowing how critical the combination of different types of data attributes is is not straightforward and very much depends on the amount of external information available to the attacker, which cannot be known in advance.

Here we propose a simple approach to the problem of disclosing multiple data items that consists in adding up the sensitivity levels of each of these items. To that end, we propose dividing the range of sensitivity levels into several sub-ranges. In Section 4.3 we considered a discrete set of 3 sensitivity levels and now we further divide each of the them into a lower and upper part, represented by $^-$ and $^+$ symbols, respectively. For example, the *Low* level has two sub-levels: L^- and L^+. Moreover, we consider two additional intermediate levels, namely, *Low-Medium* and *Medium-High*, with their corresponding sub-levels. Altogether, this is basically a 1 to 10 scale where each of the elements has a particular weight. As a result, these weights can be added up together in order to determine the privacy exposure caused by releasing information. A visual representation of this is depicted in Table 3.

In the simplest case, releasing two informational resources labelled with a *Low* sensitivity level result in a *Low*$^+$ privacy exposure. In the case that no further information is revealed, the privacy exposure can be regarded as *Low* but if an additional piece of low sensitivity data is disclosed, then the privacy exposure raises to the next level, up to *Low–Med*$^-$. Similarly, if a new release of data exposes an asset with a *Medium* sensitivity, we end up with a *Med–High*$^+$ privacy exposure, and so on, until the retained level of privacy is extremely low, which is the case that we represent with E in order to indicate that the privacy exposure is *Extreme*.

As previously stated, the intermediate levels considered here can be finally mapped to the original space consisting only of 3 levels: *Low*, *Medium*, and *High*. Depending on the criticality of the application scenario and the data being exposed, the requirements engineer can choose to take a strict or loose approach for the definition of the mapping function. A *strict* mapping function is useful in situations where the privacy of the users is extremely sensitive, say, a

[5]A former governor of the state of Massachusetts was re-identified based solely on the date of birth, zip code and sex. Moreover, these data are sufficient to re-identify 87% of the U.S. population (Sweeney, 2002).

Table 3: Privacy aggregation

		Low		Low-Med		Medium		Med-High		High	
		L^-	L^+	LM^-	LM^+	M^-	M^+	MH^-	MH^+	H^-	H^+
Low	L^-	L^+	LM^-	LM^+	M^-	M^+	MH^-	MH^+	H^-	H^+	E^-
	L^+	LM^-	LM^+	M^-	M^+	MH^-	MH^+	H^-	H^+	E^-	E^+
Low-Med	LM^-	LM^+	M^-	M^+	MH^-	MH^+	H^-	H^+	E^-	E^+	E^+
	LM^+	M^-	M^+	MH^-	MH^+	H^-	H^+	E^-	E^+	E^+	E^+
Med	M^-	M^+	MH^-	MH^+	H^-	H^+	E^-	E^+	E^+	E^+	E^+
	M^+	MH^-	MH^+	H^-	H^+	E^-	E^+	E^+	E^+	E^+	E^+
Med-High	MH^-	MH^+	H^-	H^+	E^-	E^+	E^+	E^+	E^+	E^+	E^+
	MH^+	H^-	H^+	E^-	E^+	E^+	E^+	E^+	E^+	E^+	E^+
High	H^-	H^+	E^-	E^+	E^+	E^+	E^+	E^+	E^+	E^+	E^+
	H^+	E^-	E^+	E^+	E^+	E^+	E^+	E^+	E^+	E^+	E^+

healthcare application. This function can be as simple as translating an intermediate sensitivity level into the level immediately above it. In this way, LM^- and LM^+ are both mapped to *Medium* and, similarly, MH^- and MH^+ are mapped to *High*. On the other hand, a *loose* mapping function makes more sense when the scenario and the data being transferred are not so critical. In this case, the function can map each of the elements of an intermediate level to the sensitivity level that is closer. In other words, the $^-$ element can be mapped to the original sensitivity level below it (on its left) and the $^+$ element can be mapped to the level above it. In any case, *E* will be mapped to *High*, unless a new category is added to represent that this situation is extremely sensitive. Clearly, other mapping functions are possible.

4.5. Privacy Assessment

So far we have considered how privacy degrades when one actor discloses informational assets to another actor. However, a single user may be involved in several trust negotiations and thus it is important to determine how to proceed in these situations.

As new trust negotiation relationships are established, the privacy exposure of the actors typically grows since they need to share more resources with other users. Yet it is not trivial to quantify how much privacy degrades, which basically depends on how the actors receiving the data behave. For that reason, it may be useful to consider different approaches for quantifying the privacy exposure of actors involved in trust negotiation relationships. We consider three main approaches to deal with this situation:

1. **Worst case**: assumes that the privacy of an actor degrades whenever the actor releases more detailed information about a resource regardless of the entity receiving these data. The rationale behind this approach is that the owner of the data loses control over these items as soon as they are shared with another actor and eventually they may be learned by any other actor in the system.
2. **Ideal case**: assumes that the privacy exposure of an actor depends on the maximum amount of data being released to a single actor. This approach assumes that the actors of the system will not share the informational assets received from one actor with other actors in order to reduce the privacy of this one actor.
3. **Realistic case**: assumes that the informational assets being offered by one actor to another can be eventually shared with any other actors connected with him by means of a trust relationship. Therefore, the privacy exposure of an actor is equal to the maximum amount of information that a group of related actors may learn if they shared all the information they know about this actor.

The first approach is the most conservative one as it considers that privacy is lost as soon as data leaves the sphere of control of the user. This is the typical approach to data privacy as one cannot know for certain whether the recipient of the data will redistribute them to other users. To the contrary, the second approach is the most permissive one as it presumes that the actors in the systems are never going to share information about another actor. In one sense, the first approach can be seen as if all the actors of the system are connected to each other and collude while the second approach considers that all the actors are isolated and follow the rules of the game.

The third approach can be regarded as the middle ground between the other two, since it assumes that actors can share information about another actor only if they are related to each other, i.e. there is a trust path in the model connecting them. This approach can be further complicated to consider two different cases: (a) whenever there is a path between two actors they share all the information they know, (b) the amount of information shared between two related actors is dependent on the length of the path that connects them. Moreover, if there is more than one path connecting two actors, we can assume that the amount of information leaked is equal to the leakage of the shortest path. Notwithstanding, one might argue that the greater the number of paths connecting two actors, the greater the likelihood that these actors share information. Many similar approaches may be devised but this is left for future work. In this paper we will concentrate on the first case.

