Building Trust and Reputation In: A Development Framework for Trust Models Implementation

Francisco Moyano, Carmen Fernandez-Gago, Javier Lopez
NICS
www.nics.uma.es
University of Malaga,
Spain
{moyano,mcgago,jlm}@lcc.uma.es

Abstract

During the last years, many trust and reputation models have been proposed, each one targeting different contexts and purposes, and with their own particularities. While most contributions focus on defining ever-increasing complex models, little attention has been paid to the process of building these models inside applications during their implementation. The result is that models have traditionally considered as ad-hoc and after-the-fact solutions that do not always fit with the design of the application. To overcome this, we propose an object-oriented development framework onto which it is possible to build applications that require functionalities provided by trust and reputation models. The framework is extensible and flexible enough to allow implementing an important variety of trust models. This paper presents the framework, describes its main components, and gives examples on how to use it in order to implement three different trust models.

1 Introduction

There is not a standard definition of trust, although it is agreed that it is of paramount importance when considering systems security, as a tool to leverage decision-making processes. The concept of trust spans across several areas beyond computer science, such as psychology, sociology or economy.

The concept and implications of trust are embodied in the so-called trust models, which define the rules to process trust in an automatic or semi-automatic way in a computational setting. There are different types of trust models, each one considering trust in different ways and for different purposes. The origins of trust management date back to the nineties, when Marsh [10] proposed the first comprehensive computational model of trust based on social and psychological factors. Two years later, Blaze [2] identified trust management as a way to enhance the problem of authorization, which up to that date was separated into authentication and access control.

These two seminal contributions reveal the two main branches or categories of trust models that have been followed until today, and which we classified in a previous work [13]. On the one hand, and following Marsh's approach, we find evaluation models, where factors that have an influence on trust are identified, quantified and then aggregated into a final trust score. Uncertainty and evaluation play an important role in these models, as one entity is never completely sure whether it should trust another entity, and a decision process is required after evaluating the degree of trust placed in the entity.

On the other hand and following Blaze's approach, we find decision models, which are tightly related to authorization. An entity holds credentials and a policy verify whether these credentials are enough to permit access to certain resources. Here, trust evaluation is not so important in the sense that there are no degrees of trust (and as a consequence, there is not uncertainty), and the outcome of the process is a binary answer: access granted or access denied. In this paper, we lay aside these models and focus only on evaluation models.

Both categories, evaluation and decision models, evolved, leading to ever-complex models. One of the branches of evaluation models with higher impact has been reputation models, in which a reputation score about a given entity is derived from other entities' opinions about it. Reputation and trust are related concepts, and as stated by [8], reputation can be used to determine whether an entity can trust another entity.

One issue with trust models is that they are very context-dependent, and are often designed as ad-hoc mechanisms to work in a limited range of applications. Actually, the standard is to plug a trust model into an existing, already-built application after-the-fact. This might lead to architectural mismatches between the application and the model, and the reusability of the model could also be damaged. Moreover, it is not possible for the model to exploit all the information available to the application, since there is not any systematic procedure to include the model as a holistic part of the application. As a consequence, there are no mechanisms to consider trust requirements from the very beginning of the software development lifecycle or to align the design of the model with the design of the application.

To overcome these shortcomings, we propose an object-oriented development framework that allows implementing trust evaluation models as a core part of the applications themselves. Our aim is to assist developers during the development of applications that might require using evaluation models. The contributions of this paper are (i) a domain analysis for trust evaluation models; (ii) the elicitation of the requirements that the framework should meet; (iii) a first design of the framework architecture; (iv) and guidelines to implement three different trust evaluation models using the framework.

The rest of the paper is organized as follows. Section 2 reviews several contributions that are related to ours. A conceptual model of trust, which constitutes the domain analysis for the framework, is presented in Section 3. This analysis is used as an input to elicit the requirements and the design of the framework architecture, described in Section 4, whereas Section 5 explains how the framework can be used to implement three evaluation models. Finally, the conclusion and future work are presented in Section 6

2 Related Work

SECURE project [3] proposes a trust model to formally reason about trust, and a framework to provide applications with trust functionalities. Trust decisions rely on cost-PDFs that compare the benefits of a given interaction with the cost of such interaction. Thus, although the authors propose an interesting framework, we do not find it general enough to implement other types of trust or reputation models found in the literature.

Kiefhaber et al. [15] present the Trust-Enabling Middleware, which provides applications running on top of it with methods to save, interpret and query trust related information. The middleware uses built-in functions to measure the reliability of nodes by considering packets losses. Although rather complete, it lacks a framework-oriented approach since it does not make explicit the process of implementing existing or new trust models, and its focus is on distributed, message-oriented applications.

