Dealing with Security Requirements During the Development of Information Systems

Lawrence Chung

Department of Computer Science, University of Toronto,
Toronto, Ontario, Canada M5S 1A4

Abstract. A growing concern for information systems (*ISs*) is their quality, such as security, accuracy, user-friendliness and performance. Although the quality of an IS is determined largely by the development process, relatively little attention has been paid to the methodology for achieving high quality. A recent proposal [32] takes a process-oriented approach to representing non-functional, or quality, requirements (*NFRs*) as potentially conflicting or harmonious goals and using them during the development of software systems. By treating security requirements as a class of NFRs, this paper applies this process-oriented approach to designing secure ISs. This involves identification and representation of various types of security requirements (as goals), generic design knowledge and goal interactions. This treatment allows reusing generic design knowledge, detecting goal interactions, capturing and reasoning about design rationale, and assessing the degree of goal achievement. Security requirements serve as a class of criteria for selecting among design decisions, and justify the overall design. This paper also describes a prototype design tool, and illustrates it using a credit card system example.

1 Introduction

Information systems (ISs), such as credit card, health insurance and criminal record systems, are becoming more and more central in our everyday life. A growing concern for such systems is their *quality*, such as security, accuracy, user-friendliness and performance. The crucial role that non-functional or quality requirements play in requirements engineering has been recognized, for example, by the multi-national NATURE project [24].

One important concern is *information security* [37], the protection of information as an asset of an enterprise, just like money or other forms of property. Leakage, damage or loss of information, which is resident either in computer memory or in a medium such as paper, microfilm or communication line, could lead to violations of privacy, financial loss or even loss of human life. For instance, revealing credit ratings, medical history or criminal records could have serious consequences for individuals, while destruction of computer equipment or networks could jeopardize the operation of an enterprise. Examples of information security breaches have been reported widely in the literature [1, 34, 39, 37].

Problem. Although the quality of an IS is determined largely by the development process, relatively little attention has been paid to the methodology for achieving high quality. In the past, developers relied mostly on their own intuition, without much systematic help or guidance. A rational approach to designing a secure IS is to consider design alternatives and their selection criteria, and then select one that best fits the needs of the intended application domain, in the spirit of rational design [42]. One way to take this approach is to provide a systematic methodology and construct a *design environment* that offers tools and automated assistance to the designer. To make the design process less *ad hoc* and more formal, a systematic methodology needs to address the following issues:

1. *How can the wide variety of well-known and lesser-known security techniques be made available to the designer through systematic search?* A systematic methodology facilitates the capture and reuse of design knowledge. Firstly, design knowledge can guide disambiguating and choosing appropriate notions of security from a rich, diversified set of security notions. For instance, the designer can focus on the confidentiality aspects of run-time operations, instead of addressing broader issues, such as availability and recovery. An abstract security requirement can also be gradually refined into one or more concrete ones. For instance, security concerns can be expressed along a specialization hierarchy [16]. More concretely, design knowledge can guide the selection of specific security techniques and functions such as authentication or access authorization (e.g., [45, 20, 21, 7]). A security function can be carried out by many mechanisms and their variations, such as a password authentication mechanism, using personal knowledge, biometrics for fingerprint-verification or voice-recognition, which test personal characteristics. Control or deterrent measures [38] are also available, such as alarms, encryption and physical-access-locks.
2. *How can interactions among potentially conflicting or synergistic requirements be managed systematically?* Deploying a technique for safeguarding information security could have negative consequences for other NFRs. While biometric authentication, for instance, may improve the security of a system, it may conflict with the requirements of cost and user-friendliness. Auditing could degrade system performance. Limiting access time may contravene availability, timeliness and user-friendliness.
3. *How can the nature of the relationships between design decisions be represented? How can the effect of each design decision be systematically evaluated?* Applying a technique affects other previous applications, with varying degrees of impact. For instance, one technique may be more valuable than another in a particular domain. One technique, which is good in one domain, may not be as good in another, due to different types of risks involved.
4. *How can security requirements be systematically integrated into the design, together with other types of NFRs?* Without a systematic methodology, security requirements are often retrofitted late in the design process (e.g., how to securely update an account), or pursued in parallel but separately from functional design (e.g., how to update an account). These practices tend

to result in systems which cannot be accredited, are more costly and less trustworthy [4].

5. *What drives design actions? What representational structures are appropriate for systematically recording the results of such actions?* Designing a large IS demands taking numerous actions. Such actions range from searching for design knowledge, selecting an appropriate technique, to adding and revising design decisions. Design steps, alternatives (both selected and discarded ones) and decisions need to be cohesively captured.

Solution. A recent proposal [32] takes a *process-oriented approach* to representing non-functional, or quality, requirements (*NFRs*) as potentially conflicting or harmonious *goals* and using them during the development of software systems. By treating security requirements as a class of NFRs, this paper applies this process-oriented approach to designing secure ISs. This involves identification and representation of various types of security requirements as *goals*, design knowledge as generic *methods*, and goal interactions as generic *correlation rules.* This treatment provides the following five benefits, thereby dealing with the corresponding five issues above. 1) Methods allow capturing and reusing design knowledge, including design rationale for making trade-offs explicit. 2) Correlation rules represent synergistic or antagonistic interactions between goals and are used in detecting actual interactions. 3) *Link types* express the degree of interaction among goals and goal relationships, and a *labelling procedure* assesses the degree of goal achievement. 4) Security requirements serve as a class of criteria for selecting among design decisions, and justify the overall design, thereby becoming an integral part of the design process. 5) Driving the design process, security goals are recorded in a structured manner into a *goal graph structure*, in the spirit of AND/OR trees [35]. Since design decisions usually contribute either positively or negatively and only partially towards a particular goal, security goals, along with other types of NFR goals, can rarely be said to be "accomplished" or "satisfied" in a clearcut sense. Accordingly, the term goal *satisficing* [42][1] is intended to suggest that generated software is expected to satisfy NFRs within acceptable limits, rather than absolutely. This notion captures, for instance, the point that the risk of security breaches can only be limited in magnitude, reduced in likelihood and made detectable, but *not* removed [30]. Security *assurance* [25] is a familiar term, which is related to satisfying security requirements within acceptable limits, rather than absolutely.

