Requirements Eng (2006) 11: 138–151
DOI 10.1007/s00766-005-0023-4

ORIGINAL ARTICLE

Charles B. Haley · Robin C. Laney
Jonathan D. Moffett · Bashar Nuseibeh

Using trust assumptions with security requirements

Received: 25 July 2005 / Accepted: 28 July 2005 / Published online: 13 December 2005

Abstract Assumptions are frequently made during requirements analysis of a system about the trustworthiness of its various components (including human components). These trust assumptions, whether implicit or explicit, affect the scope of the analysis, derivation of security requirements, and in some cases how functionality is realized. This paper presents trust assumptions in the context of analysis of security requirements. A running example shows how trust assumptions can be used by a requirements engineer to help define and limit the scope of analysis and to document the decisions made during the process. The paper concludes with a case study examining the impact of trust assumptions on software that uses the secure electronic transaction specification.

1 Introduction

An important constituent of a system's requirements is its *security requirements*. Security requirements arise because stakeholders assert that some objects, be they tangible (e.g., cash) or intangible (e.g., information and state), have direct or indirect value. Objects valued in this way are called *assets* [1], and the stakeholders naturally wish to protect these assets from harm. For example, tangible assets might be destroyed, stolen, or modified; information assets might be destroyed, revealed, or modified; and state might be modified, revealed, or disputed (this list is not exhaustive). An asset can also be used to cause indirect harm, such as to reputation. Security requirements are used to restrict the number of cases wherein these undesirable outcomes can take place, and should be elicited and enumerated during a requirements engineering process.

One definition of requirements engineering is that it is "concerned with the real-world goals for functions of and constraints on software systems" [2]. This definition is interesting because it firmly places requirements engineering in the *world*, as opposed to in the software. Two additional definitions extend this first definition, stating that requirements engineering is about *systems* in the world: "requirements engineering is concerned with the identification of the goals to be achieved by the envisioned system" [3], and "requirements definition is the task of gathering all of the relevant information to be used in understanding a problem situation prior to system development" [4]. The extension is significant. A *system* comprises not only software, but also all the diverse constituents needed for it to achieve its purpose. For example, a computing system clearly includes the computers, but also incorporates real-world elements such as the people who will use, maintain, and depend on the system; the physical and logical environment within which the system will exist; and any systems already in place.

Extending requirements engineering from the software to include the system is very significant to security. When considering security, all of the elements in a system participate, and therefore production of security requirements demands a system-level analysis [5–7]. If the requirements engineer considers only the software, he or she is limited to production of general goals of the form *X must not occur*. Nothing can be said about how such goals are satisfied, or even if they can be satisfied, as the requirements engineer cannot analyze interactions between the software and unknowns.

When operating in a systems context, the requirements engineer must determine which real-world elements are to be included in the analysis. An extreme view is that every atom in the universe is part of every system, and therefore the analysis must include everything made

C. B. Haley (✉) · R. C. Laney · J. D. Moffett · B. Nuseibeh
Department of Computing, The Open University, Walton Hall, MK7 6AA Milton Keynes, UK
E-mail: C.B.Haley@open.ac.uk
E-mail: R.C.Laney@open.ac.uk
E-mail: J.Moffett@open.ac.uk
E-mail: B.Nuseibeh@open.ac.uk

of atoms. As this is clearly impractical, the analyst must define the *context* within which requirements analysis takes place by selecting the *domains* (the aforementioned real-world constituents) that are considered pertinent [8, 9]. In doing so, the analyst reduces the size of the context to those domains relevant to the problem.

One factor influencing an analyst's choice about whether or not a domain is relevant to a system's security, and therefore to be included in the context, is the analyst's set of *trust assumptions* [10]. Trust assumptions are explicit or implicit choices to trust some characteristics of domains. These assumptions can have a significant impact on the security of a system. For example, most analysts implicitly assume that the compiler is not a security risk; it would never occur to them to include it in the analysis. Thompson demonstrated that this assumption is not necessarily justified by showing how a compiler could introduce trapdoors into applications [11]. Viega et al. [10] say in that "application providers often assume that their code will execute in a non-hostile environment", and then show how this assumption leads to security breaches by using an example of hiding 'secrets' in code where the secrets are not truly hidden. These two examples illustrate how the requirements engineer's implicit trust of some domains in the environment can introduce unknown amounts of risk into the system. Viega et al. went as far as to say "without recognizing all the entities and their trust relationships in a software system during the requirements phase of a project, that project is doomed from the start."

