A Constraint framework for the qualitative analysis of dependability goals: Integrity

Stefano Bistarelli[1,2*] and Simon N. Foley[3**]

[1] Dipartimento di Scienze, Università "G. D'Annunzio" di Chieti-Pescara, Italy
bista@sci.unich.it
[2] Istituto di Informatica e Telematica, CNR, Pisa, Italy
stefano.bistarelli@iit.cnr.it
[3] Department of Computer Science, University College, Ireland.
s.foley@cs.ucc.ie

Abstract. An integrity policy defines the situations when modification of information is authorized and is enforced by the security mechanisms of the system. However, in a complex application system it is possible that an integrity policy may have been incorrectly specified and, as a result, a user may be authorized to modify information that can lead to an unexpected system compromise. In this paper we propose a scalable and quantitative technique that uses constraint solving to model and analyze the effectiveness of application system integrity policies.

1 Introduction

Conventional security models such as [3, 13, 27, 32] are operational in nature in that they define *how* to achieve integrity but do not define *what* is meant by integrity. For example, the Clark-Wilson model [13] recommends that well-formed transactions, separation of duties and auditing be used to ensure integrity. However, the model does not attempt to define whether a particular security policy configuration actually achieves integrity: evaluating a system according to the Clark-Wilson model gives a confidence to the extent that good design principles have been applied. However, when we define a complex separation of duty policy, we cannot use the model to guarantee that a user of the system cannot somehow bypass the intent of the separation via some unexpected circuitous route.

In [17, 18] it is argued that to provide such guarantees it is necessary to model the behavior of both the system (with its protection mechanisms) and the *infrastructure* in which the system operates. Infrastructure is everything that serves the system requirements: software, hardware, users, and so forth. Even if a system is functionally correct, the infrastructure is likely to fail: software fails, users are dishonest, do not follow procedures, and so forth. The system and

* Partially supported by MIUR project "Constraint Based Verification of Reactive Systems" (COVER), and by the MIUR project "Network Aware Programming: Object, Languages, Implementation" (NAPOLI).
** Support received from Science Foundation Ireland under Grant 00/PI.1/C075.

its security mechanisms must be designed to be resilient to these infrastructure failures. Only when a system is characterized in this way can it become possible to analyze whether a particular system configuration (including security policy) ensures integrity.

The approach in [17,18] provides a formal trace based semantics for integrity that requires detailed formal specifications to be provided for the system and its infrastructure. This requires considerable specification effort and the cost of such in-depth specification and subsequent analysis may be justified for small critical security mechanisms. However, we conjecture that such integrity analysis would not scale well to the configuration of a large and/or complex application system because it would be necessary to formally specify and reason about the potential behavior of *every* infrastructure component, user and so forth. Furthermore, [17,18] does not consider any approach to mechanizing the analysis and formal verification process.

In this paper we extend the work outlined in [6] that proposes the use of constraints as a more abstract and complementary approach to [17, 18]. The approach requires less semantic detail about the operation of the system and its infrastructure. Rather than attempting to model the complete behavior of the system and infrastructure (as in [17,18]), we model only those components that are considered relevant to the security policy and configuration. This is done by modeling the system and infrastructure in terms of the *constraints* that they impose over security relevant components of the system. This results in a definition of integrity consistency that can be solved as a constraint satisfaction problem [22,23].

An advantage to expressing integrity analysis as a constraint satisfaction problem is that there exists a wide body of existing research results on solving this problem for large systems of constraints in a fully mechanized manner [1, 11,15,20,25]. Constraints have been used in many practical analysis tools, such as Concurrent Engineering and Computer Aided Verification [10,12,14]. Thus, the results in this paper provide a basis for the development of practical tools for integrity analysis of complex application system security policies.

A further advantage to using a constraint based framework is that it becomes possible to carry out a quantitative analysis of integrity using *soft constraints* [4,7–9,16,19,26,29,30]. A quantitative analysis provides a fine-grained measure of how secure a system is, rather the simple coarse-grained false/true provided by the conventional 'crisp' constraints.

The paper is organized as follows. Section 2 provides an introduction to constraints and the constraint satisfaction problem. Section 3 adapts the results in [17, 18] and proposes a more abstract approach to modeling systems within a crisp constraint framework. Section 4 describes how soft constraints are used to carry out quantitative integrity analysis. A number of examples are used throughout the paper to illustrate the approach.