Regardless of the approach, we will assume that the privacy of the users degrades in a way that it is consistent with the method described in Section 4.4. Moreover, in the Appendix we devise three basic scenarios where actors are involved in one or more trust negotiations. These scenarios are then fed into our framework and we compare the resulting privacy leak of the actors for the aforementioned approaches.

5. Model Formalisation

The graphical notation for privacy-aware trust negotiations presented in Section 4 has to be translated into a set of facts and rules to enable the automatic verification of the models. As mentioned in Section 3, SI* uses the ASP paradigm for this purpose. Next, we introduce both extensional and intensional predicates to formalise the graphical description of the model and then describe the rules that determine its behaviour. Some of these predicates and rules were introduced in our preliminary work in (Rios et al., 2016) but only superficially.

5.1. Predicates

The first predicate that we need to incorporate is one for representing the trust negotiation relationship itself. This is done by using the predicate *trust_neg*, which indicates that actors[6] a_1 and a_2 may initiate a trust negotiation as a means to achieve a common goal g. Note that it is unreasonable to have a trust negotiation where the two actors are the same, therefore it will always be the case that $a_1 \neq a_2$. Formally, the predicate is defined as follows:

P_1: trust_neg(Actor: a_1, Actor: a_2, Goal: g)

The predicate *offers* denotes that actor a is willing to offer resource r up to a given granularity level $l \in \{Low, Medium, High\}$. The granularity level offered is determined by the sensitivity of the resource in such a way that the higher the sensitivity level, the lower the granularity of the resource. This is denoted by means of the *sensitivity* predicate, which associates a resource r with a given sensitivity level l. Similarly, the predicate *demands* indicates that an actor a requests a resource r with at least a given granularity l. Additionally, this predicate indicates to which actor the resource is demanded. This is necessary since an actor can be involved in several trust negotiation relationships and may thus request data from different actors. It helps to identify which of the negotiations are satisfied. This is not the case in the *offers* predicate as it expresses the maximum level of granularity of data that the agent will release regardless of the agent involved in the trust negotiation. Finally, the predicate *privacy_req* denotes the level of privacy l that needs to be satisfied for a particular resource r, which is given by the application and set by the requirements engineer. Thus, if for example, the privacy level is *Low*, the exposure level will be *High*. These predicates are formally represented as follows:

P_2: offers(Actor: a, Resource: r, Level: l)
P_3: demands(Actor: a_1, Actor: a_2, Resource: r, Level: l)
P_4: sensitivity(Resource: r, Level: l)
P_5: privacy_req(Resource: r, Level: l)

So far we have only presented extensional predicates, which translate the graphical description of the system into a formal specification. In order to reason about the system and to validate it, we need to define a series of intensional predicates, which express relevant intermediate information. Intensional predicates will be derived after a set of rules, which define the semantics of the framework, are applied to the extensional predicates. Finally, some conclusions can be derived about the potential presence of privacy conflicts.

The predicate *satisfy* denotes that an actor a_1 is able to meet the demands of another actor a_2 for a particular resource r. Moreover, this predicate indicates the level of granularity l that the requested actor will offer to the requesting actor in the case that the trust negotiation is successful. This last parameter will be useful to account for the privacy exposure level of the requested actor. The predicates *n_demands* indicates that actor a_1 demands a total of n informational resources from actor a_2. Similarly, *n_satisfy* denotes the number of resources that the first actor is able to offer to the second actor at the desired granularity level. These predicates are necessary to determine whether or not a trust negotiation will be successful, which is denoted by the following *establish_trust* predicate. More precisely, this predicate represents that actors a_1 and a_2 trust each other to reach a particular goal g, meaning that they have reached an agreement according to the privacy policy of each actor. Moreover, we define an additional predicate, which is useful to determine the trust paths between actors. A trust path between two actors exists if there is a group of actors who can establish pairwise trust relationships. The predicate *trust_path* indicates that there is a trust path p of length n starting at actor a_1 and ending at actor a_2. Finally, predicate P_{11} indicates the opposite to predicate P_6 but will be useful for the identification of resources that are limiting the establishment of trust between actors.

P_6: satisfy(Actor: a_1, Actor: a_2, Resource: r, Level: l)
P_7: n_demands(Actor: a_1, Actor: a_2, $\mathbb{N}^0 : n$)
P_8: n_satisfy(Actor: a_1, Actor: a_2, $\mathbb{N}^0 : n$)
P_9: establish_trust(Actor: a_1, Actor: a_2, Goal: g)
P_{10}: trust_path(Actor: a_1, Actor: a_2, Path: p, $\mathbb{N}^+ : n$)
P_{11}: not_satisfy(Actor: a_1, Actor: a_2, Resource: r)

[6]We use the notion of actor for simplicity. Instead, the actual predicates and rules consider roles and agents as arguments.

P_{12}: data_exposure(Actor: a, Resource: r, $\mathbb{N}^0 : n$)
P_{13}: aggreg_func(Actor: a, $\mathbb{N}^0 : n$)
P_{14}: privacy_leak(Actor: a, $\mathbb{N}^0 : n$)
P_{15}: privacy_threat(Actor: a, Resource: r, Level: l)

The last four predicates (P_{12}-P_{15}) are useful for identifying privacy violations. The predicate *data_exposure* indicates that the resource r belonging to an actor a is exposed to a certain degree l. The predicate *aggreg_func* calculates the aggregated privacy exposure result for all the resources exposed by a given user according to the function described in Section 4.4. The predicate *privacy_leak* represents the level of exposure of a particular actor and, finally, the *privacy_threat* predicate denotes that the exposure level of an actor has exceeded the desired privacy exposure level.

Some additional predicates are used by the framework but they will be presented when necessary.

5.2. Rules

Once the required predicates have been presented we can proceed with the definition of the rules that determine the reasoning of the model. First, we describe the rules related to the process exchange of resources as a means to establish trust, and then we present the rules for detecting privacy threats caused by these negotiations.

5.2.1. Trust Negotiation Rules

The first set of rules described here, from R_1 to R_3, express that the sensitivity of a resource is inversely proportional to the granularity level that the actor owning the resource is willing to offer. Rule R_4 denotes that an actor A satisfies the demands of another actor B if the resource required by B is offered by A with a granularity level higher than or equal to the desired granularity[7]. In the case that actor A can satisfy the demands of actor B, he will follow a data minimisation principle thus sharing the resource to the granularity level being requested (i.e., L_B) and not to the level that A might be able to offer. Also note that in this rule there is no need to specify that actors A and B are different since it is not possible to derive from the model that an actor demands a resource to himself.

