On Designing Usable Policy Languages for Declarative Trust Aggregation

Michael Huth and Jim Huan-Pu Kuo

Department of Computing, Imperial College London
London, SW7 2AZ, United Kingdom
{m.huth, jimhkuo}@imperial.ac.uk

Abstract. We argue that there will be an increasing future need for the design and implementation of declarative languages that can aggregate trust evidence and therefore inform the decision making of IT systems at run-time. We first present requirements for such languages. Then we discuss an instance of such a language, Peal$^+$, which extends an early prototype Peal that was researched by others in collaboration with us. Next, we formulate the intuitive semantics of Peal$^+$, present a simple use case of it, and evaluate to what extent Peal$^+$ meets our formulated requirements. In this evaluation, particular attention is given to the usability aspects of declarative languages that mean to aggregate trust evidence.

1 Introduction

There is little doubt that the advances in computing and information technology are transforming the manner in which we conduct our business and lead our personal lives. The use of small devices such as tablets and smart phones, the rapid pace with which such technologies evolve, and the increased reach of these technologies – to name smart meters for electric power supply – are prominent examples of this.

One consequence of this is that ever more things have programmable interfaces to which other things and processes may connect. In this *Internet of Things*, designers, programmers, and users alike need to be able to formulate constraints on the interactions across such interfaces that adequately reflect implicit trust assumptions, risk appetite, and other intentions. We think that trust management will play a key role in the articulation of such interaction constraints. Let us first state what we mean by the term "trust" in this paper. We say that an agent A (a program, a user, a system, etc.) trusts another agent B for a planned interaction I with B, when agent A has collected and inspected evidence that leads agent A to believe that engaging in the interaction I with B is worth taking any risk reflected in the studied evidence. This asymmetric view of trust can be made symmetric by letting agent B perform a similar inspection and decision process regarding the interaction I with agent A – typically based on evidence pertinent to agent B. If we think of the symmetric view as a *logical and* of the asymmetric view (a trust-mediated interaction would only take place if both agents agree to it), we can focus on the asymmetric view subsequently.

In this paper, we investigate how declarative languages can help with formalizing the process of collecting and studying indicators of trust in context-dependent interactions with other agents. We posit that such languages will be increasingly needed, and formulate requirements that they should satisfy. For sake of illustration, we will study an extension of a trust-aggregation language that we have designed with others [8, 2, 7, 9] and assess its suitability against the requirements we will formulate further below. This will in part make use of a toy example written in that language. The paper concludes by identifying future work for the design of more usable trust aggregation languages.

2 Declarative trust aggregation through policies

We refer to *trust aggregation* as a process in which agent A first collects observable indicators of trust in an interaction I with agent B, and then systematically combines such indicators to more compact or abstract expressions that can directly inform the decision making of agent A (e.g. whether or not to commit to interaction I with agent B). Note that observable indicators may also be estimates, for example, the estimated uncertainty in the computed reputation of an agent. Such indicators may themselves be talking about perceived trust, reputation scores, risk levels, or about things that influence trust perceptions indirectly – for example the financial risk to agent A in interaction I with agent B. We note that existing approaches to computing trust or reputation scores (including those that have only binary scores as in "trust" or "don't trust") acknowledge that such computations benefit from incorporating context-dependent information in the aggregation of trust evidence. For example, reputation systems for online trading sites that base reputation scores on the number of successful past interactions without taking into account the monetary values of these transactions are subject to active attacks. Such attacks can indeed be prevented or mitigated against by making reputation scores dependent on transaction values as well – see for example the nice discussion in [11].

We posit that the future Internet of Things will have an increased need for using such trust aggregations at many interfaces, and that this creates the need for a sort of *trust calculus* as the basis of computing the perceived trust in interactions across interfaces, which can then be enforced at run-time. No doubt will this lead to many dialects or variants of such a calculus. But there is evolutionary pressure to standardize such aggregation languages in order to get portability across platforms and technologies. Additionally, the creation of a more generic trust calculus will facilitate the development of robust analyses of aggregations formulated in such a calculus. It will also allow the decoupling of executable and analyzable such core languages from user-facing and domain-specific languages for expressing trust aggregation. With such a separation of concerns, one would for example only have to compile user-facing domain-specific languages into a (not user-facing) core language that can be implemented in systems and for which the desired analyses can be performed.