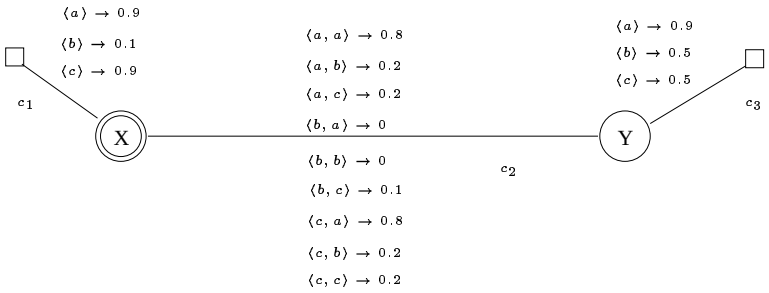

Fig. 1. A fuzzy CSP.

2 Introduction to Constraint Solving

Constraint Solving is an emerging software technology for declarative description and effective solving of large problems. The constraint programming process consists of the generation of requirements (constraints) and solution of these requirements, by specialized constraint solvers.

When the requirements of a problem are expressed as a collection of boolean predicates over variables, we obtain what is called the *crisp* (or classical) Constraint Satisfaction Problem (CSP). In this case the problem is solved by finding any assignment of the variables that satisfies all the constraints.

Sometimes, when a deeper analysis of a problem is required, *soft* constraints are used instead. Soft constraints associate a qualitative or quantitative value either to the entire constraint or to each assignment of its variables. Such values are interpreted as level of preference or importance or cost. The levels are usually ordered, reflecting the fact that some levels (constraints) are better than others. When using soft constraints it is necessary to specify, via suitable combination operators, how the level of preference of a global solution is obtained from the preferences in the constraints.

Several formalizations of the concept of *soft constraints* are currently available. In the following, we refer to the formalization based on c-semirings [4, 7–9], which can be shown to generalize and express both crisp and soft constraints [5, 8].

2.1 Semiring-based CSPs

A semiring-based constraint assigns to each instantiation of its variables an associated value from a partially ordered set. When dealing with crisp constraints, the values are the boolean *true* and *false* representing the admissible and/or non-admissible values; when dealing with soft constraints the values are interpreted as preferences.

The framework must also handle the combination of constraints. To do this one must take into account such additional values, and thus the formalism must provide suitable operations for combination ($\times$) and comparison ($+$) of tuples

of values and constraints. This is why this formalization is based on the concept of c-semiring.

Semirings. A *semiring* is a tuple $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ such that: A is a set and $\mathbf{0}, \mathbf{1} \in A$; $+$ is commutative, associative and $\mathbf{0}$ is its unit element; $\times$ is associative, distributes over $+$, $\mathbf{1}$ is its unit element and $\mathbf{0}$ is its absorbing element. A c-semiring ("c" stands for "constraint-based") is a semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ such that $+$ is idempotent with $\mathbf{1}$ as its absorbing element and $\times$ is commutative [4,8]. In the following we will always use the word semiring as standing for c-semiring.

Let us consider the relation $\leq_S$ over A such that $a \leq_S b$ iff $a + b = b$. It is possible to prove that: $\leq_S$ is a partial order; $+$ and $\times$ are monotone on $\leq_S$; $\mathbf{0}$ is its minimum and $\mathbf{1}$ its maximum, and $\langle A, \leq_S\rangle$ is a complete lattice with lowest upper bound operator $+$. Moreover, if $\times$ is idempotent, then: $+$ distributes over $\times$, and $\langle A, \leq_S\rangle$ is a complete distributive lattice with greatest lower bound operator $\times$. The $\leq_S$ relation is what we will use to compare tuples and constraints: $a \leq_S b$ it intuitively means that b is better than a.

Constraint Problems. Given a semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ and an ordered set of variables V over a finite domain D, a *constraint* is a function which, given an assignment $\eta : V \to D$ of the variables, returns a value of the semiring.

By using this notation we define $\mathcal{C} = \eta \to A$ as the set of all possible constraints that can be built starting from S, D and V.

Consider a constraint $c \in \mathcal{C}$. We define his support as $supp(c) = \{v \in V \mid \exists \eta, d_1, d_2 . c\eta[v := d_1] \neq c\eta[v := d_2]\}$, where

$$\eta[v := d]v' = \begin{cases} d & \text{if } v = v', \\ \eta v' & \text{otherwise.} \end{cases}$$

Note that $c\eta[v := d_1]$ means $c\eta'$ where η' is η modified with the association $v := d_1$ (that is the operator [ ] has precedence over application).

A *constraint satisfaction problem* is a pair $\langle C, con\rangle$ where $con \subseteq V$ and C is a set of constraints: con is the set of variables of interest for the constraint set C, which however may concern also variables not in con. Note that a classical CSP is a SCSP where the chosen c-semiring is: $S_{CSP} = \langle \{false, true\}, \vee, \wedge, false, true\rangle$.