Related Work. The process-oriented approach of [32] adapts work on decision support systems, e.g., [26, 18], specializing them to the context of development of software systems with NFRs. Such work on decision support systems, in turn, extends an earlier model for representing design rationale [41] by making explicit the goals presupposed by arguments for (or against) design decisions. Compared to these preceding works, the process-oriented approach of [32] offers finer-grained link types with a formal semantics, and a built-in, albeit semi-automatic,

[1] Simon actually uses the term to refer to decision methods that look for satisfactory solutions rather than optimal ones.

labelling procedure. This approach also offers three types of goals, which facilitate the use of NFRs as criteria for selecting among design alternatives, and three types of generic methods, which allow separation of concerns in capturing and reusing different types of design knowledge. In particular, methods, correlation rules and the labelling procedure make it possible to automate parts of the design process. This process-oriented approach was also influenced by the DAIDA environment for IS development [22]. This process-oriented approach is similar in spirit to the goal-oriented approach of [14]. Both of these process- and goal-oriented approaches build on long-term experience in developing respectively information systems and general software systems. The goal-oriented approach of [14] puts relatively more emphasis on requirements acquisition than the process-oriented approach of [32] which specializes in NFRs and uses the notion of "satisficing" instead of the traditional notion of "satisfying". Additionally, the process-oriented approach of [32] offers an argumentative style of reasoning and several link types, as well as a labelling procedure.

Treating security requirements as a class of NFRs and using the process-oriented approach of [32] can be seen as complementary to a product evaluation approach to security. In a product evaluation approach, evaluation criteria serve as benchmarks for selecting a level of security and meeting it for the target system. For instance, the "Trusted Computer Systems Evaluation Criteria" (also known as *TCSEC* or the "Orange Book") [45] lays out criteria to categorize a system into one of eight hierarchical classes of enhanced protection. The "IT Security Criteria: Criteria for the Evaluation of Trustworthiness of Information Technology Systems" (or the "Green Book") [20] describes eight security functions, addressing a wide variety of commercial and governmental security concerns, in a less formal and less rigidly prescriptive manner. A more recent document, the "Harmonized Criteria" [21], emphasizes the assurance aspect of security and methodological aspect for evaluating security functionality. For a further comparison, see [25]. Due to their generality, these product evaluation criteria are *not* intended to prescribe or proscribe a specific development methodology for semi-formally managing or guiding the complex development process.

While not specific to ISs, there is a body of work which adapts existing software life cycle models to the development and evolution of secure systems. For instance, security activities can be integrated into the system life cycle, and reviewed and audited in the software quality assurance process [46]. Additionally, guidelines can be offered for dealing with certain security concerns for battle management systems, such as visibility and configuration control [12], in accordance with military standards for the software development process [13]. Similarly, the spiral model [5] can be adapted to meet military standards, by providing mappings for phases and points of iterations [4]. The spiral model can also be specialized to address both trust and performance, in the context of Ada development [27]. For a decision support system, which addresses concerns for cost-effective systems, a statistical approach can be taken to evaluate and choose the most preferred set of security control activities [6]. These works suggest the need for a comprehensive semi-formal development methodology, such as that offered by the process-oriented approach of [32].

Running Example. As the running example, this paper uses a bank's credit card system which offers consumer payment services with credit cards, by maintaining information about cardholder accounts and merchant accounts. To maintain an edge in this highly competitive market, there is a strong commitment by the credit card vendors for continued introduction of new technologies to improve the security of their services [48]. As shown in the statistics [8], the number of transactions, customers and fraud cases is ever-increasing. For instance, the number of sales slips processed by Canadian banks during 1991 was over 500 million, the number of cards fraudulently used was slightly over 50 000, and the amount of fraudulent accounts written off was slightly below $50 million.

Section 2 presents a goal graph structure, and illustrates the features of the process-oriented approach of [32] in terms of security requirements. Sections 3, 4 and 5 respectively present security-specific *goals*, *methods* and *correlation rules.* A prototype design tool is then described and illustrated. The final section summarizes the contributions of this paper and presents directions for further research.

2 Goal Graph Structure

In a goal graph structure, nodes denote goals, and links denote relationships between goals. From an initial set of security and other types of NFR goals, new goals are generated, via methods, along with new links. New links between existing nodes can also be generated, via correlation rules. Links are differentiated into link types according to the nature of the relationships between goals. A node is assigned a label indicating the degree to which its associated goal is satisficed.

Figure 1 shows a goal graph structure that a designer might construct in attempting to satisfice the security goal "all accounts should be secure", represented as *Sec*[`Account`][2]. The designer uses the method, *Subsort3*, to decompose the goal into three other goals for integrity, confidentiality and availability: *Integrity*[`Account`], *Conf*[`Account`] and *Avail*[`Account`]. The designer successively generates more specific goals, such as *IntConf*[`Account`][3], *Authentication* [`Account`] and *Biometric*[`Account`]. Each more specific goal (the *offspring*) is generated with the aim of satisficing the *parent* goal. A correlation rule comes into play when an offspring has an impact on some goals other than its parent. The generation of *Authentication*[`Account`] triggers a correlation rule, which generates a link between this goal and *Acc*[`Account`], because *Authentication*[`Account`] has an impact on *Acc*[`Account`], as well as on *IntConf*[`Account`]. When a goal contributes positively to another goal, we designate it as a *sub* link[4] (e.g., between

[2] `Account` evaluates to the set of all instances associated with the account information class.

[3] This refers to confidentiality of account information resident internally in the system.

[4] Not all link types are shown in figures.