Such a core trust calculus has to be able to collect indicators of trust that have different semantic types, for example, the location of an agent, the past interaction history with agent B, and the nature of the intended next interaction with B. This need adds cognitive complexity to the aggregation of such indicators. We believe that such cognitive complexity cannot be eliminated by formal foundations of a core calculus and its aggregation mechanisms, even though such foundations will have many other benefits discussed further below. For example, the interaction of indicators of trust with indicators of distrust may be non-obvious or unintuitive and so harder to understand. In mathematical terms, the introduction of distrust indicators moves from simpler semi-rings of values to rings in which negative and positive information gets combined.

Many reputation systems and related approaches hardwire the aggregation of indicators and its possible state-change semantics through mathematical formulas that express scoring functions. This allows for easy implementations and supports the formal analysis of the mathematical system being described. However, we claim that future IT systems will require more expressive and more adaptable mechanisms for specifying, implementing, and analyzing such evidence aggregation and state-transformation mechanisms. For example, the management of trust and risk across different logical, physical or legal domains can no longer rely on scoring functions that implicitly assume a closed system with observable boundaries. There are also systems for which not enough prior information might be available and yet some form of trust evidence is present and perhaps the only basis for decision making. An example thereof are processes by which parties of arms reduction treaties can confirm to each other that specific arms have indeed been destroyed.

We propose to use *policy languages* to attain such declarative flexibility and adaptability. Policy languages have already been used successfully in trust management (e.g. for public-key systems [1]) and access control (see e.g. [3]). Such languages have often a means of composing simple rules into global policies, something we feel to be desirable from a usability perspective.

3 Requirements for declarative trust aggregation

We now describe requirements that we deem to be important for the design of policy languages $\mathcal{L}$ for declarative trust aggregation:

Expressiveness: Such a language $\mathcal{L}$ needs to be able to declare aggregations and state changes as they occur or are needed in a wide range of real systems, at least up to an acceptable degree of abstraction.

Scalable analysis: Declarations made in such a language $\mathcal{L}$ should be subject to formal analysis that aids in the validation of these declarations, and such analyses should scale up to realistic declaration sizes.

Interface-facing: Such a language $\mathcal{L}$ should be designed so that it can interface easily with other languages down-stream (by using those other languages) and up-stream (by being used by other languages). Down-stream, we need to be able to plug into $\mathcal{L}$ desired expressions from other languages that specify indicators of trust, distrust, risk, etc.. Up-stream, we want to be able to take $\mathcal{L}$ expressions that declare trust aggregation or state-changes based on such aggregation, and plug them into other languages (e.g. as conditions) to be used for decision making or further computation.

Usability: Declarations made in $\mathcal{L}$ should be easy to formulate, should support the intuitions of a specifier, and should also have easily specifiable and intuitive analyses used to validate such declarations. Equally, the feedback provided by analyses should be easy to understand and be presented at the same cognitive layer as the analyzed declarations. This may require that syntactic patterns of language fragments of a core calculus be identified (so called embedded domain-specific languages). Or it may require the development of application-specific user-facing languages that have efficient and transparent translations into a core language $\mathcal{L}$. Language choices such as "embedded" versus "compiled" will typically be made based on external factors and will therefore vary.

4 A core language for trust aggregation

We now sketch a language Peal^+ that may serve as a core calculus for trust aggregation. This language supports aggregation but not yet declarations of state changes based on computed trust. We leave such aspects to future work. The language Peal^+ is depicted in Figure 1. Its most abstract expressions are conditions *cond*, which are Boolean combinations of two sorts of formulas: predicates q as indicators of trust, risk, value, etc., and inequalities $pSet1 \leq pSet2$ which declare that the score computed by a policy set $pSet1$ is not larger than the score computed by policy set $pSet2$. Note that we can derive language expressions $cond \,||\, cond$ (for disjunction) and $pSet < pSet$ (for strict inequality of scores) as syntactic sugar of Peal^+ since the latter contains the dual constructors conjunction for conditions (&&) and less-than-or-equal comparison for policy sets ($\leq$), as well as the negation operator for conditions ($\neg$).

In language Peal^+, the language of predicates q itself is left unspecified. This omission is intentional as this is how we want to ensure that we can interface with down-stream languages that may express such indicators in whichever way. The decision to use Boolean variables means we require that interfaces to such other languages render their trust indicators in Boolean form. Examples of such a down-stream languages would be first-order or higher-order logic, where predicates q would be defined or bound to formulas of such logics.