Huynh [6] proposes the Personalized Trust Framework (PTF), a rule-based system that makes use of semantic technologies for, given a domain, to apply the most suitable trust model. In a similar direction, Suryanarayana et al. [17] present PACE (Practical Architectural approach for Composing Egocentric trust), an architectural style for composing different trust models into the architecture of a decentralized peer in a P2P architecture. The first contribution is a user framework that assists users in determining the trustworthiness of resources, but it is not a development framework. The second contribution is an architectural style. Thus, its purpose is helping the architect of the application with a style to compose trust models, but like the PTF, it is not a framework, in the sense that it does not provide developers with mechanisms to implement trust models, nor to use them in their own applications.

Har Yew [5] presents a computational trust model and a middleware called SCOUT, made up of three services that implement the model: the evidence gathering service, the belief formation service and the emotional trust service. Regardless being a comprehensive model, it is not designed as an extensible framework and it is not clear, if possible at all, how a developer could implement existing trust models.

Finally, Lee and Winslett present TrustBuilder2 [9], where they propose an extensible framework that supports the adoption of different negotiation-based trust models. Although this is a development framework, they focus on decision models, laying aside the evaluation models we are considering in this paper.

3 Trust and Reputation: A Domain Analysis

The aim of this section is to shed light on concepts related to trust and reputation. First, in Section 3.1, we discuss some definitions of trust that are often found in the literature, whereas in Section 3.2 we put forward a conceptual model in the form of knowledge graphs that constitutes a domain analysis of trust and reputation models. This analysis is required for identifying the concepts that are likely to be part of the framework, as well as their relationships.

3.1 Definitions

Many definitions of trust have been provided along the years. This is due to the complexity of this concept, which spans across several areas such as psychology, sociology, economics, law, and more recently, computer science. The vagueness of this term is well represented by the statement "trust is less confident than know, but also more confident than hope" [12].

Gambetta [4] defines trust as "a particular level of the subjective probability with which an agent will perform a particular action [...] in a context in which it affects our own action". McKnight and Chervany [11] explain that trust is "the extent to which one party is willing to depend on the other party in a given situation with a feeling of relative security, even though negative consequences are possible". Ruohomaa and Kutvonen [16] state that trust is "the extent to which one party is willing to participate in a given action with a given partner, considering the risks and incentives involved". Finally, Har Yew [5] defines trust as "a particular level of subjective assessment of whether a trustee will exhibit characteristics consistent with the role of the trustee, both before the trustor can monitor such characteristics [...] and in a context in which it affects the trustor's own behaviour".

We propose the following definition : *trust is a subjective, context-dependent property that is required when (i) two entities need to collaborate (i.e. there is a dependence relationship between them and there exists the willingness to collaborate), but they do not know each other beforehand, (ii) and when the outcome of this collaboration is uncertain (i.e. entities do not know if they will perform as expected) and risky (i.e. negative outcomes are possible).* In this situation, trust acts as a mechanism to reduce the uncertainty in the collaboration and to mitigate the risk. As risk increases (either the probability or the impact of negative outcomes), trust becomes more crucial.

The concept of reputation is more objective than the concept of trust. According to the Concise Oxford dictionary, reputation is "what is generally said or believed about a person or the character or standing of a thing". Although the exact relationship between trust and reputation remains fuzzy, we think that Jøsang [8] linked these two terms appropriately with the following two statements: "I trust you because of your good reputation" and "I trust you despite your bad reputation". Thus, reputation can be considered as a building block, or indicator, to determine trust, although it does not have the final say.

3.2 Conceptual Model

This section presents the most important concepts related to evaluation trust models. These concepts were identified surveying relevant literature and finding commonalities and variations in the definition of different models. This conceptual model constitutes a domain analysis and the starting point for the framework requirements elicitation and for the architecture design, as some concepts and relationships can map to object-oriented components. The conceptual framework is graphically described by means of knowledge graphs and using a UML notation, as depicted in Figures 1 and 2. Due to space limitations, we concentrate on those concepts that have a higher impact for the requirements and architecture of the framework.

A trust model aims to compute trust in a given setting. This setting should have, at least, two entities that need to interact. An entity might play a role or even several ones. The basic roles are trustor (the entity that places trust) and trustee (the entity on which trust is placed). Once there is a trustor and a trustee, we claim that a trust relationship has been established. A trust relationship has a purpose, which can be for example controlling the access to a resource, the provision of a resource or the identity of an entity. It might also serve to set trust in the infrastructure (devices, hardware, etc). In the very end, the purpose of a trust model is to aid making a decision. At the higher level, it is a trust decision in the sense of answering the question: would this entity behave as expected under this context? At a lower level, an entity trusts a property of another entity. For instance its capability to provide a good quality of service. A trust model also makes some assumptions, such as "entities will provide only fair ratings" and follows a modeling method.