Although the above examples demonstrate the need to capture and analyze trust assumptions, little work has been done exploring how to find, represent, and quantify them; and then to analyze their effect on the system under discussion. The contribution of this work is a first step toward correcting this lack, investigating *trust assumptions* within the framework of representing security requirements as *constraints* on a system's functionality [7], using *problem frames* [9] to describe the system context, and *threat descriptions* [12] to represent threats.

The remainder paper is organized as follows. Section 2 provides some background material on problem frames. Section 3 discusses security requirements, Sect. 4 describes trust assumptions, and Sect. 5 presents some examples. Section 6 describes a case study, Sect. 7 presents related work, and Sect. 8 concludes.

2 Problem frames

The view of requirements exemplified by problem frames [9] is that a system is intended to solve a *problem* in a *context* of *real-world physical domains*, where the context includes system design decisions. One uses problem frames to analyze the problem in terms of the context and the design decisions the context represents. The context contains domains, which are physical elements around which the system (not just the software) will be built. The problem frames approach differs from some other approaches (e.g., KAOS [13]) that hold that a requirements engineer should reason about a system's characteristics without using a physical model of the world; under this view, a requirements engineer enumerates goals for a system under consideration and produces a temporal logic model of the system's desired behavior. We will show how using the real-world system perspective provided by problem frames assists with the determination of security requirements.

In the problem frames universe, all computing problems involve the interaction between domains in the world. Domains are tangible (e.g., people, equipment, networks) but may contain intangibles (e.g., information). Every domain has *interfaces*, which are defined by the *phenomena* visible to other domains. Descriptions of phenomena of given (existing) domains are *indicative*; the phenomena and resulting behavior can be observed. Descriptions of phenomena of designed domains (domains to be built as part of the solution) are *optative*; one hopes to observe the phenomena in the future.

One special domain is the *machine*; the domain that performs the transformations to satisfy the *requirement*. The interplay of phenomena between the machine and its connected domains defines what the machine has to work with to satisfy the requirement. The interplay of phenomena is a *specification*, describing how the requirements are satisfied [14]. The difference between specification and requirement is important. A specification is an expression of the behavior of phenomena visible at the boundary of the domains, whereas a requirement is a description of the problem to be solved. For example, in the context of a building we might find the requirements 'permit passage from one room to another' and 'physically separate rooms when possible'. Clearly the problem involves something like doors. Equally as clearly, it does not specify that doors be used, nor does it specify internal phenomena or behavior. It is up to the designer (the architect in this case) to choose the 'door' domain(s) for the system. One might satisfy the requirement with a blanket, an automatic door, a futuristic iris, or a garden maze. Each domain implementation presents different phenomena at its boundaries (i.e., they work differently), and the resulting system specification must consider these differences. However, the requirement does not change.

There are two fundamental diagram types in a problem frames analysis, *context diagrams* and *problem diagrams*. A context diagram shows the domains of interest in a system, how the domains are interconnected, and optionally the phenomena on the interfaces between domains. A problem diagram is used to describe a *problem* in the system; the problem is expressed by a requirement. The problem diagram is a projection of the context, showing only the domains or groups of domains of interest to the particular problem. A *problem frame diagram* is a kind of problem diagram that describes the problem as one of a known set of problem classes, showing how a given requirement is to be satisfied using the pattern that the problem class represents.

We use only context diagrams and problem diagrams in this paper. We do not (yet) make use of problem classes. Doing so is a subject of future investigation.