In Peal^+, predicates are used to build rules, rules are used to build policies, and policies are used to build policy sets. Finally, predicates and policy sets are used to build conditions. These conditions can then be used as stand-alone expressions to support decision making, or they may serve as Boolean expressions

$$
\begin{aligned}
op &::= min \mid max \mid + \mid * \\
raw\_score &::= real\_const \mid real\_var \mid real\_const * real\_var \\
score &::= raw\_score \mid raw\_score\ [real\_const\,,\ real\_const] \\
rule &::= if\ (cond)\ score \\
pol &::= op\,(rule^*)\ default\ score \\
pSet &::= score \mid pol \mid op\,(pSet,\ pSet) \\
cond &::= q \mid pSet \leq pSet \mid \neg cond \mid cond\,\&\&\,cond
\end{aligned}
$$

Fig. 1. Syntax of Peal^+ where q ranges over some language of predicates, and the constants and variables in *score* range over real numbers (potentially restricted by domains or analysis methods)

in up-stream languages. A condition such as $0.5 < pSet$ might model whether or not there is sufficient trust in committing to a risky interaction, where 0.5 acts as a strict trust threshold and $pSet$ captures the aggregation of trust evidence. Note that the language Peal^+ does not explicitly assign such conditions to specific agents. Such designations would indeed be expected to happen in an up-stream language that were to use such conditions to regulate and enforce trust-mediated interactions in a multi-agent system. Note further that conditions *cond* might also appeal to agents and their states through predicates q that have meaning in a suitable logic, and so down-stream languages may also reflect agency if needed.

Policy sets of Peal^+ are either atomic, in which case they are scores or policies, or they are composite objects, in which case they are recursively composed from policy sets through composition operators listed under syntactic clause *op*. The composition choices for *op* supported in *core* are minimum, maximum, addition and multiplication of scores – but one can well image extensions of this.

Let us now discuss the two forms of atomic policy sets, beginning with scores. A score is defined to be either a raw score or a raw score annotated with a real interval. A raw score is either a real constant, a real variable, or the product of a real constant with a real variable. For example 0.56, y_2, and $-1.4 * z$ could all be declared as raw scores and so as scores as well. The annotation of an interval allows us to write expressions such as $0.456 * x\ [-0.1, 0.2]$ as a score. The role of the interval $[-0.1, 0.2]$ is to express *non-deterministic uncertainty* in the value of $0.456 * x$. The intuition is that, no matter what value x has, the value of the score will be in the set $\{0.456 * x + u \mid -0.1 \leq u \leq 0.2\}$. One advantage of having such an annotation is that it allows the specifier to make uncertainty in the true value of the score explicit (said value may be a best-effort estimate of a probability, for example). And the analysis of conditions *cond* written in Peal^+ can then reflect such non-deterministic choices, and may so validate conditions so that they are robust under any sensitivity changes within the ranges of these declared intervals.

It remains to explain the syntax for declaring policies and rules. A rule *if* (*cond*) *score* evaluates a condition *cond*. If the latter is true, the rule evaluates to the value of *score*; otherwise, the rule does not evaluate to anything. A policy consists of zero or more rules, a composition operator for rules (ranging over the same operators as for composition of policy sets), and the specification of a default score:

$$p_i = op\,(\mathit{if}\ (c_1\ s_1) \ldots \mathit{if}\ (c_n\ s_n))\ \mathit{default}\ s \qquad \text{or} \qquad p_i = op\,()\ \mathit{default}\ s \quad (1)$$

The intuitive semantics of a policy p_i as in (1) in Peal^+ is then as follows. First, determine which rules in policy p_i have a true condition c_i: this is the set $X = \{s_i \mid c_i \text{ true and occurs in } p_i\}$. Second, if X is non-empty, return $op(X)$; otherwise, return default score s as meaning of p_i. This semantics requires that we can reliably determine the truth values of all predicates within a policy.

The language Peal^+ is an extension of the language Peal which was developed in the papers [8, 2, 7, 9]. Let us therefore quickly state what new features Peal^+ contains over the version of Peal described in [7, 9]. In Peal^+, composition operators are now unified in that they are the same for policies and policy sets, meaning that we now can also combine policy sets with addition and multiplication. Scores in Peal^+ may not just be raw scores but may be annotated with a constant real-valued interval, and scores can now also be casted into policy sets. Finally, the condition expressions $score < pSet$ and $pSet \leq score$ of Peal are generalized in Peal^+ to $pSet1 < pSet2$ and $pSet1 \leq pSet2$. The justification for such extensions from Peal to Peal^+ is that it allows for more expressive score calculations, for the modelling of score uncertainty, but at the same time won't complicate much the *symbolic* generation of analysis code in the Z3 SMT solver as described for Peal in detail in [7, 9].