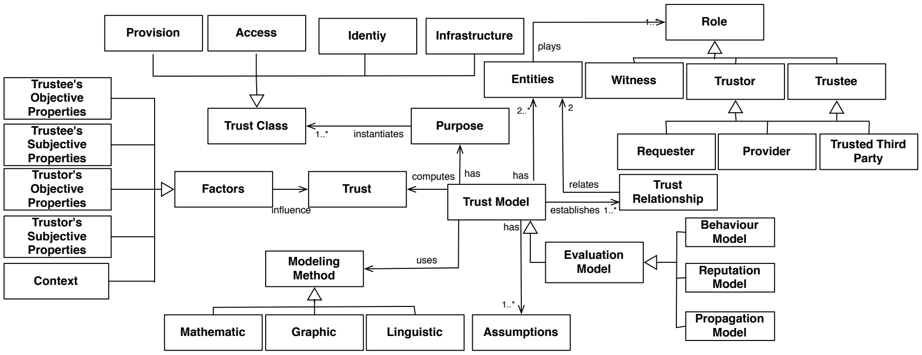


Figure 1: Concepts for Evaluation Models (i)

There are three types of evaluation models, namely reputation models, behaviour models and propagation models.

Behaviour models often follow a trust lifecycle with three phases. In the bootstrapping phase, initial trust values are assigned to the entities of the system. Then, some monitoring is performed to observe a variable or set of variables. Finally, a trust assessment process is done in order to assign values to these variables and to aggregate them into a final trust evaluation.

In these models, trust relationships are tagged with a trust value that describe to what extent the trustor trusts the trustee. This trust value has semantics and dimension, which might be simple or a tuple. Trust values are assigned during trust assessment through trust metrics, which receive a set of variables as input and produce a measure of one or several attributes using a computation engine. There are several computation engines used in the literature, ranging from the most simple ones such as summation engines, to complex ones that entail probability distributions or fuzzy logic.

There are several sources of information that might feed a trust metric. The most common one is the direct interaction of the entity with the trustee. Other possible sources of information, although less frequent, are sociological information (e.g. considering the roles of entities or their membership to a group) and psychological information (e.g. prejudice).

Reputation models can be, in turn, another source of information where opinions of a given trustee by different entities are made public and are used to compute a score. Reputation can be centralized or distributed, depending on whether reputation scores are stored in a central location or by each entity.

Propagation models aim to create new trust relationships from existing ones. Some of them assume that trust is transitive and exploit this property. New trust values are often computed by means of operators, and in several models, we find two of them: a concatenator and an aggregator. The former is used to compute trust along a trust path or chain, whereas the latter aggregates the trust values computed for each path into a final trust value.

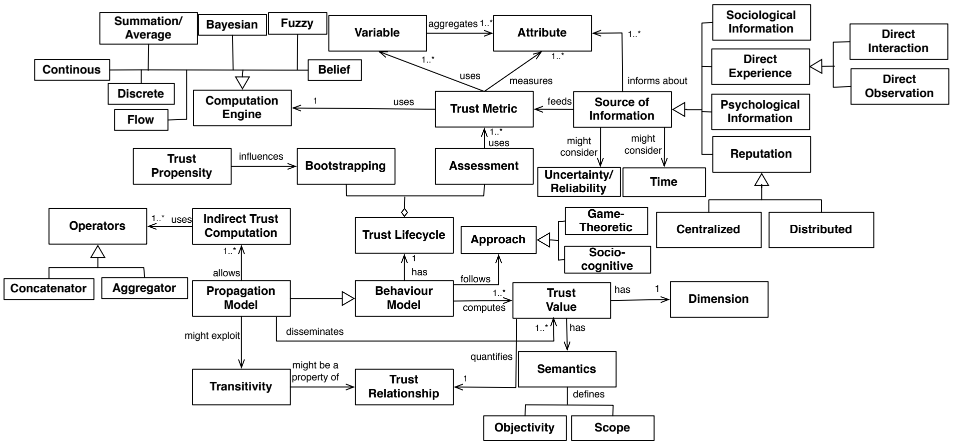


Figure 2: Concepts for Evaluation Models (ii)

4 Trust and Reputation Development Framework

In this section, the object-oriented development framework for trust is presented. Both the requirements and the design are influenced by the domain analysis presented in the previous section. Section 4.1 describes the requirements that the framework should fulfil, whereas Section 4.2 presents the first version of the framework architecture.

4.1 Framework Requirements

This section summarizes the requirements that the framework must meet. At a high-level, the framework has to support the implementation of three types of evaluation models, namely reputation models, behaviour models and propagation models. Although these models have commonalities, they also pose subtle differences that the framework must support.