Figure 1 shows a context diagram for a system that will be used as an example in Sects. 3 and 4 of this paper. The system is a subset of a Human Resources system having four problems/requirements:

- Salary, personal, and benefits information shall be able to be entered, changed, and deleted by HR staff. This information is referred to as *payroll information*.
- Each employee shall be able to view a subset of his or her own personal and benefits information.
- Users shall have access to kiosks located at convenient locations throughout the building and able to display an 'address list' subset of personal information consisting of any employee's name, office, and work telephone number.
- At most 24 hours of modifications to information shall be vulnerable to loss.

The context diagram could be broken down into four problem diagrams, one for each requirement. In the interest of brevity, only one of the problem diagrams, the one for the third requirement, is discussed in this paper. Figure 2 shows the problem diagram for this requirement (the 'address list' function). Phenomena have been intentionally omitted. Security will be discussed starting in Sect. 3.

3 Security requirements

Security requirements are often defined as "restrictions or constraints" placed on system services [15–17]. We slightly restate this definition: security requirements express constraints on the behavior of a system sufficient to satisfy security goals [7]. The constraints are intended to limit undesired system behavior as much as possible while still satisfying the system's requirements. For example, a goal for an ATM might be *provide cash to customers*. This goal is obviously overly broad from a security point of view. By providing constraints such as

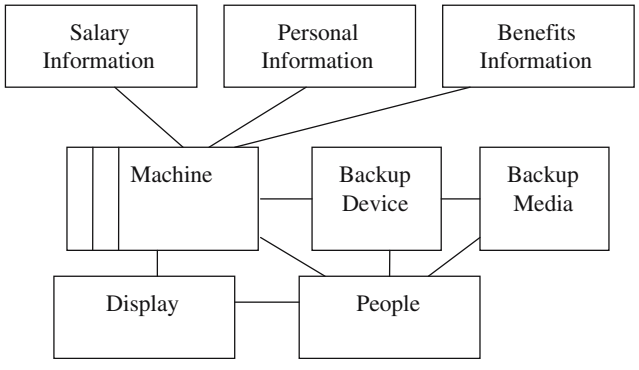

Fig. 1 Example context diagram

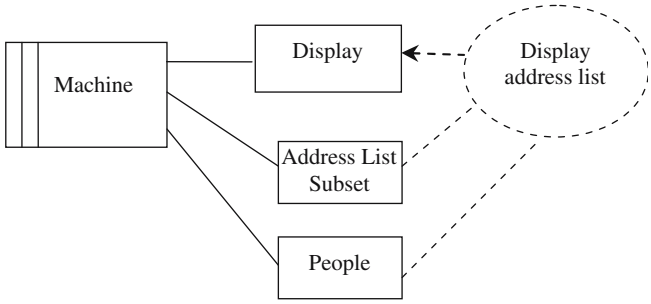

Fig. 2 Address list

only if customer physically possesses an ATM card associated with the account and *only if customer provides the correct PIN* (two security requirements), the circumstances under which cash is to be provided are reduced.

Security requirements are added to functional requirements (requirements that say what the system is to do) in order to prevent harm through misuse of *assets* [7, 12]. An asset is something in the context of the system, tangible or not, that is to be protected. A *threat* is the potential for abuse of an asset that will cause harm. A *vulnerability* is a weakness in the system that an attack exploits to realize a threat. Security requirements are constraints on functional requirements, intended to reduce the scope of vulnerabilities. Thus, security requirements stipulate the location and elimination of vulnerabilities that an attacker can exploit to carry out threats.

The security community has enumerated some general security goals, labeling them with the acronym CIA, and more recently another A ([18] and other security textbooks):

- Confidentiality: Ensure that an asset is visible only to actors authorized to see it. Confidentiality is larger than 'prevent read access to a file'. For example, it includes controlling visibility of a data stream on a network, and of papers on someone's desk.
- Integrity: Ensure that the asset is not corrupted. Integrity is larger than 'prevent write access to a file', for example including ensuring that transactions that should not occur indeed do not, that the contents of backup media are not changed, that incorrect entries in a paper-based accounting system are not made, and data streams are not modified between their endpoints.
- Availability: Ensure that the asset is readily accessible to agents that need it. A counterexample is preventing a company from doing business by denying it access to something important, such as access to its computer systems or its offices.
- Authentication: Ensure that the provenance of the asset or actor is known. A common example is the simple login. More complicated examples include mutual authentication (e.g., exchange of cryptography keys), and intellectual property rights management.