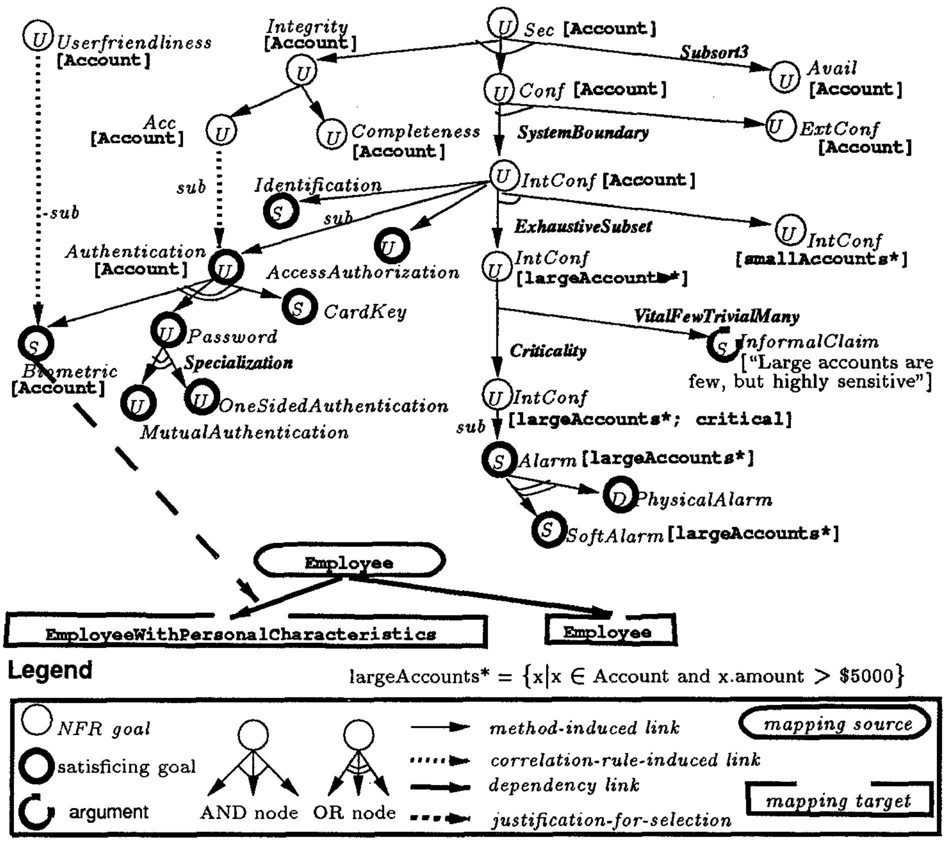


Fig. 1. Goal graph structure for secure account.

Authentication [Account] and *Acc*[Account]); when the contribution is negative, we designate it ***-sub***, as between ***Biometric***[**Account**] and ***Userfriendliness***[**Account**]. Throughout the goal graph expansion process, the effect of each design decision is propagated from offspring to parents via a labelling procedure. The result of such propagation is annotated via labels (enclosed within nodes), such as *U* (*undetermined*), *S* (*satisficed*) and *D* (*denied*).

A goal graph structure such as Figure 1 cohesively captures design steps, alternatives and decisions with respect to security requirements, as well as other classes of NFRs. Both selected and discarded alternatives serve as part of the design history, along with the way conflicts are resolved (e.g., the usage of the criticalities of goals)

Goals, methods, correlation rules, link types and labels are described in more detail in the following sections.

3 Goals

In Figure 1, there are three mutually exclusive classes of goals: ***non-functional requirements goals (NFR goals), satisficing goals*** and ***argumentation goals.*** Each goal has an associated *sort* and *parameters* whose nature depend on the goal sort. For instance, *Sec*[`Account`] is a security goal, the primary focus of this section, where *Sec* is the sort and `Account` is the parameter.

Security Goals. Information security means protecting information. When it comes to the issue of the scope of protection, a multitude of definitions is encountered. 1.) There are different emphases in definitions of security. (1.a.) *Confidentiality,* guarding against unauthorized disclosure, is the primary emphasis in evaluation criteria [45, 20, 21]. (1.b.) In some commercial applications, the focus is on *integrity* [11], guarding against unauthorized update or tampering. (1.c.) *Availability* or *assured service,* against interruption of service; this, along with confidentiality and integrity, are general concerns in evaluation criteria. (1.d.) A broader definition of security encompasses *availability* as well as *authenticity,* genuineness or faithfulness of true representation (comparable to "external consistency" in [21]), *integrity,* wholeness or completeness, *utility,* fitting the use (e.g., money amount in dollar not in yen), and *confidentiality* or secrecy [38]. 2.) The primary scope of protection could be confined to information resident *internally* in the computer, *externally,* or both. 3.) The scope can be *operational,* concerning run-time operation, or *developmental,* concerning the development stage [2].

Without guidance, however, these diversified notions related to security could be a source of confusion. For instance, focusing on the confidentiality aspect may not be sufficient to meet the accuracy needs of a specific application domain. Authorized access, meeting access-rule security requirements, does *not* necessarily preserve accuracy, due to either unintentional mistakes or intentional fraud. In order to serve as a rich set of alternatives to choose from as well as check-points to guard against omitting any important security concerns, such diversified notions are captured in *sorts* of security goals and organized along a partially ordered hierarchy. Figure 2 illustrates such a hierarchy. There are many specializations, or sub-sorts, of sort *Security. OperationalSecurity* is one sub-sort, which refers to information security during system operation, while *InternalConfidentiality* refers to the confidentiality of information items residing in the system. Note that there could also be crossovers among the sorts. For instance, confidentiality could be linked to integrity.

In addition to information items, such as `Account`, security goals can have other parameters, such as the authorizer, access condition, delegation function ([49, 19]), and task [44].

In the example used in this paper, the process-oriented approach of [32] is applied mainly to operational security. As security goals have information items as the parameter, satisficing such goals is understood in terms of enhancing the degree of confidence in the security of information items.

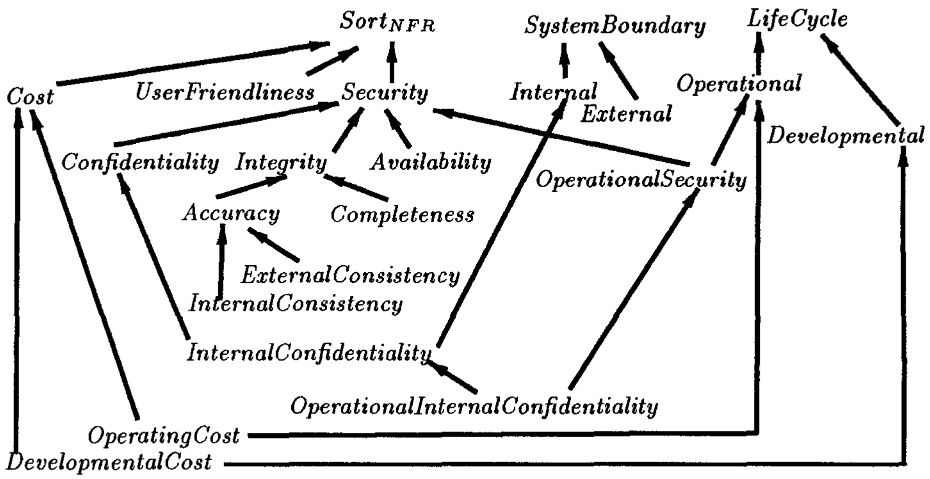


Fig. 2. A taxonomy of *sorts* for NFR goals.