Many other "soft" CSPs (Probabilistic, weighted, ...) can be modeled by using a suitable semiring structure ($S_{prob} = \langle [0,1], max, \times, 0, 1\rangle$, $S_{weight} = \langle \mathcal{R}, min, +, +\infty, 0\rangle$, ...).

Example 1 Figure 1 shows the graph representation of a fuzzy CSP[4]. Variables X and Y, and constraints are represented respectively by nodes and by undirected (unary for c_1 and c_3 and binary for c_2) arcs, and semiring values are written to the right of the corresponding tuples. The variables of interest (that is the set con) are represented with a double circle. Here we assume that the domain D of the variables contains only elements a, b and c.

[4] Fuzzy CSPs can be modeled in the SCSP framework by choosing the c-semiring $S_{FCSP} = \langle [0,1], max, min, 0, 1\rangle$.

If semiring values represent probability/fuzziness values then, for instance, the tuple $\langle a, c\rangle \rightarrow 0.2$ in constraint c_2 can be interpreted to mean that the probability/fuzziness of X and Y having values a and c, respectively, is 0.2. $\triangle$

Combining constraints. When there is a set of soft constraints $\mathcal{C}$, the combined weight of the constraints is computed using the operator $\otimes : \mathcal{C} \times \mathcal{C} \rightarrow \mathcal{C}$ defined as $(c_1 \otimes c_2)\eta = c_1\eta \times_S c_2\eta$.

Given a constraint $c \in \mathcal{C}$ and a variable $v \in V$, the *projection* of c over $V - \{v\}$, written $c \Downarrow_{(V-\{v\})}$ is the constraint c' s.t. $c'\eta = \sum_{d \in D} c\eta[v := d]$. Informally, projecting means eliminating some variables from the support. This is done by associating to each tuple over the remaining variables a semiring element which is the sum of the elements associated by the original constraint to all the extensions of this tuple over the eliminated variables. In short, combination is performed via the multiplicative operation of the semiring, and projection via the additive one.

Solutions. The *solution* of a SCSP $P = \langle C, con\rangle$ is the constraint $Sol(P) = (\bigotimes C) \Downarrow_{con}$. That is, we combine all constraints, and then project over the variables in *con*. In this way we get the constraint with support (not greater than) *con* which is "induced" by the entire SCSP. Note that when all the variables are of interest we do not need to perform any projection.

Solutions are constraints in themselves and can be ordered by extending the $\leq_S$ order. We say that a constraint c_1 is at least as constraining as constraint c_2 if $c_1 \sqsubseteq c_2$, where for any assignment η of variables then

$$c_1 \sqsubseteq c_2 \equiv c_1\eta \leq_S c_2\eta$$

Thus, if $c_1 \sqsubseteq c_2$ holds, then constraint c_1 may be thought of as a *refinement*, or 'suitable' (more restrictive) replacement of constraint c_2.

Example 2 Consider again the solution of the fuzzy CSP of Figure 1. It associates a semiring element to every domain value of variable X. Such an element is obtained by first combining all the constraints together and then projecting the obtained constraint over X.

For instance, for the tuple $\langle a, a\rangle$ (that is, $X = Y = a$), we have to compute the minimum between 0.9 (which is the value assigned to $X = a$ in constraint c_1), 0.8 (which is the value assigned to $\langle X = a, Y = a\rangle$ in c_2) and 0.9 (which is the value for $Y = a$ in c_3). Hence, the resulting value for this tuple is 0.8. We can do the same work for tuple $\langle a, b\rangle \rightarrow 0.2$, $\langle a, c\rangle \rightarrow 0.2$, $\langle b, a\rangle \rightarrow 0$, $\langle b, b\rangle \rightarrow 0$, $\langle b, c\rangle \rightarrow 0.1$, $\langle c, a\rangle \rightarrow 0.8$, $\langle c, b\rangle \rightarrow 0.2$ and $\langle c, c\rangle \rightarrow 0.2$. The obtained tuples are then projected over variable X, obtaining the solution $\langle a\rangle \rightarrow 0.8$, $\langle b\rangle \rightarrow 0.1$ and $\langle c\rangle \rightarrow 0.8$. $\triangle$

3 Integrity Analysis with Crisp Constraints

In [17, 18] functional requirements are expressed as properties over the possible traces of actions at the interface of a system. In this section we take a more

abstract approach by describing requirements in terms of constraints on variables that are invariant over the lifetime of the system.