By connecting these general goals to assets, and then postulating an *action* that would violate the goal, one can construct descriptions of possible threats on assets. These *threat descriptions* [12] are phrases of the form *performing action X on/to/with asset Y could cause harm Z*. Threat descriptions permit a form of asset-centered threat modeling, and are represented by a three-element tuple: the asset, the action that will exploit the asset, and the subsequent harm. They are generated by enumerating the assets involved in the system, then for each asset, listing the *actions* that exploit the asset to cause direct or indirect harm. For example, one can imagine *erasing* (the action) the *customer records* (the asset) of a company to cause *loss of revenue* (the harm).

Referring to the HR example presented in Sect. 2, some possible threat descriptions are:

- Exposing salary data could reduce employee morale, lowering productivity.
- Changing salary data could increase salary costs, lowering earnings.
- Exposing addresses (to headhunters) could cause loss of employees, raising costs.

The requirements engineer examines each context diagram in the problem (or problem diagram, if they are available) to see if the asset mentioned in the threat description is involved in the problem. To be involved, the asset must be either a domain or contained in a domain, or be found in the phenomena. If the asset is found, then the requirements engineer must operationalize the 'avoid' goal represented by the threat description to produce constraints on functionality that ensure that the asset cannot be abused in the way the threat description requires. These constraints are security requirements [7], which, like all requirements, must be satisfied by the system. They are satisfied by changes and/or additions to the domains or phenomena, by changing the behavior of the domains in the context, by requiring specific behavior of the machine, or by adding trust assumptions explaining why undesired behavior is believed not to occur.

Without going into the mechanics of how the security requirements are determined (see [12] and [7]), analysis of Fig. 2 shows that in order to maintain confidentiality and integrity of the data, a constraint must be added—only authorized people may see the data. To satisfy this constraint, the network needs to be protected and the authorization of users must be verified. The former, protecting the network, is accomplished using encryption. The resulting problem diagram is shown in Fig. 3. The security requirement has been added to the oval, phenomena have been added to support encryption, and the encrypted network has been made explicit. The latter, authorization of users, is discussed in Sect. 5.

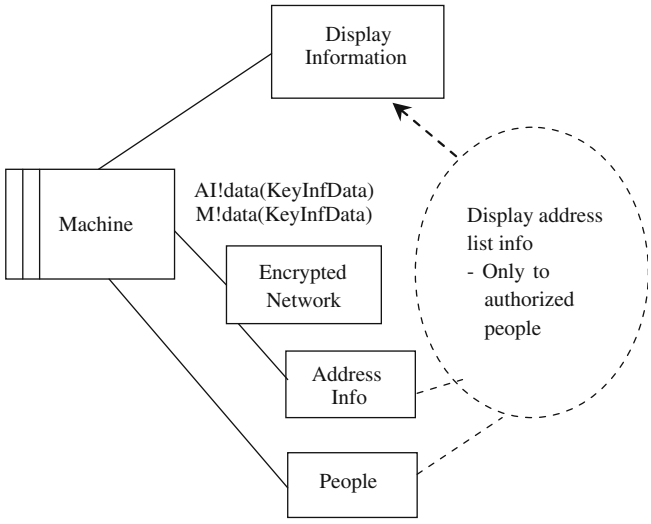

Fig. 3 Address list revisited

4 Trust assumptions

We define a trust assumption as an *assumption* by a requirements engineer that, in order to satisfy a security requirement, the membership or specification of a domain can depend on certain properties. The requirements engineer *trusts* the *assumption* to be true. These assumed properties or assertions act as *domain restrictions;* they restrict the domain in some way that contributes to the satisfaction of the security requirement.

4.1 Purpose of trust assumptions

The requirements engineer is responsible for constructing an argument that security requirements are satisfied—the *correctness argument* described in [7], hereafter called the *satisfaction argument*. We prefer the term 'satisfaction argument' because it implies a weaker standard, something appropriate when working with security. One cannot prove that a system is secure. One is instead limited to arguing that, in the context of the system and with information known at that point, the system is adequately secure.