The next rule, R_5, establishes that two actors A and B can trust each other to reach a particular goal if there is a trust negotiation relationship between them and all the demands of each of the actors are satisfied. This rule is defined in a way that it enables unconditional trust if one of the actors (or both) trust the other actor without the need to receive any informational assets from him. Also note that this rule is not limited by the existence of privacy threats. Our approach here is to address trust and privacy issues separately. By doing this, the framework enables to determine whether or not a trust negotiation might be established based solely on the resources being offered and requested. In the case that trust can be established and there is a potential threat to privacy, the security expert can take measures to prevent the loss of privacy.

R_1: offers(A, R, *High*)	$\leftarrow$	own(A, R) $\wedge$ sensitivity(R, *Low*).
R_2: offers(A, R, *Med*)	$\leftarrow$	own(A, R) $\wedge$ sensitivity(R, *Med*).
R_3: offers(A, R, *Low*)	$\leftarrow$	own(A, R) $\wedge$ sensitivity(R, *High*).
R_4: satisfy(A, B, R, L_B)	$\leftarrow$	demands(B, A, R, L_B) $\wedge$ offers(A, R, L_A) $\wedge$ ($\mathrm{L}_A \succeq \mathrm{L}_B$).
R_5: establish_trust(A, B, G)	$\leftarrow$	trust_neg(A, B, G) $\wedge$ n_demands(A, B, D_{AB}) $\wedge$ n_satisfy(B, A, S_{BA}) $\wedge$ n_demands(B, A, D_{BA}) $\wedge$ n_satisfy(A, B, S_{AB}) $\wedge$ ($\mathrm{D}_{AB} \leq \mathrm{S}_{BA}$) $\wedge$ ($\mathrm{D}_{BA} \leq \mathrm{S}_{AB}$).
R_6: not_satisfy(A, B, R)	$\leftarrow$	demands(B, A, R, L_B) $\wedge$ offers(A, R, L_A) $\wedge$ ($\mathrm{L}_B \succ \mathrm{L}_A$).
R_7: not_satisfy(A, B, R)	$\leftarrow$	demands(B, A, R, L_B) $\wedge$ not offers(A, R, L_A).
R_8: trust_path(A, B, [ ], 1)	$\leftarrow$	establish_trust(A, B, G).
R_9: trust_path(B, A, [ ], 1)	$\leftarrow$	establish_trust(A, B, G).
R_{10}: trust_path(A, C, P, L)	$\leftarrow$	trust_path(A, B, P_{AB}, L_{AB}) $\wedge$ trust_path(B, C, [ ], 1) $\wedge$ not member(C, P_{AB}) $\wedge$ (A $\neq$ C) $\wedge$ (L = $\mathrm{L}_{AB} + 1$) $\wedge$ insLast(P_{AB}, B, P).

Rules R_6 and R_7 identify that the demands of an actor B for a particular resource R cannot be satisfied by actor A. This may happen for two reasons. The first being that actor A is not willing to offer the resource with as much precision as B is demanding, and the second being that A does not offer the resource requested by B. The *not_satisfy* predicate is useful for identifying the cause of unsuccessful trust negotiations.

The last set of rules represent all possible trust paths between actors. A trust path exists between any two actors A and B if they can establish a base of trust towards a particular goal. A path exists in each direction. This is represented by rules R_8 and R_9. Moreover, rule R_{10} establishes that a trust path P exists between actors A and C if there already exists a path P_{AB} between A and B of unitary length and there is another path between B and C such that C is not yet in the path. Note that this rule uses some built-in predicates available in DLV. In particular, we use *member* for checking the existence of an item in a list and *insLast* to insert an item at the end of a list.

5.2.2. Privacy Rules

The rules presented so far have been aimed at detecting whether the actors in the system are capable of establishing trust relationships based on the informational resources they are willing to share with other actors. Subsequently, we focus on detecting potential privacy issues arising as a consequence of the exchange of information between actors.

[7]We use the $\succeq$ symbol to compare ordinal values. In particular, *High* $\succ$ *Medium* $\succ$ *Low*.

As described in Section 4.5, dealing with the degradation of user privacy when the actor is involved in multiple trust negotiations may be approached from different angles. In particular, we consider three different scenarios: the worst case, the ideal case, and the realistic case. The first approach is the most conservative one as it considers that once an informational asset is released it will eventually be known by any actor of the system. The second approach assumes that the actors receiving data will not share it with anyone, and the third approach assumes that they will share data only with those actors they trust. Here, we provide the three approaches:

$$\begin{array}{lll}
R_{11}\text{: data\_exposure(A, R, X)} & \leftarrow & \text{offers(A, R, L)} \wedge \\
 & & \text{max\_exposure(R, X).} \\
R_{12}\text{: aggreg\_func(A, X)} & \leftarrow & \text{n\_low(A, N}_L\text{)} \wedge \\
 & & \text{n\_med(A,N}_M\text{)} \wedge \\
 & & \text{n\_high(A, N}_H\text{)} \wedge \\
 & & \text{X} = \text{N}_L + 5 \times \text{N}_M + 9 \times \text{N}_H. \\
R_{13}\text{: privacy\_leak(A, X)} & \leftarrow & \text{data\_exposure(A, \_ , \_)} \wedge \\
 & & \text{aggreg\_func(A, X).}
\end{array}$$

The *worst case* approach is represented by rules from R_{11} to R_{12}. In particular, R_{11} is used to determine the maximum exposure of the resources being offered by an actor. This rule takes advantage of the *max_exposure* predicate, which is true if X is the maximum level of exposure of a given resource. Rule R_{12} is used to calculate the aggregated privacy exposure for a given user. This rule counts the number of resources being offered at a low, medium and high granularity level and calculates the exposure based on the method defined in Section 4.4. Finally, rule R_{13} allows the privacy leak of an actor to be calculated, provided that he has exposed some of his resources.

The *ideal case* approach is modelled with rules R_{14} and R_{15}. The first of these rules is used to calculate the exposure of one actor to another actor provided that they have established a trust relationship. Again, this is done using the aggregation function described before, but in this case it must only account for the number of resources given to each particular actor separately. The following rule obtains the privacy leak of a user, which is calculated as the maximum exposure of the user to each of the actors.

$$\begin{array}{lll}
R_{14}\text{: data\_exposure\_id(A, B, X)} & \leftarrow & \text{trust\_path(A, B, [ ], 1),} \wedge \\
 & & \text{aggreg\_func\_id(A, B, X).} \\
R_{15}\text{: privacy\_leak\_id(A, X)} & \leftarrow & \text{data\_exposure\_id(A, \_, \_)} \wedge \\
 & & \text{max\_exposure\_id(A, X).}
\end{array}$$

The *realistic* approach is more complex as it involves the discovery of groups of actors that may share information from another actor. Thus, the number of rules required to model this situation is larger, as shown below.