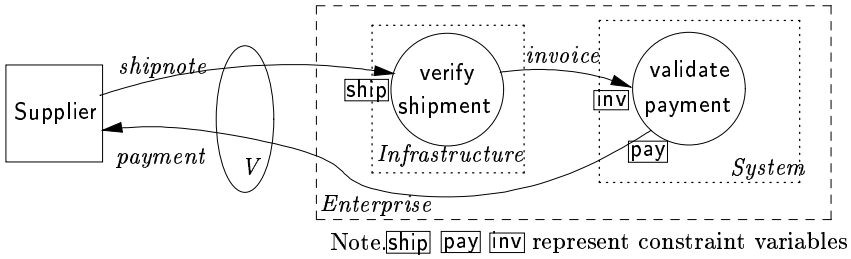

Fig. 2. A simple payment enterprise

Example 3 A simple enterprise receives shipments, and generates associated payments for a supplier. Requirements Analysis identifies the actions *shipnote* and *payment*, corresponding to the arrival of a shipment (note) and its associated payment, respectively. For the purposes of integrity, the analysis has identified a requirement that the system should not pay its supplier more than the stated value of goods shipped.

Let the constraint variables ship and pay represent the total value of goods shipped to date and the total value of payments made to date, respectively. Constraint Probity describes the requirement as an invariant over variables ship and pay.

$$\mathsf{Probity} \equiv \mathsf{pay} \leq \mathsf{ship}$$

Figure 2 outlines a possible implementation of this requirement. A clerk verifies shipment notes and enters invoice details (action *invoice*) to a computer system, which in turn, generates payment to the supplier. This implementation is described in terms of variables ship, pay and variable inv which represents the total value of invoices generated to date.

A clerk should not process more invoices than shipments and, therefore, the clerks behavior is represented by the following constraint.

$$\mathsf{Clerk} \equiv \mathsf{inv} \leq \mathsf{ship}$$

The requirement on the invoice processing application system is

$$\mathsf{Appl} \equiv \mathsf{pay} \leq \mathsf{inv}$$

and the enterprise design is specified as the constraint

$$\mathsf{Imp1} \equiv \mathsf{Appl} \otimes \mathsf{Clerk}$$

obtained by combining together Appland Clerk constraints. Intuitively, integrity is ensured in this system since Imp1 ensures the high-level requirement Probity. $\triangle$

In the above example, the supplier's interface V to the system is modeled in terms of the variables ship and pay. Constraints between these variables are used to characterize our requirements for the system. We want to ensure that the implementation upholds probity through this interface, that is,

$$\mathsf{Imp1}_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}} \sqsubseteq \mathsf{Probity}$$

We are unconcerned about the possible values of the 'internal' variable inv and thus the constraint relation $\mathsf{Imp1}_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}}$ describes the constraints in Imp1 that exist between variables ship and pay. By definition, the above equation defines that all of the possible solutions of $\mathsf{Imp1}_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}}$ are solutions of Probity, that is, for any assignment η of variables then

$$\mathsf{Imp1}_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}}\ \eta \leq_S \mathsf{Probity}\ \eta$$

Definition 1 We say that the requirement S *locally refines* requirement R through the interface described by the set of variables V iff $S_{\Downarrow V} \sqsubseteq R_{\Downarrow V}$. △

Example 4 Continuing Example 3, assume that the application system will behave reliably and uphold Appl. However, it is not reasonable to assume that the clerk will always act reliably as Clerk. In practice, the clerk could take on any behavior:

$$\overline{\mathsf{Clerk}} \equiv (\mathsf{inv} \leq \mathsf{ship} \vee \mathsf{inv} > \mathsf{ship}) = \mathit{true}$$
$$\mathsf{Imp2} \equiv \overline{\mathsf{Clerk}} \otimes \mathsf{Appl}$$

Imp2 is a more realistic representation of the actual enterprise. It more accurately reflects the reliability of its infrastructure than the previous design Imp1. However, since inv is no longer constrained it can take on any value, and therefore, pay is unconstrained and we have

$$\mathsf{Imp2}_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}} \not\sqsubseteq \mathsf{Probity}$$

that is, the implementation of the system is not sufficiently robust to be able to deal with internal failures in a safe way and uphold the original probity requirement. △

In [21], integrity is given as one attribute of *dependability*. Dependability is characterized as a *"property of a computer system such that reliance can be justifiably placed on the service it delivers"* [21]. In [17, 18] we argue that this notion of dependability may be viewed as a class of refinement whereby the nature of the reliability of the enterprise is explicitly specified.