In many cases, the satisfaction argument cannot be made without depending on domain properties that cannot be verified with the information in hand; the scope of the analysis must be expanded to include anything that the domain in question depends upon. The requirements engineer has a choice, either to expand the scope as necessary to verify the properties, which is a recursive process, or to add a trust assumption that asserts that the properties are valid. By choosing the trust assumption, the requirements engineer ends the recursion and explicitly limits the scope of the analysis.

To illustrate this choice, assume the existence of a security requirement stipulating that the computers operate for at least 8 h in the event of a power failure (an availability requirement). The requirements engineer can satisfy this requirement by adding backup generators to the system. Appropriate phenomena would be added so that the machine can detect the power loss, control the generators, detect going beyond 8 h, etc. In most situations, the requirements engineer can trust the manufacturer of the generators to supply equipment that does not intentionally permit an attacker to take control of the generators to prevent them from operating (a denial of service attack). The analyst trusts the behavior of the generators, and adds a trust assumption to this effect. By adding the trust assumption, the requirements engineer does not need to include the manufacture of the generators in the analysis. The analyst uses the trust assumption to limit the scope of the analysis.

As explained above, trust assumptions contribute to the satisfaction of security requirements. There is not necessarily a one-to-one correspondence between a trust assumption and the security requirements satisfied. Several trust assumptions may be necessary to satisfy a security requirement (an *and* decomposition), any one of several trust assumptions may be sufficient to satisfy a security requirement (an *or* decomposition), or some combination of the two. In addition, one trust assumption may play a role in satisfying multiple security requirements.

4.2 The 'trust' in trust assumptions

We use a variant of the definition of trust proposed by Grandison and Sloman [19]: "[Trust] is the quantified belief by a trustor with respect to the competence, honesty, security and dependability of a trustee within a specified context". In our case, the *requirements engineer* trusts some domain to participate 'competently and dependably' in the satisfaction of a security requirement in the context of the problem.

In the Grandison and Sloman definition, the quantification of a trust assumption represents a level of confidence that the trust assumption is valid. Said another way, it represents the risk that including the trust assumption and thereby limiting the analysis may not be justified. In our work, the quantification is binary; the trust assumption is thought to be valid, or it is not. Taking this restriction into consideration, the variant of the definition we are using begins with "[Trust] is the [] belief by ..." (the word *quantified* is removed).

The Thompson example in Sect. 1 ([11]) gives us an example of a trust assumption. An analyst's (probably implicit) trust of the compiler vendor not to include trapdoor generators in the compiler may be misplaced. If the compiler has been compromised, then some number of vulnerabilities may come into existence, such as the existence of a universal password, denial-of-service traps, or information leaks. Successful attacks using these vulnerabilities will have some impact on the organization: i.e., they will cause harm. The organization must decide whether the risk presented by the vulnerabilities that might come into existence if the trust assumption is not valid is sufficient to justify the time and expense of the expansion of the analysis required to validate the compiler.

The risk presented by a trust assumption is not the same as the risk presented by a vulnerability. The risk associated with a vulnerability measures the likelihood that the vulnerability can be successfully exploited. The risk in a trust assumption measures how likely it is that the vulnerability exists as a consequence of the trust assumption being invalid. As the example in the previous paragraph shows, the two measures are independent. If a compiler has been compromised to modify the password checker of the login program (the case described by Thompson) but the login program is not used in a system, then the risk presented by the vulnerability is nil, regardless of the validity of a trust assumption stating that the compiler vendor can be trusted.

An analysis and discussion of risk analysis is outside the scope of this paper. However, where appropriate, certain risk factors are introduced without further justification.