The first two rules are intended to obtain all actors connected with any actor B that receives information from another actor A. This includes actor B itself (rule R_{16}) and all nodes that can be reached from B such that A is not in the path (rule R_{17}). This allows us to create all disjunct groups of actors that may share information

$$\begin{array}{lll}
R_{16}\text{: data\_recipient(A, B, B)} & \leftarrow & \text{trust\_path(A, B, [ ], 1).} \\
R_{17}\text{: data\_recipient(A, B, C)} & \leftarrow & \text{trust\_path(A, B, [ ], 1)} \wedge \\
 & & \text{trust\_path(B, C, P, L)} \wedge \\
 & & \text{not member(A, P)} \wedge \\
 & & (\text{C} \neq \text{A}). \\
 & & \\
R_{18}\text{: tmp\_exp(A, B, C, R, X)} & \leftarrow & \text{data\_recipient(A, B, C)} \wedge \\
 & & \text{satisfy(A, C, R, X).} \\
R_{19}\text{: data\_exposure\_r(A, B, R, X)} & \leftarrow & \text{tmp\_exp(A, B, \_, R, \_,)} \wedge \\
 & & \text{max\_exposure\_r(A, B, R, X).} \\
R_{20}\text{: group\_leak(A, B, X)} & \leftarrow & \text{data\_recipient(A, B, C)} \wedge \\
 & & \text{aggreg\_func\_r(A, B, R, X).} \\
 & & \\
R_{21}\text{: privacy\_leak\_r(A, X)} & \leftarrow & \text{satisfy(A, \_ , \_)} \wedge \\
 & & \text{max\_group\_exposure(A, X).}
\end{array}$$

from another particular actor. Then, with rule R_{18} we can obtain the leak level of a particular resource to any node of each group and rule R_{19} obtains the maximum exposure of a given resource using these intermediate predicates. Rule R_{20} obtains the total leak of all resources from A with each of the groups that know information about this actor and rule R_{21} calculates the privacy leak of A as the leak to the group that has most information about him.

Finally, we need to map the privacy exposure of a user with the defined privacy policies in order to detect potential privacy conflicts. To that end we first need to define the function for transforming the privacy leak value into a privacy label and then compare it with the privacy requirements of the user.

$$\begin{array}{lll}
R_{22}\text{: privacy\_label(A, }\textit{Low}\text{)} & \leftarrow & \text{privacy\_leak(A, X)} \wedge (0 < \text{X} \leq 3) \\
R_{23}\text{: privacy\_label(A, }\textit{Med}\text{)} & \leftarrow & \text{privacy\_leak(A, X)} \wedge (3 < \text{X} \leq 7) \\
R_{24}\text{: privacy\_label(A, }\textit{High}\text{)} & \leftarrow & \text{privacy\_leak(A, X)} \wedge (7 < \text{X} \leq 10) \\
R_{25}\text{: privacy\_label(A, }\textit{Extr}\text{)} & \leftarrow & \text{privacy\_leak(A, X)} \wedge (\text{X} > 10) \\
 & & \\
R_{26}\text{: max\_desired\_expos(R, }\textit{Low}\text{)} & \leftarrow & \text{privacy\_req(R, }\textit{High}\text{)} \\
R_{27}\text{: max\_desired\_expos(R, }\textit{Med}\text{)} & \leftarrow & \text{privacy\_req(R, }\textit{Med}\text{)} \\
R_{28}\text{: max\_desired\_expos(R, }\textit{High}\text{)} & \leftarrow & \text{privacy\_req(R, }\textit{Low}\text{)} \\
 & & \\
R_{29}\text{: privacy\_threat(A, R, L}_E\text{)} & \leftarrow & \text{max\_desired\_expos(R, L}_R\text{)} \wedge \\
 & & \text{privacy\_label(A, L}_E\text{)} \wedge \\
 & & (\text{L}_E \succ \text{L}_R)
\end{array}$$

Rules R_{22} to R_{25} describe the behaviour of the mapping function, which converts integers into labels as discussed in Section 4.4. Additionally, rules from R_{26} to R_{28} transform a privacy requirement label into the maximum desired exposure. The last rule R_{29} is in charge of comparing the privacy requirements for user data with their actual level of exposure. In the case that the actual exposure is greater than the maximum desired privacy exposure, the predicate *privacy_threat* is activated.

6. Evaluation

This section presents the validation of our privacy-aware trust negation framework. First, we describe the main features of the scenario as well as the roles involved in it together with the relationships between these roles. Then, we focus on a particular instantiation of the model in order to show how it translates into ASP. Finally, we run our framework in order to detect trust and privacy conflicts.

6.1. Scenario Definition

Trust negotiations are mostly relevant in situations where two entities, which do not know each other in advance need one another in order to achieve a common goal. There are many scenarios where these mechanisms are useful, including the typical trust negotiation scenario where an unknown client and a server exchange credentials in order to determine whether access to a controlled resource can be granted to the client. However, the proposed framework can also model more advanced scenarios where the goal is not simply an access control decision.

Therefore, to illustrate the utility of our framework for detecting privacy violations during the process of trust negotiations, we consider a social network intended to connect drivers with empty seats and passengers willing to travel and share the costs of the journey with the driver. This scenario is similar to the one provided by the BlaBlaCar community (BlaBlacar)[8].

In this simple scenario it is easy to identify basically two main types of roles, namely the *Driver* and the *Passenger*, whose main features are:

- *Driver* is the person who is going to drive his vehicle from one location to another and has seats available to share for that particular journey;
- *Passenger* is the person who is willing to take the same journey and is willing to share the costs incurred during the trip, say road tolls and fuel.

These two roles[9] share the goal *share ride* in order to reduce costs, meet other people or simply for convenience. To meet this goal, the actors need to establish a base of trust. Basically, the driver wants to know whether the passengers will pay for the expenses and the passengers want to know whether the driver is experienced or trustworthy. Thus, trust between entities needs to be established and this can be done by sharing information between the actors involved. However, an indiscriminate disclosure of information will result in a significant loss of privacy.