Let us see how Peal^+ can be used to plug into up-stream languages. For example, an expression of form $0.9 < min(pSet1, pSet2)$ is a Boolean condition in which 0.9 acts as a strict threshold for the score of a declared policy set, where the *min* composition means that the scores of both policy sets $pSet1$ and $pSet2$ have to be above 0.9. This condition may aggregate evidence for trusting a request to some resource and were we combine two policy sets in a conservative manner as both policy sets have to attain sufficient evidence for the threshold constraint $0.9 < x$. If we plug this condition into an access-control language that supports rules such as `grant if (cond) else deny`, then using $0.9 < min(pSet1, pSet2)$ in place of `cond` articulates the circumstances under which access would be granted. Note that conditions in Peal^+ are agnostic as to whether or not they support positive (as in this example) or negative decisions. This depends on the up-stream context into which such conditions are placed, and this is a potential usability issues.

5 Usability issues of Peal^+

We provide a small example of condition declarations in the language Peal^+. The example captures a fictional setting in which a car rental company might assess

the trust it places in US rental agreements within a variety of contexts, and it shows that trust declarations may have ethical or legal dimensions as well.

Example 1. The policies, policy sets, and conditions for this example are depicted in Figure 2, using concrete syntax very close to that of the tool PEALT that implements language Peal [9]. These declarations specify four policies:

- Policy `b1` classifies the type of car to be rented, and associates with it a monetary value, where the default value is higher than that of a compact car. The composition operator is maximum here.
- Policy `b2` classifies the driver who wishes to rent the car by assigning a trust score based on the country of origin of the driver's license. US Licences are trusted more than European ones, and European ones are trusted somewhat more than UK ones (as Europeans and Americans drive on the same side of the road). Licences from other parts of the world are trusted less and there is uncertainty about their trustworthiness coded in the interval $[-0.1, 0.1]$. Drivers with no licence are not trusted at all (default of `0`). The composition operator is here minimum (seeking the least trust).
- Policy `b3` classifies the risk of the car rental in terms of the type of intended car usage: there is the highest risk if some off road driving is planned, followed by city driving as the next highest risk, whereas long distance driving has the lowest risk (lower than the default risk of `0.3`). A mixed usage of long distance and city driving has an intermediate risk associated with it. The composition operator is maximum (going for the highest risk).
- Finally, policy `b4` accumulates evidence for trusting to rent out the car, based on evidence aggregated from driver information: a trust score that is linear in the number x of years driven accident-free within the past years from now is one source, as is the indication of being able to speak English (e.g. so that road signs can be read and understood), and the fact that the driver is female. Note that negative trust evidence is included when the driver would travel alone. The composition operator is addition here, accumulating trust and distrust.

These policies are composed into policy sets in condition $\mathtt{c_1}$, where we take the asset value of the case in `b1` and multiply this with the perceived risk – which is the trust score of `b2` "inverted" to $\mathtt{1 - b2}$ in order to capture such risk:

- Condition `c0` limits the credit for number of years driven without accidents to 10 and forces x to be non-negative.
- Condition `c1` stipulates that this weighted risk be no larger than $\mathtt{50,000}$.
- Condition `c2`, on the other hand, specifies that the accumulative trust evidence collected about the driver be strictly larger than 0.4.
- The next three conditions express, using propositional connectives, that the events listed in the three respective policies `b1` up to `b3` are mutually exclusive (but not necessarily *across* such policies).
- Condition `c5` captures a logical constraint (company policy), that no luxury car is rented out if the intended usage includes some off road driving.