4.3 Representation of trust assumptions

A trust assumption consists of the following information:

- Identification of the dependent domain. This is the domain being restricted by the trust assumption.
- Effect of the trust assumption. The trust assumption restricts membership of the dependent domain, phenomena on the interfaces of the dependent domain, or some combination of the two. Note that phenomena restrictions can be an assertion that some phenomena will not appear on the interface, or will only occur in a specific sequence/interchange.
- Narrative description of the restriction(s). If the trust assumption restricts domain membership, then describe the membership of the domain before and after application of the restriction. If the trust assumption restricts phenomena, then describe the restriction and its effect, if any, on the valid interplay of phenomena. When discussing the validity and effect of the restrictions in this section, the analyst should take the position that the trust assumption is valid. The validity of the trust assumption is examined below.
- Preconditions. Some trust assumptions may be valid only if some other conditions are true. Some examples might be the earlier application of some other trust assumption to the dependent domain and/or the existence of domains not otherwise included in the analysis.
- Justification for the inclusion of the trust assumption. This is not a justification of the restrictions, but is instead an informal discussion of why the trust assumption should be considered valid. If there are risks associated with the trust assumption, they should be listed and discussed.

- List of security requirements (the constraints) that this trust assumption satisfies partially or completely. If the trust assumption is part of an *and* or an *or* decomposition, its siblings should be listed.

Diagrammatically, a trust assumption is represented by an arc from the dependent domain to an oval containing a short summary of the properties being depended upon.

5 Examples of trust assumptions

Returning to the example begun in Sect. 3, trust assumptions must be added to the diagram in order to complete the picture. For example, the analysis does not explain why the encrypted network is considered secure or how address information is to be protected. We must also determine whether a user is authorized to see the information. We look at two cases: using authentication and using the building's physical security.

5.1 Using authentication

Authentication is a system-level problem involving many potentially complex processes. In order to derive requirements for authentication, the requirements engineer must understand how users are to be authenticated, perhaps based on cost, risk, and ease-of-use factors. In many cases, the decision is imposed by the IT organization. In other cases, the stakeholders will insist on a particular method. Regardless of where the decision comes from, the phenomena involved must be added to the problem diagram.

Figure 4 presents a login and password solution, along with some trust assumptions related to authentication and to protection of access to data. The requirements engineer is convinced by the IT organization that the encryption system is strong, and decides that the encrypted network connection domain does not require further analysis. TA1.1 is added to document this decision. TA1.4 is added to document accepting that the key system tells the data server the access level of the client machine; the behavior of the server is constrained to refuse to supply information above the access level indicated by the keys. TA1.2 indicates that the requirements engineer chooses to trust the systems administrators to properly manage access credentials, constraining the domain to contain accurate information.

Finally, the engineer assumes that employees will keep their credentials confidential (TA1.3), constraining the 'people' domain to be 'people with their own credentials'. We expand this trust assumption to include the information listed in Sect. 4.3.

- The dependent domain: People.
- Effect: The People domain is restricted to contain individuals who are using their own credentials.
- Explanation: Before the restriction, the people domain can contain individuals who have credentials that may or may not have been allocated to them. After application of the restriction, the people domain can contain only individuals who have credentials allocated to them and who are using their own credentials.
- Preconditions: This trust assumption depends on TA1.2—that administrators will not expose one person's credentials to another person.
- Justification: The employees of this company are all stockholders who stand to benefit greatly from the success of the company, and therefore will respect the security rules out of self interest. The employees are also all security experts who understand at a visceral level the reasons for keeping credentials private. For these reasons we assume that they will not expose their credentials, either accidentally or intentionally.
- Security requirements partially satisfied: Address information shall be restricted to employees.

5.2 Example: using building security

The login/password scheme may be unacceptable to the customer. The IT department may refuse, saying that giving all employees authentication information would be too costly. The stakeholders may refuse, insisting that