The data items owned by the participants of the ride-sharing scenario are depicted in Figure 3a. This figure provides a general representation in SI* of the actors involved in the system together with their goal and available resources. The model is adequate for illustrating the default (i.e., system-defined) policies but since privacy is a subjective property that very much depends on the experiences and beliefs of the user, the model should also consider privacy at the individual level (i.e., user-defined policies). Since we consider two levels for defining privacy policies when modelling the system, some of these policies may be defined only at one level. In the case that the policy is defined at user level we consider it to be the right policy and if defined at role level alone, it is inherited by the actors with no user-defined preferences. There may also exist situations in which the privacy preferences are defined both at the agent level and at the role level and be in conflict with each other. We resolve this by assuming that user-defined policies are dominant. This same issue was taken into consideration by Giorgini et al. (Giorgini et al., 2005b) when modelling functional trust in SI*.

For the purpose of the application scenario under consideration, the *Driver* must be able to provide information about the vehicle he/she owns, address information, the type of job, and personal details related to gender and age. Similarly, the *Passenger* might be required to provide personal details as well as information about his/her address, job and also information to contact him/her in the case that the driver is interested. Moreover, each piece of these data is labelled with a particular sensitivity level whose value must be established by the requirements engineer based on his/her own experience. Specifically, the figure represents that, by default, a *Driver* is not extremely concerned about privacy (i.e., privacy requirement is set to medium) as long as passengers are willing to cover the expenses of the journey. Similarly, the *Passenger* will usually demand a moderate level of privacy.

6.2. An Instantiation of the Model

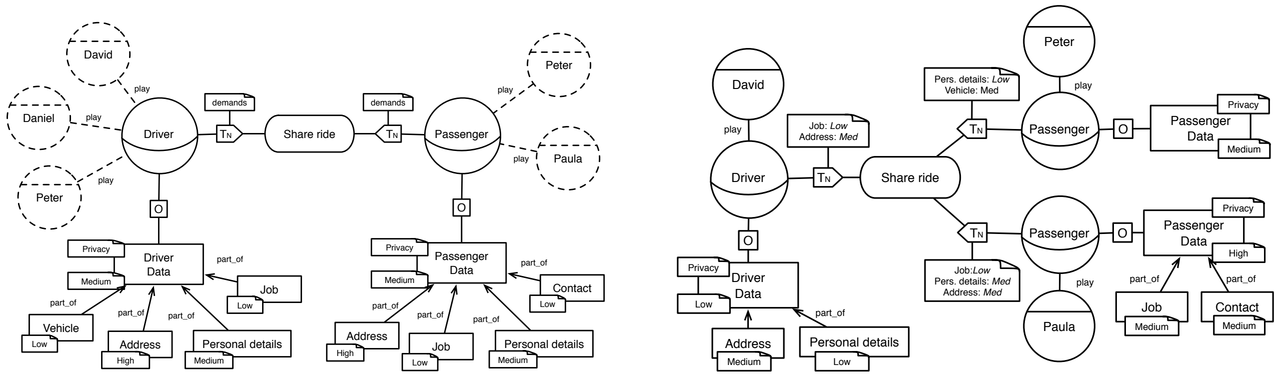
Now that we have presented the general model of the system, let us assume that *David* is a *Driver* who is going to travel by car from New York City to Philadelphia and wants to share two free seats in his car with other people in order to reduce travel costs. *Peter* and *Paula* both play the role of *Passenger* as they are willing to travel but they do not have either a license or a vehicle. Therefore, the driver can engage in a negotiation with each of the potential passengers in order to determine whether they will share the ride. These statements can be put into ASP syntax as follows[10]:

The actors of the system own a number of informational assets that they might be willing to share in order to fulfil the trust negotiation. These informational assets may be offered to other actors only when they are required and up to the granularity level determined by the sensitivity of the resources. Next we provide a subset of the predicates that are necessary to define the ownership of resources and the relationships between resources and with the roles. Moreover, we provide the predicates that define the system-defined privacy preferences for each role as well as the default sensitivity level for each informational asset:

[8]Although the participants in this type of social networks usually build trust based on reputation systems they might benefit from trust negotiations in fully distributed scenarios where there is no central authority to manage the reputation of the participants.

[9]A single person may play both roles at different instants, that is, a driver may be a passenger in a future journey and vice versa.

[10]Only the predicates that are relevant to our model are presented.



(a) General Model (b) Model Instantiation

Figure 3: Ride-sharing Scenario

```
role(driver)
role(pass)
agent(david)
agent(peter)
agent(paula)
play(david, driver)
play(peter, pass)
play(paula, pass)
goal(share_ride)
trust_neg(david,driver,peter,pass,share_ride)
trust_neg(david,driver,paula,pass,share_ride)
```

So far we have presented the representation of the general model but it is necessary to define the particular trust and privacy preferences of the agents, as shown in Figure 3b. The information being requested by *David* from the passengers is their *Address* and *Job*. In particular, *David* demands a very imprecise (i.e., low) level of address information as he simply wants to know whether the passenger is close to the departure location or on route to his destination. With respect to the job information he requires a medium level of detail. The actors playing the role *Passenger* also demand information from the *Driver*. More specifically, *Peter* is interested in information about the type of *Vehicle* that is owned by the driver as well as some *Personal details*. On the other hand, *Paula* demands more information before accepting a ride with someone. The information she requests is related to the *Job* and *Address* of the driver and *Personal details*. The level of granularity of the data requested by *Paula* is low for the first data item and moderate for the remaining ones. Some of the requests for particular resources (i.e., resource instances) are modelled using predicates like the ones shown next:

Moreover, the privacy preferences of particular actors and the sensitivity level of their resources can be re-defined by using two particular predicates, which are introduced here for the first time. In particular, the *user_def_sensitivity* to specify the sensitivity of a particular instance of a re-

```
resource(driver_data)
resource(pass_data)
resource(vehicle)
resource(driver_addr)
resource(pass_addr)
...
part_of(vehicle,driver_data)
part_of(driver_addr,driver_data)
part_of(pass_addr,pass_data)
...
own(driver, driver_data)
own(pass, pass_data)

privacy_req(driver_data,low)
privacy_req(pass_data,med)
sensitivity(driver_addr,high)
sensitivity(driver_job,low)
...
sensitivity(pass_addr,high)
sensitivity(pass_job,low)
```

source and *user_def_privacy_req* redefines the privacy requirements of a particular actor for a given resource. Some additional rules have been defined to generate the actual policies from system-defined and user-defined policies. These rules have not been included in Section 5.2 to improve the readability of the paper.