Definition 2 (Dependability) If R gives requirements for an enterprise and S is its proposed implementation, including details about the nature of the reliability of its infrastructure, then S is as *dependably safe* as R at interface that is described by the set of variables E if and only if $S_{\Downarrow E} \sqsubseteq R_{\Downarrow E}$ △

Separation of duties [13] is a common implementation technique for achieving integrity. While fault-tolerant techniques replicate an operation, separation of duties can be thought of as a partitioning of the operation across different user domains.

Example 5 When a shipment arrives a clerk verifies the consignment at goods-inwards (entering *consign* into the system). When an invoice arrives, a different clerk enters details into the system, and if the invoice matches a consignments, a payment is generated. So long as the operations are separated then a single clerk entering a bogus consignment or invoice can be detected by the system. This is depicted in Figure 3.

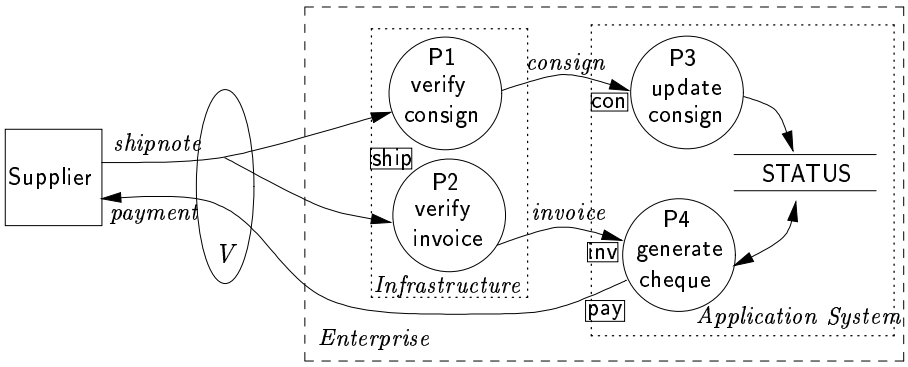

Fig. 3. Supporting separation of duties

Let variables inv and con represent the total value of invoices and consignments, respectively, received to date. Specifications Clerk1, Clerk2 and App3 define the constraints on the system variables, reflecting invariants that are expected to be upheld by the clerks and the application system.

$$\begin{aligned} \mathsf{Clerk1} &\equiv \mathsf{con} \leq \mathsf{ship} \\ \mathsf{Clerk2} &\equiv \mathsf{inv} \leq \mathsf{ship} \\ \mathsf{App3} &\equiv \mathsf{pay} \leq min(\mathsf{con}, \mathsf{inv}) \end{aligned}$$

This system is as dependably safe as Probity even when a single clerk fails, that is, we have

$$\begin{aligned} (\mathsf{Clerk1} \otimes \mathsf{Clerk2} \otimes \mathsf{App3})_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}} &\sqsubseteq \mathsf{Probity} \\ (\mathsf{Clerk2} \otimes \mathsf{App3})_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}} &\sqsubseteq \mathsf{Probity} \\ (\mathsf{Clerk1} \otimes \mathsf{App3})_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}} &\sqsubseteq \mathsf{Probity} \end{aligned}$$

Note that the absence of a constraint means that it does not restrict variables to any values. However, the system is not resilient to the failure of both clerks

nor to the failure of the application software. Removing the 'normal behavior' constraints imposed by both clerks or by the application yields the following.

$$\begin{aligned} \mathsf{App3}_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}} &\not\sqsubseteq \mathsf{Probity} \\ (\mathsf{Clerk1} \otimes \mathsf{Clerk2})_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}} &\not\sqsubseteq \mathsf{Probity} \end{aligned}$$

As currently defined, our specification favors the payment-enterprise, not the supplier: payments may be very late, or not made at all, but are never bogus. If a clerk fails then payment may not be made. In reality, the infrastructure contains many additional components; audit logs to record failures and supervisors, who make judgments and rectify these inconsistencies. △

Example 6 Another approach to dealing with unreliable systems (infrastructure) is to replicate the faulty components and make the system fault tolerant. We can make the payment enterprise fault tolerant if we replicate the clerk. We assume that every shipment is processed by $2k+1$ replicated clerks. The system votes (on the $2k+1$ invoices) to decide whether or not a consignment is valid. In this case, the abnormal behavior of the infrastructure is represented by properly constraining the behavior of at least $k+1$ clerks, and we can argue that the resulting system is as dependably safe as Probity. △

4 Quantitative Integrity Analysis

The examples in the previous section use crisp constraints to describe system requirements and implementations. When a quantitative analysis of the system is required then it is necessary to represent these properties using soft constraints.