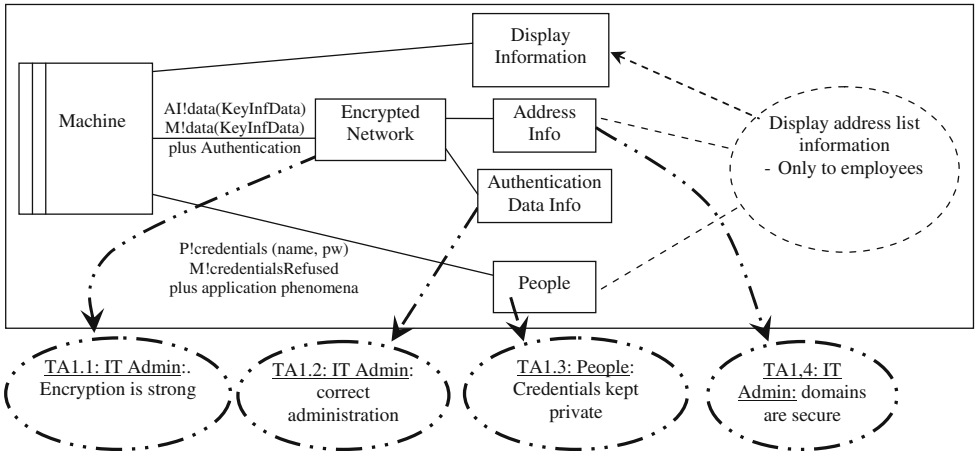

Fig. 4 Address list with authentication

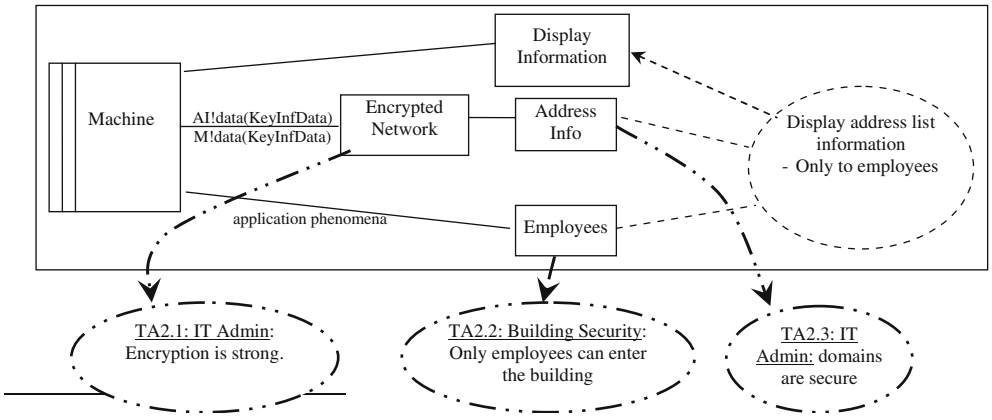

Fig. 5 Address list with door security

requiring a login would make the system too hard to use. An alternate solution is to make use of the fact that the doors of the building are protected by a security guard; the guard restricts entrance to authorized personnel. The security manager agrees that the security guard can stand in for authentication.

Figure 5 presents this alternate solution. Authentication is removed and trust assumption TA2.2 is added, having the effect of changing the *People* domain to *Employees* by restricting membership to *people allowed to enter the building by the security system*. The authentication-related trust assumptions are removed. It is up to the customer to decide if the associated risk profile is acceptable.

The details of TA2.2 are:[1]

- The dependent domain: Employees.
- The effect: The original People domain is restricted to contain Employees.
- Explanation: Before the restriction, the people domain contains individuals, whether or not they can actually enter the building. After application of the restriction, the people domain contains only employees, the permitted occupants of the building.
- Preconditions. This trust assumption depends on the existence and operation of a building security system.
- Justification: The entrances to the building are protected by professional security staff who verify that people entering the building are employees. If a person who is not an employee is permitted entrance, that person is escorted by a member of security staff while in the building.
- Security requirements partially satisfied: Address information shall be restricted to employees.

5.3 Trust assumptions as domain restrictions

The above examples support our position that trust assumptions are domain restrictions. The clearest example is the security system trust assumption (TA2.2 in Fig. 5); it restricts the membership of the *People* domain to people acceptable to the door guard, effectively converting the domain to *employees*. The other trust assumptions play a similar role. For example, TA1.2 (IT Admin*: correct administration*) trust assumption limits the number of people having acceptable credentials.