The primary goal of reputation models is to compute reputation scores for entities. These scores must be stored (centrally or distributively) and entities should be able to access this information before interacting with other entities. On the other hand, behaviour models establish relationships between entities, and their main goal is to compute trust values for these relationships. Finally, propagation models also build on trust relationships, and their primary goal is to disseminate trust information to establish new trust relationships.

The following list of requirements describes the coarse-grained functionality that the framework should provide to developers:

- Entities management: entities hold trust values in other entities. The framework must allow the creation, binding and naming of entities.
- Trust relationships management: trust relationships might change along time. New trust relationships might be created (e.g. by propagation models), other relationships might be deleted, and it is likely that trust values change too.
- Trust metrics definition: although the framework can provide some default built-in metrics implementations, it is important to let developers to define their own trust metrics, as they are the core concept in evaluation models.
- Variables management: a trust metric is composed of variables. It is important to let developers to create new variables, which can be used by user-defined metrics.
- Computation engines management: an engine implements a trust metric. This engine uses variables according to certain rules. Engines range from simple summation functions to complex probability distributions.
- Indirect trust computation: the framework should provide ways to determine the value of an undefined trust relationship based on defined ones by propagating trust information.

- Operators definition: indirect trust computation relies on operators that take trust paths as input and return trust values as output (and thus, a new trust relationship). Although several operators should be provided by default, the framework should allow developers to define new operators.

The ultimate goal of the framework is to allow developers to implement both existing evaluation models and new ones. Next section describes the architecture that supports these requirements.

4.2 Framework Architecture

This section describes a first version of the framework architecture. The structural view of the architecture is depicted as a class diagram in Figure 3. Note that some classes have been mapped directly from the conceptual model described in Section 3, such as *Entity* and *TrustRelationship* among others.

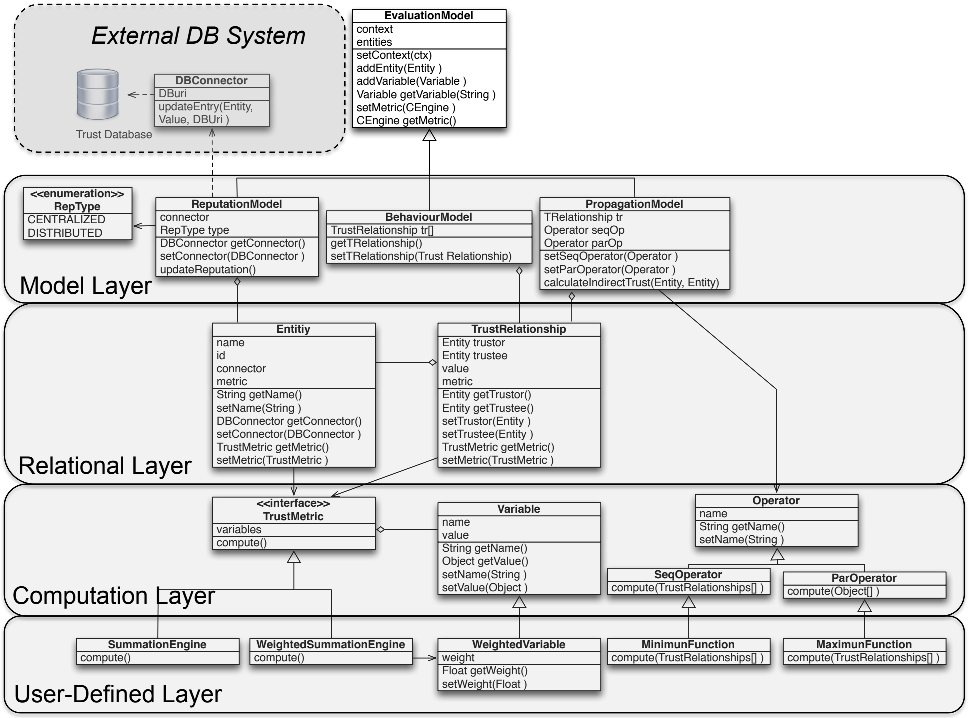


Figure 3: Framework Architecture

The architecture follows a layered design, where each layer uses the services provided by the lower layer. Likewise, the framework follows a grey-box approach, where the developer can use several functionalities in a black-box fashion as well as define new functionalities based on his needs. Next we describe the classes and relationships for each of the layers.

4.2.1 Model Layer

In this layer we find the models that the developer can implement, namely reputation models, behaviour models, and propagation models. More information about each type of model is provided in Section 5. *ReputationModel*, *BehaviourModel* and *PropagationModel* are inherited classes from *EvaluationModels* and as such, they share a context (a string describing the context under which the model operates) and a list of entities that take part in the model. *EvaluationModel* also provides other methods, and their functionality will be delegated to lower layer classes, depending on the model type.