Example 7 Consider again the Probity requirement and suppose that we aim not only to have a correct implementation, but, if possible, to have the "best" possible implementation. To do this we consider a soft constraint between variables ship and pay that assigns to the configuration $\mathsf{ship} = a$ and $\mathsf{pay} = b$ the preference level represented by the integer $a - b$[5]. If we are looking for the best implementation for the buyer, then we will try to maximize this level. In this way, different instances of the same system can be compared, and different implementations can be checked and analyzed.

Soft constraints also provide a basis for evaluating and comparing less resilient implementations that cannot uphold the intended requirement. For example, if an acceptable implementation Imp cannot be found to satisfy

$$\mathsf{Imp}_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}} \sqsubseteq \mathsf{Probity}$$

then one might be satisfied (in some sense) by selecting the best of the less resilient, but acceptable implementations. Given insufficiently resilient implementations Imp_1 and Imp_2 then their corresponding semiring levels provide a

[5] This value represent how much pay differs from ship. Our goal is to have pay=ship, but sometimes this is impossible and our goal will be to minimize the $a-b$ difference.

relative ordering that allow the selection of the 'best' of the less resilient implementations.

For example, suppose that the payment system is implemented in such a way that a payment *fully* clears the outstanding balance and that payment must be a multiple of 100. Such an implementation satisfies Probity if payments are made only when the outstanding balance is a multiple of 100. However, there a family of other implementations that round up the outstanding amount to a multiple of 100, and, while they do not uphold probity, may be acceptable in some sense. The overpayment pay-bal gives us a soft measure (defined by the semiring) of how acceptable the 'invalid' implementations might be; the 'best' implementation seeks to minimize this, that is, the solution that rounds upwards to the nearest multiple of 100. △

Probabilistic based reasoning can also be done within the soft constraints framework. For example, consider an implementation Imp3 that ensures that the number of payments is never more than 3, regardless of the number of shipments received. This is represented as:

$$\mathsf{Imp3} \equiv \mathsf{pay} \leq 3.$$

Assume that there is a constraint on variable ship that specifies the probability of the possible number of shipments made at a certain time. If the nature of the probability distribution is such that it is generally more likely that the value of ship is greater than 3, then Imp3 is a not unreasonable implementation (despite $\mathsf{Imp3}_{\Downarrow\{\mathsf{ship},\mathsf{pay}\}} \sqsubseteq \mathsf{Probity}$ not holding).

The following example illustrates how one might consider the probability of component failure within a specification.

Example 8 Consider the assignment of probability of failure to the infrastructure components of Example 5.

For the sake of efficiency, assume that in this implementation Imp4, rather than verifying consignment values (and details), Clerk1 computes batch totals [33] by simply counting the number of consignments received to date. Let the variable contot represent this value. The invoice processing application also counts the number of invoices received to date (as invtot) and compares this against contot. If there is a discrepancy then an error is flagged and it is assumed that it is investigated and repaired.

The implementation is characterized by relationships between these variables that include:

- If contot = invtot (totals match) then the probability that the invoice is accurate is $P1$, that is, the probability that inv = ship is $P1$.
- If contot ≠ invtot then the probability that the invoice is accurate is $P2$, that is, the probability that inv = ship is $P2$.
- The application program could fail with an incorrect value for pay resulting in an incorrect relationship pay = inv or pay ≠ inv. Let $P3$ represent the probability of the application failing, regardless of the relationship between pay and inv.

The application Imp4 is obtained by combining all the constraints. We will easily obtain

- If contot = invtot then the probability that payment is safe (not over payment, pay $\leq$ ship) is $P1 \times (1 - P3)$ where $P3$ is the probability of the application failing.
- If contot $\neq$ invtot then the probability that payment is safe (pay $\leq$ ship) is $P2 \times P3$. We assume that when such a discrepancy is found then the application software can repair it and, therefore, the probability that the payment is safe in this case is also $P2 \times P3$.

The constraints between pay, ship, contot and invtot are built over the probability semiring $S_{prob} = \langle [0,1], max, \times, 0, 1 \rangle$.

When using probability in the implementation, then the specification must be defined in similar way. In this case, a correct specification must ensure that the probability that pay $\leq$ ship is at least $P4$.

Thus, the system specification is

$$\mathsf{Probity}_{prob} \equiv \mathsf{Prob} : (\mathsf{pay} \times \mathsf{ship}) \to \mathbb{R}$$

$$\mathsf{Prob}(a,b) = \begin{cases} P4 & \text{if } a \leq b, \\ 1 - P4 & \text{otherwise.} \end{cases}$$

To check correctness of our implementation we have to check that

$$\mathsf{Imp4} \Downarrow_{\{\mathsf{pay},\mathsf{ship}\}} \sqsubseteq \mathsf{Probity}_{prob}$$

$\triangle$

5 Discussion and Conclusion

The contribution of this paper is a scalable and quantitative technique for analyzing the configuration of application system integrity policies.