In particular, the sensitivity level for both the *Address* and *Personal details* of *David*'s resources have been lowered with respect to the default system settings. However, as *Paula* is a more privacy-conscious person her preferences for *Job*, and *Contact* have been changed to a medium sensitivity level. *Peter* has no special privacy demands and therefore inherits the system-defined privacy preferences and sensitivity levels. For the sake of readability, in Figure 3b we have only represented the informational assets whose sensitivity level have been re-defined.

```
demands(david,driver,peter,pass,
        resource_inst(pass_job,peter,pass),low)
demands(david,driver,paula,pass,
        resource_inst(pass_job,paula,pass),low)
...
demands(paula,pass,david,driver,
        resource_inst(driver_job,peter,driver),low)
demands(paula,pass,david,driver,
        resource_inst(driver_addr,peter,driver),med)

user_def_sensitivity(resource_inst(driver_addr,
                                  david,driver),med)
user_def_sensitivity(resource_inst(pass_addr,
                                  paula,pass),med)
...
user_def_privacy_req(resource_inst(driver_data,
                                  david,driver),low)
user_def_privacy_req(resource_inst(pass_data,
                                  paula,pass),high)
```

6.3. Detecting Policy Violations

Once the model has been instantiated and the rules[11] defined, we can run our framework for detecting policy violations. Specifically, the analyst will be interested in detecting problems related to the establishment of trust and privacy threats.

After running the particular instantiation of the scenario considered in Figure 3b we observe that none of the trust relationships are established. The reason is that *David* is demanding from both *Peter* and *Paula* a level of detail for the *Address* resource that they are not willing to offer. This can be easily spotted with the help of the *not_satisfy* predicate, as shown next.

```
not_satisfy(peter,pass,david,driver,
            resource_inst(pass_addr,peter,pass))
not_satisfy(paula,pass,david,driver,
            resource_inst(pass_addr,paula,pass))
```

At this point, the analyst must make a decision on whether to change the trust policy or the privacy policy. For the purpose of this paper we will consider that the decision taken is to change the default privacy policy and instead of assuming that the sensitivity of *Address* is high, it is assumed to be of medium sensitivity. Consequently, the level of detail that the passengers may offer will be higher and meet the expectations of the driver. Also, the expectations of the passengers are met by the driver, which in turn results in the establishment of a trust relationship between the actors, as shown next:

The establishment of trust between users may result in privacy violations. However, the evaluation of whether a privacy threat exists depends on how the data leak is assessed. In Section 4.5 we described three different approaches for determining the privacy leak depending on

```
establish_trust(david,driver,peter,pass,share_ride)
establish_trust(david,driver,paula,pass,share_ride)
```

the relationships between actors. Here, we focus on the worst case approach and even though there are very few relationships, the results may differ between using either of the proposed approaches. The reader is referred to the Appendix for interesting results and conclusions drawn from the application of the three approaches to more complex scenarios.

```
privacy_threat(david,driver,driver_data,extreme)
privacy_threat(paula,pass,pass_data,medium)
```

The previous predicates indicate that actors *David* and *Paula* are exposing themselves to an undesired level. This is true since their privacy requirements were set to low and high, respectively. There is no privacy threat for *Peter* since its desired privacy exposure is not strictly greater than its actual exposure.

7. Discussion

The framework proposed in this paper is intended for allowing software engineers to define trust and privacy policies for systems involving trust negotiations with the ultimate goal of detecting conflicts between them during the conception of the system. Certainly, this may raise a few questions that should be discussed and clarified.

First, trust negotiations are useful mechanisms for overcoming the uncertainty of interacting with previously unknown entities. Based on this assertion, one might argue that when the entities and their requirements are known in advance it is not necessary to resort to a negotiation. There might be situations in which the requirements engineer knows, say from a previous interview, the policies of users of the system. Therefore, it can be reasonable to think that there is no need to introduce trust negotiation mechanisms in the system. Notwithstanding, since trust and privacy are subjective properties that evolve in time based on the experiences of the users, the policies that were defined at the conception of the system will certainly change. In this situation, a software engineer precluding the use of trust negotiation mechanisms will be forced to re-engineer the system.

The scenario used for validation in Section 6 may raise questions about the validity of the framework since it is virtually impossible to know in advance all the users of large-scale systems. And even if this is the case, the requirements engineer should not be responsible for defining the trust and privacy policies of the users of the system. Since trust and privacy are subjective properties, the decision on the policies correspond to the users. What the framework is actually offering to the requirements engineer is not a mechanism for deciding about user policies and for detecting conflicts between them, but a tool for reasoning

[11]Visit `https://www.nics.uma.es/pub/development/privtrust.zip` for a complete set of rules and predicates

and finding the most suitable ones. These policies can be then used as the default system policies.

Moreover, the framework has been designed to cover as many situations as possible. There may be applications where some of the policies are fixed and cannot be changed, or their values must be within a particular range, and there will be applications where the user has full control on how to set up personal policies. Also, it is important to highlight that the framework is abstract enough to cover different trust negotiation strategies and protocols for exchanging information between actors. In this way, the requirements engineer can reason about, regardless the underlying mechanisms. We consider that all these decisions are not to be made during the requirements engineer phase but later in the software development life cycle.

8. Conclusion

This paper has presented a framework for representing trust negotiation systems based on the SI* modelling language. The proposed framework includes the definition of a number of rules written in ASP specification, which enables reasoning about the behaviour of the system. Thus, from a particular model of the system it is possible to automatically extract a set of ASP predicates, which formally describe it. The rules are triggered with these predicates and finally a stable model to detect inconsistencies is generated.

More precisely, the proposed framework is designed to identify problems derived from trust and privacy policies. On the one hand, the framework reveals which of the requirements that a user has to establish a trust relationship have not been satisfied by other users and for what reason, either because a particular resource is not being offered or because the level of detail that the other user is offering is insufficient. On the other hand, the framework also enables the detection of potential privacy threats to users involved in multiple trust negotiations and offering resources beyond that permitted by their privacy policy. Furthermore, the framework has been successfully validated with a distributed car-sharing social network scenario.