Satisficing Goals & Argumentation Goals. Satisficing goals, such as *Biometric* [`Account`], represent design decisions adopted to satisfice a security goal, such as *IntConf*[`Account`]. *Dependency links* [10] represent design decisions and relate design objects (e.g., `EmployeeWithPersonalCharacteristics`, a data class) to their requirements counterparts (e.g., `Employee`, a class of employee information). Argumentation goals represent formally or informally stated evidence or counter-evidence for other goals or links. *InformalClaim* ["Large accounts are few, but highly sensitive"] is an informally-stated argument which might support the treatment of protecting large accounts as a critical confidentiality goal. Through satisficing goals, security requirements serve as criteria for selecting among design decisions, and justify the overall design.

4 Methods

Goals may be refined by the designer, who is then responsible for satisficing not only the goal's offspring but also the parent-offspring relationship represented as a link. Methods represent generic design knowledge, which can be instantiated to refine a goal or a link into one or more offspring. There are three types of refinements, corresponding to the three types of goals mentioned in the previous section.

4.1 Decomposition Methods

Reflecting the traditional wisdom of structural "divide and conquer," these methods facilitate linking security concerns with NFRs of different sorts. They also facilitate separating more sensitive information from less sensitive, thereby avoiding a single but costly strategy that *uniformly* protects *all* types of information.

They selectively add features on top of a particular type of operating system which might be adopted to meet a target class of evaluation. Examples include mandatory security (the system is responsible for enforcing access control based on the assignment of labels to information and clearance levels to users) and discretionary security (the user/owner is responsible for access control). Methods also alleviate the extreme difficulty with assuring that the target system meets the given security requirement[5].

Decomposition methods include *sort*, *parameter* and *criticality* decomposition methods. The *subSort* method is a sort decomposition method which makes a goal reduction on its sort by introducing its sub-sorts. Specializations of this method include establishing security of information items for each of the component sub-sorts of the various definitions of security described in Section 3, e.g., introducing three sub-sorts *confidentiality*, *integrity* and *availability*:

$$i/\texttt{InformationItem}: \; Sec[i] \xrightarrow{AND} \{Conf[i], Integrity[i], Avail[i]\}.$$

Instantiating this method, *sec*[Account] can be decomposed into *Conf*[Account], *Integrity*[Account] and *Avail*[Account]. The *systemBoundary* method decomposes confidentiality into *InternalConfidentiality* and *ExternalConfidentiality*, which are respectively abbreviated as *IntConf* and *ExtConf*. By the *categorization* method, a given goal is categorized into a security class (e.g., mandatory or discretionary) to meet a target level of security.

A parameter decomposition method is the *subset* method, which decomposes a goal on a set of information items into goals for each subset of information items. For instance, this method can be used to refine *IntConf*[Account] into two goals: one for those accounts whose amounts exceed $5000 (*large accounts* hereafter, which may be represented as {x | x ∈ Account and x.amount > $5000} and abbreviated as largeAccounts*) and the other for those that do not:

$$IntConf[\texttt{Account}] \xrightarrow{AND} \{IntConf[\texttt{largeAccounts}^*], IntConf[\texttt{smallAccounts}^*]\}.$$

In fact, the above is the application of the *exhaustiveSubset* method, a specialization of the *subset* method. With an *AND* link, the parent can be satisficed, when all offspring are. The subsets are exhaustive, since the union of their extensions covers all extensions of Account class. Otherwise, the *subset* method will be specialized into *properSubset* method with a *sub* link, which indicates a partial contribution of the offspring to the satisficing of the parent. Then, the *subset* method itself would have the *+und* link, indicative of inconclusive positive evidence, as the method can be specialized into two different types of methods.

There are other examples of such methods. The *subclass* method is an adaptation of the inheritance policy in [16]. This method reduces a goal on a set of information items into goals for each subclass of information items, with a *+und* link. Depending on the coverage by the subclasses, this method is specialized

[5] Di Vito et al. [15] note (p. 307) "If over 380 000 lines of text were printed at 50 lines per page, we would have over 7600 pages of proof documentation", and advocate the need for decomposition: "It is essential that the proof effort be decomposed and modularized to avoid confronting too many details at once."

into *exhaustiveSubclass* (with an *AND* link) and *inexhaustiveSubclass* (with a *sub* link). By the *classAttribute* method, to establish the security of a class, establish the security of its inherited and specialized attributes. This is adaptation of the second policy in [16]. The *individualAttribute* method decomposes a goal on the attributes of a class into goals for each attribute of the class. This method is an adaptation of ***vertical decomposition*** in [43]. The ***individualAttributeSecurityLevel*** method decomposes a goal on the attribute of a class into goals for each security level associated with the attribute, adapted from ***horizontal decomposition*** in [43].

A *criticality* method induces changes in goal criticality. Suppose that a high-level of confidentiality is demanded for large accounts. This may be treated as a critical confidentiality goal which can be induced by the *criticality* method:

$$Conf[\texttt{largeAccounts*}] \xrightarrow{eql} Conf[\texttt{largeAccounts*}; \texttt{critical}]$$

This method facilitates achieving a sensible ordering of degrees of criticality in the design structure, as advocated in [33]. The *eql* link conveys the meaning that the satisficing (or denial) of offspring is equivalent to that of the parent.

4.2 Satisficing Methods

A satisficing method commits the design to a particular way of satisficing a NFR goal. For instance, satisficing the internal confidentiality goal for accounts may demand, among other things, authentication — verifying the identity of the user to ensure that the user is in fact whom she claims to be. Such a demand can be met by instantiating the ***authentication*** method, where **Account** would be supplied as the parameter:

$$i/\texttt{InformationItem}: \; IntConf[\texttt{i}] \xrightarrow{sub} Authentication[\texttt{i}].$$

The resulting satisficing goal ***Authentication*** [**Account**], in turn, might be satisficed in terms of other satisficing goals. For instance, according to specializations of the method, the goal may be satisficed, via the *mutualMultiLayerPassword* method, which requires multiple passwords (like multiple keys for bank safes, where each key clears one lock), and both the user and the system to go through a test procedure and mutually ensure the identity of each other [40].