A reputation model adds a connector to an external database system to store reputation scores, and it holds the type of reputation model, which might be centralized or distributed. Moreover, this class exposes the method *updateReputation* which computes the reputation score and saves it in the trust database. A behaviour model contains a list of trust relationships and exposes methods to get and set these relationships. Finally, a propagation model, in addition to containing a list of trust relationships, it also contains a sequential operator and a parallel operator [1], and exposes a method to calculate indirect relationships.

4.2.2 Relational Layer

This layer contains the basic building blocks onto which the models of the upper layer are developed: entities and trust relationships.

Entities have a name, an automatically-generated identifier, a database connector and a trust metric. The fact that each entity holds a database connector enables distributed reputation systems, where each entity must store the reputation information regarding another entity in a personal database. Likewise, as each entity holds a trust metric instance, we allow each entity in the model to use a different trust metric to compute other entities' reputation.

Regarding trust relationships, they consist on a tuple that specify which is the entity that places trust (trustor), the entity on which trust is placed (trustee), the extent to which the trustor trusts the trustee (value), and the trust metric used to derive this value. Again, having the metric as an instance variable improves flexibility as each relationship could be measured with different metrics.

The decision that both an entity and a trust relationship may define their metrics supports the implementation of more advanced trust models where the final trust value that a trustor places on a given trustee might be determined by both the reputation of the trustee and the trust relationship between the trustor and the trustee.

[1] In the conceptual model we called them *concatenator* and *aggregator* operators. However, as we later implement a model where they are called *sequential* and *parallel*, we have adopted this notation for the architecture.

4.2.3 Computation Layer

Evaluation models rely on trust metrics to perform trust values calculations. This is the layer in charge of such computation.

Basically, *TrustMetric* is an interface that a developer should implement to override the *compute()* method, where the trust calculation takes place. Trust metrics use variables, through the class *Variable*, which have a name and a value, as well as methods to get and set these parameters. Operators for propagation models belong also to this layer.

Note that trust metrics contain instances of variables. As entities and trust relationships hold in turn instances of trust metrics, each entity or relationship might use different variables, increasing the flexibility of the framework to accommodate complex models.

4.2.4 User-Defined Layer

This layer is created as users extend the computation layer to accommodate their own definitions. As we explain in the next section, users can create new computation engines (implementations of the *TrustMetric* interface) and new variables to implement an important range of models. For illustration purposes, the architecture includes a summation engine (that basically sums up the variables that it contains) and a weighted summation engine (that adds a weight to each variable). The latter requires creating a specialized variable class that adds the weight to its internal state.

Up to now, we have described the framework from a structural point of view. The behavioural view of the architecture is further analyzed in the following section, where the framework is used to implement three evaluation models.

5 Instantiations of the Framework

In this section, we describe how the framework presented in Section 4 could address the implementation of three simple evaluation models. These models have been chosen because they are well-known as well as representative of the three types of evaluation models discussed earlier. In the first part of this section, we briefly describe the models to be implemented. In the second part, we actually use the framework to implement the models. The goal of this section is to analyze the feasibility of the framework to implement different evaluation models.

5.1 Models Description

5.1.1 Ebay Reputation Model [14]

Ebay [2] is probably the most famous auction-based online marketplace where buyers and sellers interact. Once a transaction has finished, buyers can evalu-

[2] www.ebay.com

ate sellers, expressing their satisfaction with regard to the transaction outcome. This evaluation is made by providing positive or negative feedbacks. The reputation score of a seller is computed by subtracting the negative feedbacks from the positive feedbacks. This model has its shortcomings, as expressed by Jøsang et al. [8], since people are usually reluctant to provide a negative evaluation and prefer to solve their problems off-line. Thus, the reputation score of a person is not very representative, since a person with 50 positive feedbacks and 10 negative feedbacks should be considered rather more untrustworthy than another one with only 40 positive feedbacks (although they would have the same reputation score under the model). However, it is an easy-to-understand model, therefore it has been chosen it for illustration purposes.

5.1.2 Risk and Utility-based Behaviour Model

In this made-up model a trustor determines his trust in a trustee by means of two factors: the risk and utility of the interaction. The higher the risk (as perceived by the trustor), the lower the trust. Likewise, the higher the utility (as perceived by the trustor), the higher the trust. This model can be considered an oversimplification of Marsh's computational model [10], where the author identifies many parameters that influence trust and combine them into different formulas. In the simplified version that we propose, the trustor performs the division between the utility and the risk to calculate the trust in the trustee.

5.1.3 Propagation Model

Agudo et al. [1] present a graph-based propagation model that allows to compute indirect trust relationships from direct ones. Let us suppose, in the context of the previous model, that an entity e_1 does not know the risk and utility values of interacting with an entity e_3. In this setting, there is not an explicit trust relationship between these two entities. However, let us suppose that we know the risk and utility values that e_1 hold for e_2, and the ones that e_2 hold for e_3 (that is, there is a trust relationship between e_1 and e_2, and between e_2 and e_3). Then, we could use the propagation model to compute the final trust value from e_1 to e_3, establishing a new trust relationship.