The framework is specialised in the reasoning about privacy-respectful trust negotiation systems although it might be extended to consider more general privacy problems related to the loss of control over personal data, as it can be the case in social networks. This is precisely one of our most prominent lines of future work together with the definition of more sophisticated mechanisms for measuring the degradation of privacy with every single piece of data the user publishes or shares with other entities. This is a promising line of work since the upcoming EU data protection regulation, GDPR 2016/679, demands for mechanisms to enable individuals to control their own personal data. We are also planning to extend this work to consider other relevant privacy aspects such purpose and storage limitation, as well as to consider subsequent phases of the software development life cycle including the application of the framework to real systems with its own privacy policies.

Acknowledgements

This work has been partially funded by the Spanish Ministry of Economy and Competitiveness through the SMOG project (TIN2016-79095-C2-1-R) and the PRECISE project (TIN2014-54427-JIN), co-financed by FEDER.

This work has received funding from the European Union Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 675320. This work reflects only the authors view and the Research Executive Agency is not responsible for any use that may be made of the information it contains.

References

Alviano, M., Faber, W., Leone, N., Perri, S., Pfeifer, G., Terracina, G., 2011. The Disjunctive Datalog System DLV, in: de Moor, O., Gottlob, G., Furche, T., Sellers, A. (Eds.), Datalog Reloaded: First International Workshop, Datalog 2010, Oxford, UK, March 16-19, 2010. Revised Selected Papers, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 282–301.

Asnar, Y., Li, T., Massacci, F., Paci, F., 2011. Computer Aided Threat Identification, in: 2011 IEEE 13th Conference on Commerce and Enterprise Computing, pp. 145–152.

Bertino, E., Ferrari, E., Squicciarini, A.C., 2004. Trust-X: A Peer-to-Peer Framework for Trust Establishment. IEEE Trans. on Knowl. and Data Eng. 16, 827–842.

BlaBlacar, . Share your journey with BlaBlacar - Trusted Ridesharing. https://www.blablacar.com. [Last Access: June 2017].

Bonatti, P., Coi, J.L.D., Olmedilla, D., Sauro, L., 2010. A rule-based trust negotiation system. IEEE Transactions on Knowledge and Data Engineering 22, 1507–1520.

Brewka, G., Eiter, T., Truszczyński, M., 2011. Answer Set Programming at a Glance. Commun. ACM 54, 92–103.

Castro, J., Giorgini, P., Kolp, M., Mylopoulos, J., 2005. Tropos: A Requirements-Driven Methodology for Agent-Oriented Software, in: Henderson-Sellers, B., Giorgini, P. (Eds.), Agent-Oriented Methodologies, Idea Group.

Deng, M., Wuyts, K., Scandariato, R., Preneel, B., Joosen, W., 2011. A privacy threat analysis framework: Supporting the elicitation and fulfillment of privacy requirements. Requir. Eng. 16, 3–32.

Giorgini, P., Massacci, F., Mylopoulos, J., Zannone, N., 2005a. Modeling Security Requirements Through Ownership, Permission and Delegation, in: 13th IEEE International Conference on Requirements Engineering (RE'05), IEEE. pp. 167–176.

Giorgini, P., Massacci, F., Zannone, N., 2005b. Security and Trust Requirements Engineering. Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 237–272.

Horkoff, J., Yu, E., 2013. Comparison and evaluation of goal-oriented satisfaction analysis techniques. Requirements Engineering 18, 199–222.

Kalloniatis, C., Kavakli, E., Gritzalis, S., 2008. Addressing privacy requirements in system design: the PriS method. Requirements Engineering 13, 241–255.

van Lamsweerde, A., 2004. Elaborating Security Requirements by Construction of Intentional Anti-Models, in: Proceedings of the 26th International Conference on Software Engineering, IEEE Computer Society, Washington, DC, USA. pp. 148–157.

van Lamsweerde, A., Letier, E., 2000. Handling Obstacles in Goal-Oriented Requirements Engineering. IEEE Transactions on Software Engineering 26, 978–1005.

van Lamsweerde; R. Darimont ; E. Letier, A., 1998. Managing Conflicts in Goal-Driven Requirements Engineering. IEEE Transactions on Software Engineering 24, 908–926.

Lee, A.J., Winslett, M., Perano, K.J., 2009. TrustBuilder2: A Reconfigurable Framework for Trust Negotiation, in: Proceedings of the 3rd IFIP WG 11.11 International Conference on Trust Management, Springer. pp. 176–195.

Massacci, F., Mylopoulos, J., Zannone, N., 2010. Security Requirements Engineering: The SI* Modeling Language and the Secure Tropos Methodology, in: Ras, Z.W., Tsay, L.S. (Eds.), Advances in Intelligent Information Systems, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 147–174.

McDonald, A.M., Cranor, L.F., 2008. The cost of reading privacy policies. I/S Journal for Law and Policy for the Information Society. 2008, Privacy Year in Review 4.

McDonald, A.M., Reeder, R.W., Kelley, P.G., Cranor, L.F., 2009. A Comparative Study of Online Privacy Policies and Formats. Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 37–55.

Mellado, D., Blanco, C., Sánchez, L.E., Fernández-Medina, E., 2010. A systematic review of security requirements engineering. Computer Standards & Interfaces 32, 153 – 165.

Milne, G.R., Culnan, M.J., 2004. Strategies for reducing online privacy risks: Why consumers read (or don't read) online privacy notices. Journal of Interactive Marketing 18, 15–29.

Mouratidis, H., GiorginiI, P., 2007. Secure Tropos: A Security-Oriented Extension of the Tropos Methodology. International Journal of Software Engineering and Knowledge Engineering 17, 285–309.

Notario, N., Crespo, A., Martín, Y., del Álamo, J.M., Métayer, D.L., Antignac, T., Kung, A., Kroener, I., Wright, D., 2015. PRIPARE: Integrating Privacy Best Practices into a Privacy Engineering Methodology, in: International Workshop on Privacy Engineering Proceedings, pp. 151–158.

Paci, F., Fernandez-Gago, C., Moyano, F., 2013. Detecting Insider Threats: A Trust-Aware Framework, in: Availability, Reliability and Security (ARES), 2013 Eighth International Conference on, pp. 121–130.

Paja, E., Dalpiaz, F., Giorgini, P., 2015. Modelling and Reasoning about Security Requirements in Socio-Technical Systems. Data and Knowledge Engineering 98, 123–143.

Rios, R., Fernandez-Gago, C., Lopez, J., 2016. Privacy-aware trust negotiation, in: 12th International Workshop on Security and Trust Management (STM), Springer, Heraklion, Crete, Greece. pp. 98–105.