By modeling the system and its infrastructure only in terms of abstract constraints, we argue that it becomes more realistic (than [17]) to consider modeling large complex application systems. Constraint solving can be done for large problems and, therefore, the proposed integrity analysis (as a constraint satisfaction problem) should scale up to similarly large/complex application systems configurations.

In [28] a policy analysis technique is proposed for detecting possible conflicts between separation of duty, user role assignment and role inheritance rules. This can be thought of as providing an analysis on, what is, in effect, the constraints on user role assignments. While useful, it is limited since it does not consider any further semantic information about the system and/or infrastructure. It would be interesting to apply the integrity analysis techniques proposed in this paper to extend the results in [28] for RBAC models.

Section 4 describes how soft constraints provide a basis for a quantitative analysis of integrity. Soft constraints may be used in two ways. Firstly, they can be used to provide a measure of integrity to compare the effectiveness of different system configurations. If it is not possible to develop a resilient system configuration that fully meets the set of system requirements, then one may wish to consider the best of the less resilient, but acceptable (in some sense) configurations. For example, if it is not possible to configure a system that is resilient to all internal fraud, then an acceptable alternative might be to keep the fraud within some limit. Example 7 sketches a simplistic example of this; further research is required to develop this in general.

The second application of soft constraints to the analysis of integrity is that it allows the use of quantitative information in modeling the system and infrastructure configuration. By associating probability measures with component failures, Example 8 described the validation that a system configuration/policy achieved integrity within some degree of probability. Soft constraints provide a practical framework in which such sophisticated models can be analyzed.

In [2] soft constraints are used to represent confidentiality and authentication properties of security protocols. This approach is not unlike the strategy taken in this paper. The solution of the resulting constraint system gives a measure of the confidentiality/authentication of the system. In [2] a protocol run is compared with an "ideal" run without spies, When the solutions differ an attack to the protocol is identified. The proposed integrity analysis must consider various 'spy's, each characterizing the threats that a protection mechanism must withstand.

Sections 3 and 4 use constraints to describe invariant relationships between constraint variables over the lifetime of the system. This abstract approach means that it is possible to use verification tools to describe and automatically analyze whether system configurations achieve integrity. Developing such a environment is a topic for future research.

However, we are not limited to only this style of abstract reasoning. In [17] an unwound form of local refinement is given in terms of conditions (constraints) on states and state transitions. A system that is specified in a model-oriented manner (for example, [31]) can be characterized in terms of constraints over state variables and transitions, and therefore, this unwound version of local refinement can be used to analyze integrity of more concrete specifications within a constraint framework.

The use of soft constraints permit us to perform a quantitative analysis of system integrity (see Section 4) useful to compare two system implementation. In [24] they follow a similar strategy by approximating the notion of probabilistic non-interference with a notion of ϵ-similarity. To do this they use a concurrent constraint probabilistic language. Despite the fact that the use of constraints is common between their and our approach, they rely on the notion of path (or trace) and on the probability of each path.

References

[1] G. J. Badros, A. Borning, and P. J. Stuckey. The cassowary linear arithmetic constraint solving algorithm. *ACM Transactions on Computer Human Interaction*, 8(4):276–306, dec 2001.

[2] G. Bella and S. Bistarelli. Soft Constraints for Security Protocol Analysis: Confidentiality. In *Proc. of the 3rd International Symposium on Practical Aspects of Declarative Languages (PADL'01)*, LNCS 1990, pages 108–122. Springer-Verlag, 2001.

[3] K.J. Biba. Integrity considerations for secure computer systems. Technical Report MTR-3153 Rev 1 (ESD-TR-76-372), MITRE Corp Bedford MA, 1976.

[4] S. Bistarelli. *Soft Constraint Solving and programming: a general framework.* PhD thesis, Dipartimento di Informatica, Università di Pisa, Italy, mar 2001. TD-2/01.

[5] S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie. Semiring-based CSPs and Valued CSPs: Frameworks, properties, and comparison. *CONSTRAINTS: An international journal. Kluwer*, 4(3), 1999.

[6] S. Bistarelli and S.N. Foley. Analysis of integrity policies using soft constraints. In *Proceedings of IEEE Workshop Policies for Distributed Systems and Networks*, pages 77–80, June 2003.

[7] S. Bistarelli, U. Montanari, and F. Rossi. Constraint Solving over Semirings. In *Proc. IJCAI95*, San Francisco, CA, USA, 1995. Morgan Kaufman.

[8] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Solving and Optimization. *Journal of the ACM*, 44(2):201–236, Mar 1997.