5.2 Using the Framework

5.2.1 Ebay Reputation Model

Let us suppose a distributed setting in which three entities (which do not know each other beforehand) have to perform several works. These entities can choose whether to execute a given work by themselves or to delegate it to the entity with the higher reputation. Each time an entity delegates a work, it registers a listener through which the delegatee informs about the outcome of the work (e.g. success or failure, time consumed, etc). Depending on these parameters, the delegator decides whether to place a positive feedback or a negative feedback on the delegatee. Thus, we are implementing the eBay reputation model onto another kind of application.

```
ReputationModel rm = new ReputationModel("Work Dispatching",
                3, CENTRALIZED, SUMMATION);
rm.addVariable("Positive Feedback", 0);
rm.addVariable("Negative Feedback", 0);
```

This simple code snippet creates a reputation model under the context “Work Dispatching”. The next parameter represents the number of entities (three in our example). Next, we specify that the reputation model is centralized (and not distributed), and this creates a database connector that acts as the interface to the reputation database (where reputation scores for the entities are stored). If the reputation model was distributed, there would be required to create a database connector for each entity created. The final parameter is the computation engine, which in this case is a simple built-in summation engine. As mentioned earlier, the framework is designed to allow different computation engines for different entities. By default, however, all entities share the same computation engines and the same variables.

After initializing the reputation model, we add the variables required, namely the positive and negative feedbacks, specifying their default values. The code that adds the variables is shown next:

```
public class ReputationModel {

  //If no entity is specified, when we add a
  //variable, it is added to the computation
  //engine of every entity
  public void addVariable(String name, Object value) {
        for (int i = 0; i < entities.size(); i++) {
                entities.get(i).getComputationEngine().addVariable(name, f);
        }
  }

  // ... (Other methods)
}
```

From this point onwards, the developer accesses the framework functionalities through the reputation model instance variable. The following code snippet shows the method to execute when the listener is triggered.

```
//This is the method the listener invokes when a work is finished
public void onWorkFinished(Work w, Entity delegatee, Message m,
        Time tConsumed) {

        Variable nFeedback = rm.getVariable("Negative Feedback", delegatee);
        Variable pFeedback = rm.getVariable("Positive Feedback", delegatee);

        if (m.isError() || tConsumed > threshold) {
                rm.setVariable("Negative Feedback", delegatee,
                        -(++nFeedback.getValue()));
        } else {
                rm.setVariable("Positive Feedback", delegatee,
                        ++pFeedback.getValue());
        }

        rm.updateReputation(delegatee);
}
```

Variables are updated depending on the outcome of the work dispatching, and there is a call to *updateReputation*. The code for this method is shown next:

```
public class ReputationModel {

  public void updateReputation (Entity e) {
    //connector is an instance variable of reputation model that allows
    //accessing a persistent database for storing reputation scores
        connector.updateEntry(e, e.getComputationEngine().compute());
        }

        // ...(Other methods)
}
```

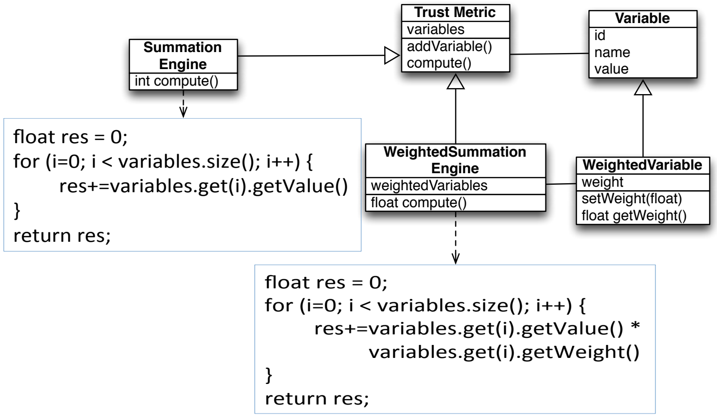
The *updateReputation* method will perform the computation of the reputation score according to the summation computation engine and the variables defined. Furthermore, it will update the central reputation database.

The summation engine overrides the *compute()* method of *TrustMetric*. This way, the framework provides enough flexibility to easily implement different metrics. Another metric could use weights to give a higher relevance to negative feedbacks, for instance. The developer would need to define two new classes: one extending *TrustMetric*, namely *WeightedSummation*, and another one extending *Variable*, namely *WeightedVariable*. The latter must contain the weight associated to the variable, whereas the former should override the method *compute()* of *TrustMetric*, as depicted in Figure 4.