Squicciarini, A., Bertino, E., Ferrari, E., Paci, F., Thuraisingham, B., 2007. PP-Trust-X: A System for Privacy Preserving Trust Negotiations. ACM Trans. Inf. Syst. Secur. 10.

Stavropoulos, V., 2011. Secure Software Engineering Initiatives: Listing (SEE) Initiatives across Europe and Abroad. Technical Report. European Network and Information Security Agency (ENISA).

Sweeney, L., 2002. k-Anonymity: A Model for Protecting Privacy. Int. Journal on Uncertainty, Fuzziness and Knowledge-Based Systems 10, 557–570.

Vimercati, S.d.C.d., Foresti, S., 2011. Encyclopedia of Cryptography and Security. Springer US, Boston, MA. chapter Quasi-Identifier. pp. 1010–1011.

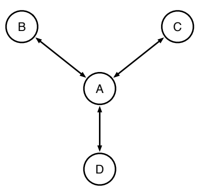
Winslett, M., Yu, T., Seamons, K.E., Hess, A., Jacobson, J., Jarvis, R., Smith, B., Yu, L., 2002. Negotiating Trust on the Web. IEEE Internet Computing 6, 30–37.

Yu, E., 1996. Modelling Strategic Relationships for Process Reengineering. Ph.D. thesis. University of Toronto. Canada.

Appendix A

Monitoring the privacy exposure of actors in the system when they have the ability to engage in multiple trust negotiations is a convoluted issue. In Section 4.5 we propose three different approaches for dealing with this situation and here we present three basic scenarios to help demonstrate the differences among them. These scenarios differ in the number of trust relationships that may be established between the actors of the system. As new trust relationships appear it also increases the number of resources being shared between actors and thus their privacy exposure.

In the first scenario there are only three trust negotiation relationships, while in the second and third scenarios there are four and five relationships, respectively. These scenarios are depicted in Figure 4, 5 and 6, where the left-hand side of the figures represent, in a very simplified way, the trust negotiation relationships between the actors of the system. The right-hand side of each figure indicates the resources being offered by each actor together with their granularity level (represented as integers). For example, actor B in the first scenario is willing to offer two resources $r4$ and $r5$ up to a $Medium$ and Low granularity, respectively. Actor A needs a low level of detail from these two resources in order to trust B.

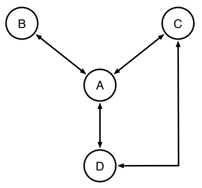


(a) Relationships

	A	B	C	D
offers	$r1:2$ $r2:1$ $r3:3$	$r4:2$ $r5:1$	$r6:3$	$r7:1$ $r8:3$
requested by	B: $\{r1:2\}$	A: $\{r4:1, r5:1\}$	A: $\{r6:2\}$	A: $\{r7:1\}$
	C: $\{r1:1, r3:2\}$			
	D: $\{r1:1, r2:1, r3:1\}$			

(b) Resources

Figure 4: Scenario 1

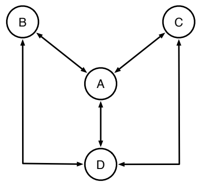


(a) Relationships

	A	B	C	D
offers	$r1:2$ $r2:1$ $r3:3$	$r4:2$ $r5:1$	$r6:3$	$r7:1$ $r8:3$
requested by	B: $\{r1:2\}$	A: $\{r4:1, r5:1\}$	A: $\{r6:2\}$	A: $\{r7:1\}$
	C: $\{r1:1, r3:2\}$		D: $\{r6:1\}$	C: $\{r8:2\}$
	D: $\{r1:1, r2:1, r3:1\}$			

(b) Resources

Figure 5: Scenario 2



(a) Relationships

	A	B	C	D
offers	$r1:2$ $r2:1$ $r3:3$	$r4:2$ $r5:1$	$r6:3$	$r7:1$ $r8:3$
requested by	B: $\{r1:2\}$	A: $\{r4:1, r5:1\}$	A: $\{r6:2\}$	A: $\{r7:1\}$
	C: $\{r1:1, r3:2\}$	D: $\{r4:2\}$	D: $\{r6:1\}$	C: $\{r8:2\}$
	D: $\{r1:1, r2:1, r3:1\}$			D: $\{r7:1, r8:3\}$

(b) Resources

Figure 6: Scenario 3

The relationships in the first scenario are such that the various actors are isolated from one another, except for the relationship with actor A. The second scenario introduces a new relationship between actors C and D, which could be used to share information about actor A.

Table 4: Privacy Exposure Results

	worst case				**ideal case**				**realistic**			
	A	B	C	D	A	B	C	D	A	B	C	D
scenario 1	11	2	5	1	6	2	5	1	6	2	5	1
scenario 2	11	2	5	6	6	2	5	5	7	2	5	6
scenario 3	11	6	5	10	6	5	5	10	11	6	5	10

This new relationship requires the exchange of additional resources between actors C and D, as shown in the table in Figure 5. Finally, another relationship between D and B is introduced in the third scenario, which also implies the disclosure of information between them. In this case, A releases informational assets to all other three actors, which may gather each piece of information A shares with them to better compromise his/her privacy.

The results of applying the three different approaches to these scenarios are presented in Table 4. Note that the results are obtained by applying the mechanism described in Section 4.4 and are presented as integer values rather than as labels for the sake of clarity.

As expected, the first approach results in the largest privacy exposure in all three scenarios because if there is an actor or group of actors receiving an informational asset to a particular level of detail, this is accounted for by the worst case approach. Interestingly, the privacy exposure in the third scenario for the worst case and the realistic approaches is the same for all actors. The reason is that for any actor sharing an informational asset, this asset reaches all other actors since there are relationships between all of them. In this case, both methods are equivalent because any actor may know all the resources shared by other actors up to the maximum granularity level provided.

Moreover, the second approach always results in the lowest privacy exposure. This is also an expected result since each of the actors is considered to be independent of each other, meaning that they are assumed not to share the information they receive, which is desirable but cannot be ensured. In Scenario 1, the results for the ideal case and the realistic approaches coincide because the actors are independent of each other anyway. However, this is not the case in Scenario 2 since a new trust negotiation relationship is introduced that enables actors C and D to share information about A.

Finally, note that as the number of interconnections grow between actors, their privacy degrades (i.e., the privacy exposure is higher). This is not surprising since more successful trust negotiations imply an increased number of resources being shared between the actors involved in the negotiation. In particular, the privacy exposure of D is very low in the first scenario but since it becomes involved in two additional negotiations in Scenario 3, the exposure of this actor grows dramatically.