Authentication not only has many variations but is only one of many available satisficing methods that can be found in the literature and have been used in practice to enforce various types of security policies. Figure 3 depicts a hierarchy of security goal satisficing methods.[6] The *alarm* method prevents potentially malicious access to certain vital information, by notifying authorities of such accesses. This method may be specialized into *physicalAlarm*, notification with alarming device, and *softAlarm*, on-line notification of authorities by the system.

[6] In the earlier goal graph structure of Figure 1, a method name starts with a lower-case character, while the name of a satisficing goal starts with an upper-case character.

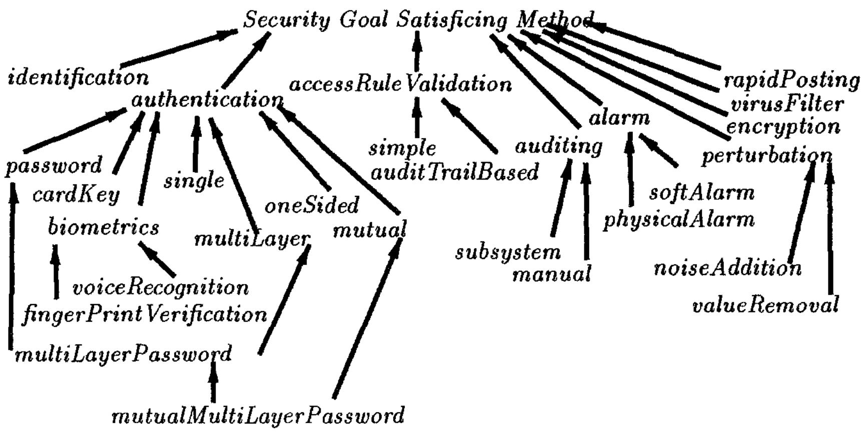


Fig. 3. A taxonomy of security goal satisficing methods.

The *perturbation* method protects a statistical database against inference by perturbation of its data [28]. Specializations of this method include *noiseAddition*, perturbing data by adding random noise, and *valueRemoval*, perturbing data by removing extreme values (e.g., for census databases).

Satisficing methods and their specializations help capture, organize and reuse knowledge about alternatives, which may be mutually antagonistic, complementary or synergistic. For instance, in Figure 1, *Alarm* [largeAccounts*] has an an *OR* link. Then, the goal can be satisficed by either *SoftAlarm*[largeAccounts*] or *PhysicalAlarm*[largeAccounts*].

4.3 Argumentation Methods

An argumentation method is used to provide either formal and informal arguments in supporting or denying a goal or a link. For instance, treating certain information as highly sensitive may be justified by the use of *vitalFewTrivialMany* [29] (the so-called *Pareto* principle) argumentation method (We borrow notation from Telos [31] to denote the link between *IntConf*[largeAccounts*] and *IntConf* [largeAccounts* ; critical] as *IntConf*[largeAccounts*]!offspring):

$$IntConf[\texttt{largeAccounts}^*]!\texttt{offspring} \xrightarrow{sup} InformalClaim[\text{“Large accounts are few, but highly sensitive”}]$$

The *sup* link indicates that the satisficing of the offspring is a sufficient evidence for the satisficing of the parent.

There are other argumentation methods. The *preferentialSelection* method selects a method among alternatives according to their relative preference. By *synergisticSelection*, a method is selected if two or more goals or links can collectively justify its application, although each alone may be incapable of justifying the application.

5 Correlation Rules

NFRs may be contradictory or harmonious. Generic interactions among NFRs can be captured through correlation rules, and used to discover actual instances. For instance, the *Biometric* authentication, which is a satisficing goal for confidentiality, could be irritating to users, hurting userfriendliness. This can be expressed by a correlation rule:

$$Userfriendliness[\mathtt{i}] \ \wedge \ Biometric[\mathtt{i}'] \ \wedge \ isA[\mathtt{i},\mathtt{i}'] \longrightarrow \\ -sub(Userfriendliness[\mathtt{i}], Biometric[\mathtt{i}'])$$

where $isA[\mathtt{P}_\mathtt{i},\mathtt{P}_\mathtt{j}]$ holds if the parameter of a userfriendliness goal is a subclass of that of a biometric goal. By specifying the parameter relationships as desired, detection of implicit relationships among different goals can be controlled. The above correlation rule can be used to infer a *-sub* link between the goals *Userfriendliness*[Account] and *Biometric*[Account].

In contrast, an authentication for a confidentiality goal is also good for accuracy goals, since a malicious user, in the absence of authentication, can penetrate the system and falsify information. The following correlation rule can be used to infer a *sub* link between the goals *Acc*[Account] and *Authentication*[Account]:

$$Acc[\mathtt{i}] \ \wedge \ Authentication[\mathtt{i}'] \ \wedge \ isA[\mathtt{i},\mathtt{i}'] \longrightarrow sub(Acc[\mathtt{i}], Authentication[\mathtt{i}']).$$

Link types, which have been introduced in previous sections, are detailed in [32]. The effect of each design decision is propagated from offspring to parents, in accordance with rules dependent on both link and label types. There are label types which indicate whether a goal has been satisficed (S in figures) or denied (D). There is also a special value for initial/inconclusive situations (U) and one for conflicts (C). Details of the labelling procedure are described in [32].

6 Prototype Design Tool

6.1 Implementation

In order to debug and refine this work on dealing with security requirements, a prototype development tool[7] for the process-oriented approach of [32] has been extended. The extension has resulted in representing taxonomies of security sorts and methods, as well as specifying correlation rules. The system is intended to assist the designer by displaying applicable methods and instantiating the selected one, detecting conflict and harmony, evaluating the status of goal satisficing and maintaining goal graphs.

The prototype is based on ConceptBase [23], which provides a knowledge base management system and an implementation of Telos [31], a knowledge representation language for ISs and a descendant of RML [17]. While Telos is

[7] Portions of the system implementation are joint work with Brian Nixon, who deals with performance requirements [36].

used for representing most of the five components, Prolog [3] is used to control and manipulate queries, correlation rules and the labelling procedure. A window-based graphical interface is used to view the goal graph expansion process, and to interactively browse, select and apply methods. A textual editor is used to enter arguments as design rationale, query and modify labels of goals and links.