– Finally, the condition for trusting the rental arrangement (from the point of view of the rental company) is expressed in condition `cond`, that specifies that all seven conditions already discussed have to be met.

```
b1 = max ((isLuxuryCar) 150,000) (isSedan 60,000)
         (isCompact 30,000)) default 50,000
b2 = min ((hasUSLicense 0.9) (hasUKLicense 0.6)
         (hasEULicence 0.7) (hasOtherLicense 0.4 [-0.1,0.1]) default 0
b3 = max ((someOffRoadDriving 0.8) (OnlyCityUsage 0.4)
         (onlyLongDistance 0.2) (mixedUsage 0.25)) default 0.3
b4 = + ((accidentFreeForYears 0.05*x) (speaksEnglish 0.05)
       (travelsAlone -0.2) (femaleDriver 0.1)) default 0
c0 = (0 <= x <= 10)
c1 = (* b1 (+ b2 (-1))) <= 50,000
c2 = 0.4 < b4
c3 = "all events in b1 are mutually exclusive"
c4 = "all events in b2 are mutually exclusive"
c5 = "all events in b3 are mutually exclusive"
c6 = !isLuxuryCar || !someOffRoadDriving
cond = c0 && c1 && c2 && c3 && c4 && c5 && c6
```

Fig. 2. Declarations in Peal$^+$ that specify criteria for a car rental company to trust renting out cars in certain usage scenarios.

Let us now illustrate usability issues of trust aggregation languages, by appealing to the above example and its use of the language Peal$^+$ when and where appropriate. One concern is the intuitive meaning and appropriateness of composition operators *op*. In policy `b1`, for example, the operator is the maximum. This conveys a false sense of purpose for this composition, as condition `c3` stipulates that all events within that policy are mutually exclusive. So an operator such as *sole* (which would return the score of the only true event or the default score if no or more than one event were true) may seem more intuitive. In fact, operator *sole* would also be usable for policy `b2`. Although it is interesting to note that policy `b1` uses maximum as it conservatively wants to estimate the value of assets under risk, whereas policy `b2` uses minimum as a conservative estimate of a trust score.

Another potential problem with policy `b1` is that the default score is not smaller than all scores within the policy body. This means that the policy is *not monotone*: all its events might be false, but when we then make more events true by making just `isCompact` true, the score of the policy *de*creases. This might be intended by the specifier but it could lead to "attacks" of these specifications by which conditions for trust could be made true by making some events false. Similarly, one might hide attributes in attribute-based access control to get unintended access. The presence of such attacks can be statically analyzed.

For example, for the attribute-based language PTaCL [3] a tool ATRAP was developed in [6] that automatically searches for such attacks and – in their absence – constructs a formal proof of their absence.

Furthermore, language Peal$^+$ does not contain types or similar annotations that might indicate whether policies, policy sets or conditions intent to express risk, trust, monetary values or any other modality. For example, we might expect that risk and trust are inversely proportionate. Similarly, the language does not say whether these modalities are specified with a pessimistic, optimistic, averaging or some other cognitive stance.

Language Peal$^+$ also has a simple but implicit scoping: there are no syntactic blocks that can rebind declared names of predicates, policies, etc. Such names refer to the same entities in all declared conditions. We think that the introduction of local names and their static scoping would introduce unwanted cognitive complexity to using Peal$^+$. On the other hand, the language does not have a direct means of defining condition names that contain parameter headers. For example, condition `c1` has policies `b1` and `b2` as parameters and so it would be convenient to write $\texttt{c1}(\texttt{b1}, \texttt{b2})$ and to be able to replace formal parameters `b1` and `b2` with actual parameters in other condition expressions.

Another interesting usability issue is the fact that rules may contain complex conditions *cond* and not just predicates q. This may mostly just be for convenience so that predicates q can reflect their propositional logical structure explicitly in Peal$^+$ as opposed to through an interface to a down-stream language. But expressions *cond* used in rules may themselves talk about policy sets. This allows richer aggregation mechanisms, yet it also introduces an apparent circularity: consider policy $\texttt{b1} = (\texttt{if}\,(\texttt{c1})\,0.3)\,\texttt{default}\,0$ and condition $\texttt{c1} = 0.2 < \texttt{max}(\texttt{b1}, \texttt{b2})$ for some policy `b2`. The meaning of `b1` (its score) depends on the meaning of `c1` (a truth value), which in turn depends on the score of `b1`. Fortunately, this is not a genuine circularity as it merely constrains the possible truth values of `c1` and scores of `b1` in analyses. However, a user-facing language may want to prevent or flag up such circularities as they are most likely due to typos or reflect unintended consequences.