In the code snippets from Figure 4., *variables.size()* equals two, since there are two variables (positive and negative feedbacks). The upper code represents the computation of the traditional eBay reputation model, whereas the other one describes a weighted version of it.

5.2.2 Risk and Utility-based Behaviour Model

In reputation models, according to the framework design, variables and computation engines belong to entities. That is, each entity has its own variables and computation engines. Now, an entity might hold a different risk and utility values for any other entity in the system. In order to support this, the framework introduces the class *TrustRelationship*, which encapsulates the information



```
float res = 0;
for (i=0; i < variables.size(); i++) {
       res+=variables.get(i).getValue()
}
return res;
```

```
float res = 0;
for (i=0; i < variables.size(); i++) {
       res+=variables.get(i).getValue() *
              variables.get(i).getWeight()
}
return res;
```

Figure 4: Creation of Metrics and Variables

regarding a trust relationship, namely the trustor, the trustee, the trust value, and the computation engine used to calculate the trust value. Thus, any trust relationship remains perfectly specified by an instance of this class.

Let us assume a setting with three entities again. The code snippet that the developer has to write is the following:

```
BehaviourModel bm = new BehaviourModel("Work Dispatching");

//If entities do not exist, they are created in the addTrustRelationship
// method. This process is tedious and could better done through
//configuration files or GUIs
bm.addVariable(bm.addTrustRelationship("e1", "e2"), "Risk", 0,3);
bm.addVariable(bm.addTrustRelationship("e1", "e2"), "Utility", 0,9);
bm.addVariable(bm.addTrustRelationship("e2", "e3"), "Risk", 0,6);
bm.addVariable(bm.addTrustRelationship("e2", "e3"), "Utility", 0,4);

// Set engine for all trust relationships
bm.setComputationEngine(riskUtilityEngine);
bm.compute(bm.getTrustRelationship("e1", "e2"));
bm.compute(bm.getTrustRelationship("e2", "e3"));
```

The computation engine that implements the trust metric, namely *riskUtilityEngine* must be defined by the developer, overriding the *compute()* method of *TrustMetric*. In our example, the engine would only need to retrieve the variable named *Risk* and divide it by the variable *Utility*.

Note that after the execution of the previous code, there is a trust relationship between e_1 and e_2, and between e_2 and e_3. We now proceed to use a propagation model to create a trust relationship between e_1 and e_3, as described next.

5.2.3 Propagation Model

The model proposed by Agudo et al. [1] uses a *sequential operator* to compute the trust value between two entities in a given trust chain, and a *parallel operator* to compute a global trust value between two entities linked by multiple trust chains. The same concepts are used in other trust models, such as Jøsang's belief model [7], which disseminates opinions through *discounting* and *consensus* operators.

Consider the same example of the previous model. Let us assume that the developer wants to compute a trust value between e_1 and e_3. The code he should write is the following:

```
PropagationModel pm = new PropagationModel(bm, MIN, MAX);
pm.calculateIndirectTrust("e1", "e3");
```

In this example, we have chosen the built-in minimun and maximum functions as sequential and parallel operators, respectively. However, as there is only one trust path from e_1 to e_3, the parallel operator will not be applied.

Also note that the first argument of the constructor is the instance of the previous model (bm), since the propagation model needs access to the trust relationships previously defined.

The method *calculateIndirectTrust* is described next:

```
public void calculateIndirectTrust(Entity e1, Entity e3) {
  float[] values;
  TrustRelationship[] trustChains = retrieveTrustChains(tr, e1, e3);
  for (int i = 0; i < trustChains.size(); i++) {
        values[i] = sequentialOp.compute(trustChains[i]);
  }
  tr.addTrustRelationship (e1, e3, parallelOp.compute(values));
}
```

The *sequentialOp* and *parallelOp* are instance variables that store an instance of the operators, which might be defined by the user extending the corresponding abstract classes. Basically, the method retrieves all the trust paths between e_1 and e_3. Then, it applies the sequential operator to every path, and finally, it applies the parallel operator to the obtained values.

6 Conclusion and Future Work

There is a huge amount of different trust models proposed in the literature. These models, however, are often designed to work in ad-hoc environments, and to be plugged into already-existing applications. In this paper, we have presented an object-oriented development framework to assist developers during the implementation of applications that might require support from trust or reputation models. As the application is developed using the framework, trust models are aligned with the design of the application and they can exploit all the data available to the application.

In order to achieve this, we have performed a domain analysis of trust models by means of knowledge graphs. This analysis has helped us to elicit the requirements that the framework should meet, and also to identify several classes and methods, from which we designed a first version of the architecture. Finally, we have proved the feasibility of the framework by giving guidelines on the implementation of three different evaluation models: the eBay reputation model, an oversimplified version of Marsh's model, and a propagation model.