6.2 Illustration

Continuing the earlier example of developing a secure credit card system in Figure 1, the designer recursively applies a variety of methods from method taxonomies. The informal concerns for security are often *ambiguous*, inviting many possible interpretations. Unlike the descriptions of goals in Section 3, a NFR or a satisficing goal is often expressed only in terms of its sort, which is not clearly explained. Thus, after refining a cost goal into several offspring, such as *EquipCost*[`System`], the designer uses several sort decomposition methods to select a particular definition and reveal relationships among different types of NFR goals, as well as between NFR goals and satisficing ones. Recognizing the importance of accuracy and external confidentiality aspects of security, the designer decomposes a security goal, *Sec*[`Account`], on its sort into *Int*[`Account`], *Conf*[`Account`] and *Avail*[`Account`] (See Figure 4). The integrity goal is further decomposed into *Acc*[`Account`] and *Comp*[`Account`]. Using successively some parameter decomposition methods, such as *ExhaustiveSubclass*, the designer has generated *Acc*[`Transaction`] from *Acc*[`Account`]. After decomposing *Acc*[`Transaction`] into *TimelyAcc*[`Transaction`], *PropertyAcc*[`Transaction`] and *ValueAcc*[`Transaction`], the designer treats *TimelyAcc* [`Transaction`] as a critical goal, and, using the *vitalFewTrivialMany* argumentation method, supports the treatment by informally arguing that market surveys show that timely accuracy is of strategic importance.

In order to help the designer to see the trade-offs in applying a satisficing method, the system offers the correlation rule table (See Figure 5), which is a synopsis of the correlation rules that were described earlier in Section 5. Entries of the form <*condition*, *linkType*> mean "if the *condition* holds, then the relationship between the NFR and satisficing goals is given by the *linkType*." The table is used for browsing and selecting a satisficing method. For *TimelyAcc*[`Transaction;critical`], the designer successively applies the ***RapidPosting***, ***ReduceTransmissionTime*** and ***InstallInputDevice*** satisficing method. However, in using the last method, there is a potential for increased cost. At this point, the system uses correlation rules and detects (via *upward detection*) a potential conflict between *InstallInputDevice*[`Transaction`] and *EquipmentCost* [`Transmission`], and proposes a new *-sub* link. The designer can take a look at the implicit link and either move on, if it is intended, or make a revision, if not.

A security breach could take place either internally, by staff accessing the system, or externally, in terms of forgery of vouchers, remittance requests, etc. Instantiating the *SystemBoundary* method, the designer decomposes *Conf*[`Account`] into *ExtConf*[`Account`] and *IntConf*[`Account`]. Instantiating successively some parameter decomposition methods, the designer generates *ExtConf*[`Transaction`]

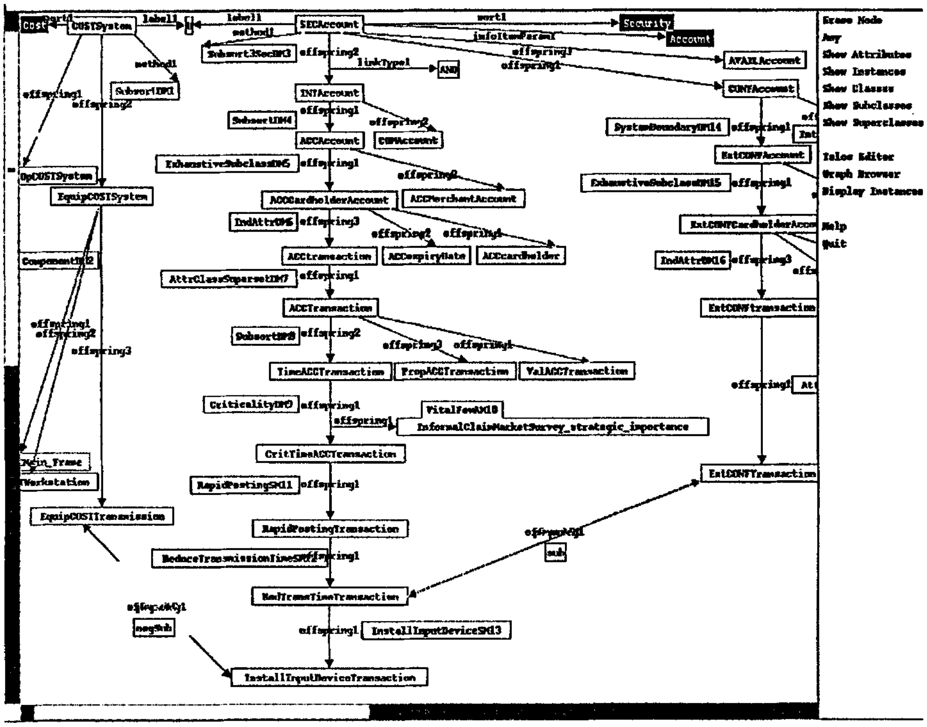
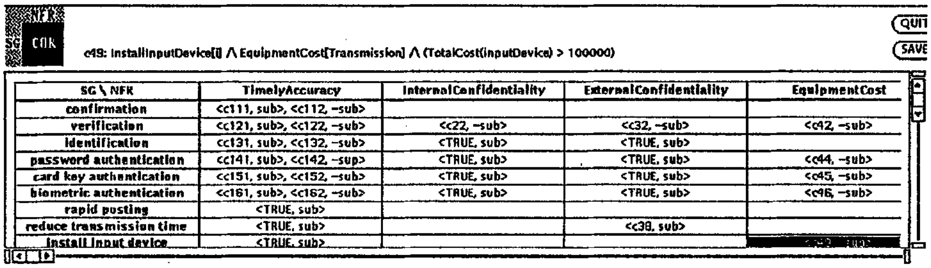


Fig. 4. Goal graph structure for a credit card system example.



c49: InstallInputDevice[I] ∧ EquipmentCost[Transmission] ∧ (TotalCost(inputDevice) > 100000)

SG \ NFR	TimelyAccuracy	InternalConfidentiality	ExternalConfidentiality	EquipmentCost
confirmation	<c111, sub>, <c112, –sub>			
verification	<c121, sub>, <c122, –sub>	<c22, –sub>	<c32, –sub>	<c42, –sub>
identification	<c131, sub>, <c132, –sub>	<TRUE, sub>	<TRUE, sub>	
password authentication	<c141, sub>, <c142, –sup>	<TRUE, sub>	<TRUE, sub>	<c44, –sub>
card key authentication	<c151, sub>, <c152, –sub>	<TRUE, sub>	<TRUE, sub>	<c45, –sub>
biometric authentication	<c161, sub>, <c162, –sub>	<TRUE, sub>	<TRUE, sub>	<c46, –sub>
rapid posting	<TRUE, sub>			
reduce transmission time	<TRUE, sub>		<c38, sub>	
install input device	<TRUE, sub>			<c49, sub>