A general usability issue of languages such as Peal$^+$ is how we aid specifiers in validating that the conditions they express in these languages reflect the intentions that they have in managing assets, risks, reputations, and trust. We believe that specifiers should be able to subject conditions to a variety of automated analyses that can boost their confidence in that intentions have been met in specified conditions. In [2] such analyses were proposed, and some of these analyses were implemented in the tool PEALT [9] for the smaller language Peal. For example, in PEALT one can ask whether a condition is always true or always false – both would typically indicate that intentions are not met; one can ask whether a condition of form $score < pSet$ changes when *score* is changed by a specified value; etc. These analyses are rendered as push-button technology through automated translation of conditions and the desired analysis to code for the SMT solver Z3, where the execution of that code performs that analysis and gives feedback. We believe that this automated means of performing anal-

yses and getting their feedback is crucial for gaining acceptance for the use of trust-aggregation languages in real systems.

6 Evaluation of Peal$^+$ against remaining requirements

We now assess to what extent our language Peal$^+$ meets the requirements we formulated above, and how these requirements interact with usability issues. We begin with **Expressiveness**. Language Peal$^+$ is certainly very expressive in that predicates q may provide plugs to very rich languages for providing the exact meaning of such predicates. Conditions have intuitive structure: propositional logic over the input language for predicates plus the comparison of policy sets. A source of cognitive complexity is whether we understand a comparison to be true in all scenarios or to be true in at least one scenario. For example, when we write $pSet1 \leq pSet2$ do we mean that the score of the first expression is always no larger than that of the second expression or that is can be no larger? Answering such questions depends on how such conditions are used in up-stream languages (or even in Peal$^+$ conditions). Again, analyses can be used to provide needed sanity checks that intended usage of conditions matches their semantics. Additionally, user-facing languages that compile into Peal$^+$ could be designed in which patterns and types make clear the intentional stances of policies and their composition (e.g. whether a policy aggregates trust scores, asset values, etc.). Use of such patterns would be expected to prevent a lot of misinterpretations that would therefore not be flagged up in analyses and so reduce the number of "condition refinement steps".

The stratification of policy sets into rules, policies, and policy sets should help with structuring more complex aggregation meachanisms. The structure of rules seems intuitive enough, but one may object to its behavior when its condition is false. For example, one might want a rule that says "if q, then 0.9 else 0.1" for expressing trust, suggesting that q indicates trust whereas $\neg q$ indicates distrust. But this is not a good language primitive as not all trust indicators suggest distrust in their absence. Moreover, we can build expressions such as the above as a policy $op\,(if\ \ (q)\,0.9)\,default\,0.1$.

Let us discuss **Scalable Analysis** next. In [9], we showed experimentally that language Peal allows analyses of fairly large conditions (with hundreds or thousands of rules and policies) within seconds or minutes, where the marked bottleneck is an extensive use of multiplication in policies. Given the specifics of symbolic code generation for these analyses in the Z3 SMT solver, we anticipate that similar scalability will be achievable for the richer language Peal$^+$, and we plan to investigate this in future work. A nice aspect of using back-ends such as Z3 is that this approach will benefit from whatever future optimizations or marked improvements will be made in SMT solving.

One important usability aspect of these analyses is that their output consists of the description of a scenario (some true predicates, some false predicates, and values of variables that support such truth values). We are currently developing techniques for the independent verification of the correctness of such output. In

general, this is needed because the method of code generation for analysis may be flawed or because the reasoning about real numbers in some back-end tool may be imprecise. By correctness of computed output we mean that the reported information is statically sufficient for explaining that the conditions supplied as arguments to an analysis have the claimed truth values. Verification of this claim involves the solution of 2-person games and fairly simple static analyses of policy expressions. It would be interesting to investigate how this algorithmic certification of correctness could be communicated to users in a form that goes beyond *"Independent verification of analysis outputs was successful"* but renders the insights of this verification process in an abstract yet still more informative form – by hiding some of the complexity of that verification process.

As for **Interfacing**, we think that the addition of parameterized headers to Peal^+ would help in defining clear interfaces to down-stream and up-stream languages. In Peal^+, we don't explicitly manage name spaces across domains. But existing naming convention could enforce globally unique names for predicates and variables within Peal^+. Also, Peal^+ can use any down-stream language that returns Booleans as predicates q; and Peal^+ can plug into any up-stream language that expects real values (for policy sets $pSet$) or Booleans (for conditions $cond$).