As future work, we intend to extend the framework in order to accommodate several models features that are often found in the literature. First, we are interested in supporting the implementation of models where the trust values are represented by a tuple of values (multiple dimensions) rather than by a single value. We intend to allow defining a different metric for each dimension in order to provide greater flexibility.

Some trust models yield an uncertainty value together with the trust value, in order to inform other entities about how certain the trust value should be considered. We plan to add support for this feature as well. Also, roles played by entities or the membership of entities to a given group are factors taken into account in other models to determine trust, and therefore we intend to include this feature in the near future too.

Finally, we aim to add more complex built-in computation engines, including beta-probability distributions and fuzzy engines.

Acknowledgements

This work has been partially funded by the European Commission through the FP7 project NESSoS under grant agreement number 256980, and by the Spanish Ministry of Science and Innovation through the research projects ARES (CSD2007-00004) and SPRINT (TIN2009-09237). The first author is funded by the Spanish Ministry of Education through the National F.P.U. Program.

References

[1] Isaac Agudo, Carmen Fernandez-Gago, and Javier Lopez. A model for trust metrics analysis. In *5th International Conference on Trust, Privacy and Security in Digital Business (TrustBus'08)*, volume 5185 of *LNCS*, pages 28–37. Springer, 2008.

[2] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *IEEE Symposium on Security and Privacy*, pages 164–173, 1996.

[3] Vinny Cahill, Elizabeth Gray, Jean-Marc Seigneur, Christian D. Jensen, Yong Chen, Brian Shand, Nathan Dimmock, Andy Twigg, Jean Bacon, Colin English, Waleed Wagealla, Sotirios Terzis, Paddy Nixon, Giovanna di Marzo Serugendo, Ciaran Bryce, Marco Carbone, Karl Krukow, and

Mogens Nielsen. Using trust for secure collaboration in uncertain environments. *IEEE Pervasive Computing*, 2(3):52–61, July 2003.

[4] Diego Gambetta. Can we trust trust? In *Trust: Making and Breaking Cooperative Relations*, pages 213–237. Basil Blackwell, 1988.

[5] Chern Har Yew. *Architecture Supporting Computational Trust Formation.* PhD thesis, University of Western Ontario, London, Ontario, 2011.

[6] TD Huynh. A Personalized Framework for Trust Assessment. *ACM Symposioum on Applied Computing - Trust, Reputation, Evidence and other Collaboration Know-how Track*, 2:1302–1307, December 2008.

[7] Audun Jøsang. A logic for uncertain probabilities. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 9(3):279–311, June 2001.

[8] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618–644, March 2007.

[9] Adam J. Lee, Marianne Winslett, and Kenneth J. Perano. Trustbuilder2: A reconfigurable framework for trust negotiation. In Elena Ferrari, Ninghui Li, Elisa Bertino, and YÃ¼cel Karabulut, editors, *IFIPTM*, volume 300 of *IFIP Conference Proceedings*, pages 176–195. Springer, 2009.

[10] Stephen Marsh. *Formalising Trust as a Computational Concept.* PhD thesis, University of Stirling, April 1994.

[11] D. Harrison McKnight and Norman L. Chervany. The meanings of trust. Technical report, University of Minnesota, Management Information Systems Research Center, 1996.

[12] Keith W Miller, Jeffrey Voas, and Phil Laplante. In Trust We Trust. *Computer*, 43:85–87, 2010.

[13] Francisco Moyano, Carmen Fernandez-Gago, and Javier Lopez. A conceptual framework for trust models. In *9th International Conference on Trust, Privacy & Security in Digital Business (TrustBus 2012)*, September 2012 In Press.

[14] Paul Resnick and Richard Zeckhauser. Trust among strangers in Internet transactions: Empirical analysis of eBay's reputation system. In Michael R. Baye, editor, *The Economics of the Internet and E-Commerce*, volume 11 of *Advances in Applied Microeconomics*, pages 127–157. Elsevier Science, 2002.

[15] Florian Siefert Gerrit Anders Theo Ungerer Wolfgang Reif Rolf Kiefhaber. The Trust-Enabling Middleware: Introduction and Application. Technical report, Institut fuÌˆr Informatik UniversitaÌˆt Augsburg, March 2011.

[16] Sini Ruohomaa and Lea Kutvonen. Trust management survey. In *Proceedings of the Third international conference on Trust Management*, iTrust'05, pages 77–92, Berlin, Heidelberg, 2005. Springer-Verlag.

[17] Girish Suryanarayana, Mamadou H. Diallo, Justin R. Erenkrantz, and Richard N. Taylor. Architectural Support for Trust Models in Decentralized Applications. In *Proceeding of the 28th international conference*, pages 52–61, New York, New York, USA, 2006. ACM Press.