[9] S. Bistarelli, U. Montanari, and F. Rossi. Soft concurrent constraint programming. In *Proc. ESOP, April 6 - 14, 2002, Grenoble, France*, LNCS. Springer-Verlag, 2002.

[10] J.A. Bowen and D. Bahler. Constraint-based software for concurrent engineering. *IEEE Computer*, 26(1):66–68, January 1993.

[11] Kirchner C. Kirchner and M.Vittek. Designing clp using computational systems. In P. Van Hentenryck and S. Saraswat, editors, *Proceedings of Principles and Practice of Constraint Programming.* MIT Press, 1995.

[12] W. Chan, R. Anderson, P. Beame, and D. Notkin. Combining constraint solving and symbolic model checking for a class of systems with non-linear constraints. In Orna Grumberg, editor, *Computer Aided Verification, 9th International Conference, CAV'97 Proceedings*, volume 1254 of *Lecture Notes in Computer Science*, pages 316–327, Haifa, Israel, June 1997. Springer-Verlag.

[13] D. D. Clark and D. R. Wilson. A comparison of commercial and military computer security models. In *Proceedings Symposium on Security and Privacy*, pages 184–194. IEEE Computer Society Press, April 1987.

[14] G. Delzanno and T. Bultan. Constraint-based verification of client-server protocols. In *Proceedings CP2001*, volume 2239 of *Lecture Notes in Computer Science*, pages 286–?? Springer-Verlag, 2001.

[15] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The constraint logic programming language chip. In *Proceedings of FGCS*, pages 693–702, 1988.

[16] H. Fargier and J. Lang. Uncertainty in constraint satisfaction problems: a probabilistic approach. In *Proc. European Conference on Symbolic and Qualitative Approaches to Reasoning and Uncertainty (ECSQARU)*, volume 747 of *LNCS*, pages 97–104. Springer-Verlag, 1993.

[17] S.N. Foley. Evaluating system integrity. In *Proceedings of the ACM New Security Paradigms Workshop*, 1998.

[18] S.N. Foley. A non-functional approach to system integrity. IEEE Journal on Selected Areas in Commications, 2003. *forthcoming.*

[19] E.C. Freuder and R.J. Wallace. Partial constraint satisfaction. *AI Journal*, 58, 1992.

[20] T. Frühwirth. Theory and practice of constraint handling rules. *Journal of Logic Programming - Special Issue on Constraint Logic Programming*, 37(1–3):95–138, oct-dec 1998.

[21] J. Laprie. Dependability: Basic concepts and terminology.

[22] A.K. Mackworth. Constraint satisfaction. In S.C. Shapiro, editor, *Encyclopedia of AI (second edition)*, pages 285–293. John Wiley & Sons, 1992.

[23] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Science*, 7:95–132, 1974. Also Technical Report, Carnegie Mellon University, 1971.

[24] A. Di Pierro, C. Hankin, and H. Wiklicky. On approximate non-interference. In *editor, Proceedings of WITS'02 – Workshop on Issues in the Theory of Security.* IFIP WG1.7, 2002.

[25] J.F. Puget. A c++ implementation of clp. In *Proceedings of the 2nd Singapore International Conference on Intelligent Systems*, 1994.

[26] Zs. Ruttkay. Fuzzy constraint satisfaction. In *Proc. 3rd IEEE International Conference on Fuzzy Systems*, pages 1263–1268, 1994.

[27] R Sandhu et al. Role based access control models. *IEEE Computer*, 29(2), 1996.

[28] A. Schaad and D. Moffett. The incorportation of control principles into access control policies. In *Workshop on Policies for Distributed Systems and Networks*, Bristol, UK, 2001.

[29] T. Schiex. Possibilistic constraint satisfaction problems, or "how to handle soft constraints?". In *Proc. 8th Conf. of Uncertainty in AI*, pages 269–275, 1992.

[30] T. Schiex, H. Fargier, and G. Verfaille. Valued Constraint Satisfaction Problems: Hard and Easy Problems. In *Proc. IJCAI95*, pages 631–637, San Francisco, CA, USA, 1995. Morgan Kaufmann.

[31] J. M. Spivey. *The Z Notation: A Reference Manual.* Series in Computer Science. Prentice Hall International, second edition, 1992.

[32] U. S. Department of Defense. Integrity-oriented control objectives: Proposed revisions to the trusted computer system evaluation criteria (TCSEC). Technical Report DOD 5200.28-STD, U. S. National Computer Security Center, October 1991.

[33] United States General Accounting Office, Accounting and Information Management Division. *Financial Audit Manual*, December 1996. GAO/AFMD-12.19.5A.