7 Related Work

We refer to the extant literature, see for example [13] and [4], for a more thorough discussion of trust mechanisms and their role in general system design. Empirical work done by social scientists in the general space of trust perceptions and its support in decision making is an important source of information for the design of user-facing trust aggregation languages.

In [5], it was studied how software engineers evaluate the trustworthiness of software components and how they decide to use such components in their software development. It was shown that these technical people used the same socio-cognitive processes as non-technical ones and also employ a "leap of faith" in which their trust decision may not reflect their trust evaluation. Interestingly, the decision to trust was *negatively* impacted by contact with component developers (since such contact was often the result of problems in said component).

The book chapter [14] recalls that a lot of research focussed on trust symbols (e.g. on web sites) that may influence the trust perceptions of users but that the effectiveness of such trust signals cannot be empirically validated [10]. In fact, technical systems need to consider trustworthiness of services already at the design phase of such systems. They also stress the need to move from mere trust symbols to trust *symptons* that can form the basis of trust assessment heuristics. In that context, we point out that the credit card industry has a history of using and modifying statistics that form the basis of so called *score cards* with which the creditworthiness of an applicant is evaluated against a history of past clients and their attributes and performance. For example, [12] studies how one might account for "population drift" in consumer credit classification – something of great importance in times of high migration and fast societal changes. There-

fore, there could be of interest to investigate whether that research in statistics may offer insights in the design of trust assessment heuristics for executable IT systems.

Acknowledgements: We are grateful that this work was supported by funding from Intel® Corporation within its *Trust Evidence* research project.

References

1. Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis. Keynote: Trust management for public-key infrastructures (position paper). In *Security Protocols Workshop*, pages 59–63, 1998.
2. Jason Crampton, Michael Huth, and Charles Morisset. Policy-based access control from numerical evidence. Technical Report 2013/6, Imperial College London, Department of Computing, October 2013. ISSN 1469-4166 (Print).
3. Jason Crampton and Charles Morisset. Ptacl: A language for attribute-based access control in open systems. In *POST*, pages 390–409, 2012.
4. Ivan Flechais, Jens Riegelsberger, and M. Angela Sasse. Divide and conquer: the role of trust and assurance in the design of secure socio-technical systems. In *Proceedings of the 2005 workshop on New security paradigms*, NSPW '05, pages 33–41, New York, NY, USA, 2005. ACM.
5. Andrew J. B. Fugard, Elke Beck, and Magdalena Gärtner. How will software engineers of the internet of things reason about trust? In *AmI Workshops*, volume 277 of *Constructing Ambient Intelligence, Communications in Computer and Information Science*, pages 274–279. Springer, 2012.
6. Andreas Griesmayer and Charles Morisset. Automated certification of authorisation policy resistance. In *ESORICS*, pages 574–591, 2013.
7. Michael Huth and Jim Huan-Pu Kuo. PEALT: A reasoning tool for numerical aggregation of trust evidence. Technical Report 2013/7, Imperial College London, Department of Computing, 2013. ISSN 1469-4166 (Print).
8. Michael Huth and Jim Huan-Pu Kuo. Towards verifiable trust management for software execution - (extended abstract). In *Proc. of TRUST*, pages 275–276, 2013.
9. Michael Huth and Jim Huan-Pu Kuo. Pealt: An automated reasoning tool for numerical aggregation of trust evidence. In *Proc. of 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2014)*, Lecture Notes in Computer Science (ARCoSS). Springer, 2014. To appear.
10. Iacovos Kirlappos, Martina Angela Sasse, and Nigel Harvey. Why trust seals don't work: A study of user perceptions and behavior. In *TRUST*, Lecture Notes in Computer Science, pages 308–324. Springer, 2012.
11. Lik Mui. *Computational Models of Trust and Reputation: Agents, Evolutionary Games, and Social Networks*. PhD thesis, MIT, 2002.
12. Nicos G. Pavlidis, Dimitris K. Tasoulis, Niall M. Adams, and David J. Hand. Adaptive consumer credit classification. *Journal of the Operational Research Society*, 63(12):1645–1654, 2012.
13. Jens Riegelsberger, Martina Angela Sasse, and John D. McCarthy. The mechanics of trust: A framework for research and design. *Int. J. Hum.-Comput. Stud.*, 62(3):381–422, 2005.
14. Angela Sasse and Iacovos Kirlappos. *Trust, Computing, and Society*, chapter Design for trusted and truthworthy services: why we must do better. Cambridge University Press, 2014. In press.