Fig. 5. Correlation Rule Table.

from *ExtConf*[Account]. Reducing the transmission time from transaction to its posting has other benefits such as improving security. Accordingly, the system now detects (via *downward-detection*) a potential harmony between *ExtConf* [Transaction] and *ReduceTransmissionTime*[Transaction]. To avoid any undesirable consequences, that might have been overlooked, the designer examines the correlation condition in the correlation table, and discovers that, for the harmony to be effective, the phone line from the input device to the system should be safeguarded against wire-tapping. This helps the designer avoid *omissions* of certain important concerns, as above.

The system, as seen above, has detected both synergistic and antagonistic interactions between goals. When a conflict is detected, the designer may need to make a trade-off, and use an argumentation method in, say, outweighing the synergistic benefits over the antagonistic penalties. The labelling procedure has assigned labels to all goals (one is shown for goals at the top in the figures). One result of this design process is that input devices can be installed at the transaction point to improve security of cardholder accounts.

7 Conclusions

The main contribution of this paper is the application of a process-oriented approach [32] to represent and use security requirements as a class of non-functional requirements during information system design. This paper has shown the need and a methodology for capturing various types of available design knowledge specific to dealing with a class of security goals, such as knowledge for goal disambiguation, criticality, enforcement and interaction, and for capturing various design rationale to make explicit trade-offs.

Through implementation of a prototype design tool, and experimentation with a credit card system example, this paper has also demonstrated how parts of the design process can be automated, with several types of functionality: displaying method hierarchies and then instantiating the method selected; displaying correlation rule tables and then using correlation rules to detect potentially conflicting or harmonious goal interactions, and warning the designer (through correlation conditions) to prevent certain actions that might jeopardize the satisficing of certain NFRs; evaluating the effects of various design decisions, using the labelling procedure; and maintaining goal graph structures.

A long term research project would be to establish a theoretical foundation for representing and reasoning with security requirements. For such a foundation, a semantics and its proof theory would be needed, along with efficient algorithms for special classes of inferences. Also a scheme is needed to marry quantitative and qualitative representations and their reasoning. Such a scheme would allow the consideration of the marginal utility of an additional or alternative method application.

Acknowledgements. Sincere thanks are due to Professor John Mylopoulos for his encouragement and guidance. Gratitude is also due to Eric Yu, Brian Nixon, John Nestor and Tae Soon Park for their valuable comments. Ian Maione contributed to the prototype implementation, and Thomas Rose and David Lauzon provided technical advice throughout the implementation. Thanks are extended to the anonymous referees for their encouragement and comments.

References

1. J. A. Adam, "Threats and Countermeasures," *IEEE Spectrum,* vol. 29, no. 8, Aug. 1992, pp. 21–28.
2. E. Amoroso, T. Nguyen, J. Weiss, J. Watson, P. Lapiska, and T. Starr, "Towards an Approach to Measuring Software Trust," *Proc. IEEE Symp. Security and Privacy,* May 1991, pp. 198–218.
3. *BIM Prolog 3.1 Manual.* BIM sa/nv, Belgium, 1992.
4. T. C. Vickers Benzel, "Developing Trusted Systems Using DOD–STD–2167A," *5th Annual Computer Security Appl. Conf.*, Tucson, Arizona, Dec. 4–8, 1989, pp. 166–176.
5. B. W. Boehm, "A Spiral Model of Software Develo ent and Enhancement", Vol. 11, No. 4, *ACM Software Eng. Notes*, Aug. 1986.
6. T. Bui and T. R. Sivasankaran, "Cost-Effective Modeling for a Decision Support System in Computer Security," *Computers & Security,* vol. 6, no. 2, Apr. 1987, pp. 139–151.
7. Canadian System Security Centre, *The Canadian Trusted Computer Product Evaluation Criteria,* Version 2.0. Ottawa, Apr. 1992.
8. Canadian Bankers' Association, *MasterCard and Visa Statistics*, Toronto, Dec. 1991.
9. L. Chung, "Representation and Utilization of Non-Functional Requirements for Information System Design." In R. Anderson, J. A. Bubenko, Jr., A. Sølvberg (Eds.), *Advanced Information Systems Eng.,* Proc., 3rd Int. Conf. CAiSE '91, Trondheim, Norway, May 13–15, 1991. Berlin: Springer-Verlag, 1991, pp. 5–30.
10. K. L. Chung, P. Katalagarianos, M. Marakakis, M. Mertikas, J. Mylopoulos and Y. Vassiliou, "From Information System Requirements to Designs: A Mapping Framework," *Information Systems,* vol. 16, no. 4, 1991, pp. 429–461.
11. D. D. Clark and D. R. Wilson, "A Comparison of Commercial and Military Computer Security Policies," *Proc. IEEE Symp. Security and Privacy,* 1987, pp. 184–194.
12. S. D. Crocker, "Software Methodology for Development of a Trusted BMS: Identification of Critical Problems," *5th Annual Computer Security Appl. Conf.*, Tucson, Arizona, Dec. 4–8, 1989, pp. 148–165.
13. U. S. Department of Defense, *Military Standard: Defense System Software Development,* Feb. 29, 1988.
14. A. Dardenne, A. van Lamsweerde and S. Fickas, "Goal-directed Requirements Acquisition," *Science of Computer Programming,* to appear; earlier version appeared in *6th Int. Workshop on Software Specification and Design,* Como, Italy, 1991.
15. B. Di Vito, C. Garvey, D. Kwong, A. Murray, J. Solomon and A. Wu, "The Deductive Theory Manager: A Knowledge Based System for Formal Verification," *Proc. IEEE Symp. Security and Privacy,* 1990, pp. 306–318.
16. E. B. Fernandez, E. Gudes and H. Song, "A Security Model for Object-Oriented Databases," *Proc. IEEE Symp. Security and Privacy,* 1989, pp. 110–115.

17. S. Greenspan, *Requirements Modeling: A Knowledge Representation Approach to Software Requirements Definition.* Ph.D. Thesis, Dept. of Computer Science, Univ. of Toronto, 1984.
18. U. Hahn, M. Jarke and T. Rose, "Teamwork Support in a Knowledge-Based Information Systems Environment," *IEEE Trans. Software Eng.*, vol. 17, no. 5, May 1991, pp. 467–482,
19. H. R. Hartson and D. K. Hsiao, "Full Protection Specification in the Semantic Model for Database Protection Languages," *Proc. ACM Annual Conf.*, Houston, TX, Oct. 1976, pp. 90–95.
20. German Information Security Agency, *Criteria for the Evaluation of Trustworthiness of Information Technology (IT) Systems,* 1st Version, Bundesanzeiger, Köln, Germany, 1989.
21. Office for Official Publications of the European Communities, *Information Technology Security Evaluation Criteria, Provisional Harmonised Criteria,* Version 1.2, June 1991, Luxembourg.
22. M. Jarke, J. Mylopoulos, J. W. Schmidt, Y. Vassiliou, "DAIDA: An Environment for Evolving Information Systems," *ACM Trans. Information Systems*, vol. 10, no. 1, Jan. 1992, pp. 1–50.
23. M. Jarke (Ed.), *ConceptBase V3.1 User Manual*, 1992.
24. M. Jarke, J. Bubenko, C. Rolland, A. Sutcliffe and Y. Vassiliou, "Theories Underlying Requirements Engineering: An Overview of NATURE at Genesis," *IEEE Int. Symp. Requirements Eng.*, Jan. 1993, pp. 19–31.
25. E. S. Lee, P. I. P. Boulton, B. W. Thomson, and R. E. Soper, *Composable Trusted Systems,* Tech. Report CSRI–272, Computer Systems Research Institute, Univ. of Toronto, May 31, 1992.
26. J. Lee, *A Decision Rationale Management System: Capturing, Reusing, and Managing the Reasons for Decisions*, Ph.D. Thesis, MIT, 1992.
27. A. Marmor-Squires, B. Danner, J. McHugh, L. Nagy, D. Sterne, M. Branstad, and P. Rougeau, "A Risk Driven Process Model for the Development of Trusted Systems," *5th Annual Computer Security Appl. Conf.*, Tucson, Arizona, Dec. 4–8, 1989, pp. 184–192.
28. N. S. Matloff, "Another Look at the Use of Noise Addition for Database Security," *Proc. IEEE Symp. Security and Privacy*, 1986, pp. 173–180.
29. T. J. McCabe and G. G. Schulmeyer, "The Pareto Principle Applied to Software Quality Assurance," In G. Gordon Schulmeyer and James I. McManus (Eds.), *Handbook of Software Quality Assurance,* New York: Van Nostrand Reinhold, 1987, pp. 178–210.
30. J. D. Moffett and M. S. Sloman, "The Source of Authority for Commercial Access Control," *IEEE Computer*, vol. 21, no. 2, Feb. 1988, pp. 59–69.
31. J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis, "Telos: Representing Knowledge about Information Systems," *ACM Trans. Information Systems,* vol. 8, Oct. 1990, pp. 325–362.
32. J. Mylopoulos, L. Chung and B. Nixon, "Representing and Using Non-Functional Requirements: A Process-Oriented Approach," *IEEE Trans. Software Eng.*, vol. 18, no. 6, June 1992, pp. 483–497.
33. P. G. Neumann, "On Hierarchical Designs of Computer Systems for Critical Applications," *IEEE Trans. Software Eng.*, SE–12, no. 9, Sept. 1986, pp. 905–920.
34. P. G. Neumann (compiler), "Illustrative Risks to the Public in the Use of Computer Systems and Related Technology," *ACM Software Eng. Notes*, vol. 16, no. 1, Jan. 1991, pp. 2–9.

35. N. Nilsson, *Problem-Solving Methods in Artificial Intelligence.* New York, McGraw-Hill, 1971.
36. B. Nixon, "Dealing with Performance Requirements During the Development of Information Systems", *IEEE Int. Symp. Requirements Eng.*, Jan. 1993, pp. 42–49.
37. D. B. Parker, "The Many Faces of Data Vulnerability," *IEEE Spectrum,* vol. 21, no. 5, May 1984, pp. 46–49.
38. D. B. Parker, "Restating the Foundation of Information Security," *2nd Annual North American Information System Security Symp.,* Toronto, Oct. 21–23, 1991.
39. T. S. Perry and P. Wallich, "Can Computer Crime be Stopped?", *IEEE Spectrum,* Vol. 21, No. 5, May 1984, pp. 34–45.
40. C. P. Pfleeger, *Security in Computing,* Englewood Cliffs, NJ: Prentice Hall, 1989.
41. C. Potts and G. Bruns, "Recording the Reasons for Design Decisions," *Proc. 10th Int. Conf. Software Eng.*, 1988, pp. 418–427.
42. H. A. Simon, *The Sciences of the Artificial,* 2nd ed., Cambridge, MA: The MIT Press, 1981.
43. G. W. Smith, "Multilevel Secure Database Design: A Practical Application," *5th Annual Computer Security Appl. Conf.*, Tucson, Arizona, Dec. 4–8, 1989, pp. 314–321.
44. G. Steinke, "Design Aspects of Access Control in a Knowledge Base System," *SECURICOM '90,* Paris, March, 1990.
45. U.S. Department of Defense, *Trusted Computer Systems Evaluation Criteria,* DOD 5200.28–STD, Dec. 1985.
46. F. G. Tompkins and R. Rice, "Integrating Security Activities Into the Software Development Life Cycle and the Software Quality Assurance Process," *Computers & Security,* vol. 5, no. 3, Sept. 1986, pp. 218–242.
47. Y. Vassiliou, M. Marakakis, P. Katalagarianos, L. Chung, M. Mertikas and J. Mylopoulos, "IRIS — A Mapping Assistant for Generating Designs from Requirements." In *Proc., CAiSE '90, 2nd Nordic Conf. Advanced Information Systems Eng.,* Stockholm, May 1991. Berlin: Springer-Verlag, 1991, pp. 307–338.
48. Visa Canada Association, *Visa Canada 1990 Regional Report,* Toronto, 1990.
49. C. Wood, E. B. Fernandez and R. C. Summers, "Data Base Security: Requirements, Policies, and Models," *IBM Syst. J.*, vol. 19, no. 2, 1980, pp. 229–252.