TA2.1 and TA2.3 (IT Admin*: domains are secure* and the IT Admin*: encryption is strong)* illustrate restricting behavior (specification) as opposed to membership. In the case of TA2.3, the behavior of the Address Info domain is restricted to supply information only at the level indicated by the key; the assertion is that no other case exists. In the case of TA2.1, the domain is restricted to supplying 'in the clear' information to holders of valid encryption keys; the assertion is that no alternate method to obtain the information exists.

6 Case study

The secure electronic transaction (SET) specifications [20–22] describe a set of mechanisms intended to provide an acceptable level of security for on-line purchasing. This case study looks at incorporating the SET specification into software to support cardholder-side payment authorization. There is one requirement (in the problem frames sense): Complete the Purchase. The study considers one asset, *customer account information* (*CAI*), and one derived security goal *Purchases shall be authorized*. Several trust assumptions are derived during the analysis.

To derive the trust assumptions, we first determine the threat descriptions, then negate them to express the security requirements (the constraints). Two threat descriptions are used in this case study: *exposure of CAI could lead to financial loss* (from the goal of confidentiality), and *unauthorized use of cardholder credentials could lead to financial loss* (from the goal of integrity). Appropriate security requirements (constraints) are added to the requirements: SR1: *only authorized users may know CAI* and SR2: *only authorized individuals may use the cardholder credentials*. The resulting trust assumptions needed

[1]For space reasons, we will not include any further long descriptions of the trust assumptions.

to satisfy the security requirements will be listed in a later section.

6.1 Secure electronic transaction overview

Secure electronic transaction describes a series of operations between players in an electronic purchase transaction using a credit card. In SET, a *cardholder* requests a cryptographic certificate from a *certificate authority* (CA). The CA verifies that the cardholder has a credit card account with an *issuer*, and then supplies a certificate. The cardholder can subsequently use the certificate to make purchases from a *merchant*. The merchant uses a *payment gateway* to pass the transaction to the *acquirer* (the merchant's bank) for collection. The acquirer normally operates the payment gateway. Figure 6 presents a simplified version of the SET "processing flows" (terminology from [20]), showing the players and the messages they interchange. Several SET messages and fields that do not have a direct bearing on this discussion have been omitted from the diagram, in particular the obtaining of certificates and private keys, and the initial verification of cardholder information. In addition, the diagram shows the merchant using the CAI, which although optional in SET is the technique that the SET specification claims will be the most often used ([21], p 14).

6.2 Secure electronic transaction-identified security assumptions

The SET specifications make the following security-related assumptions about the SET environment relevant to this case study. They are relevant because they point us at vulnerabilities considered by the writers of the SET specification.

- SA1: The cardholder ensures that no one else has access to his/her private key ([20], p 16). In particular, SET software vendors shall "ensure that the certificate and related information is stored in a way to prevent unauthorized access" ([20], p 46).
- SA2: Cardholder, merchant, and payment gateway machines are free of viruses and trojan horses, and are not susceptible to being hacked ([20], p 11).
- SA3: Programming methods and the cryptographic system, and in particular the random number generators, are of the highest quality ([20], p 16).
- SA4: The merchant's system stores account information in an encrypted form, and if possible off-line or behind a firewall ([21], p 39).

6.3 The initial problem diagram

There is only one requirement in this case study and therefore only one problem diagram. The context we are using does not show any analysis of the 'shopping' process, instead focusing on the point where a purchase is completed. Taking the SET processing flows into consideration, a first-cut problem diagram is shown in Fig. 7.

Recall from Sect. 6 the two security requirements we must satisfy: SR1, *only authorized users may know CAI* and SR2: *only authorized individuals may use the cardholder credentials*. CAI is made visible by the CAI phenomena in the problem diagram, and the asset *cardholder credentials* is stored in the machine. Our goal is to generate a satisfaction argument that these security requirements are satisfied.

By tracing the CAI through the problem diagram, we see that it must reside in unknown form within the Machine domain. According to the SET specification, the CAI must be encrypted between the machine and the merchant. There is nothing in the problem that indicates that the user or the merchant should be able to see the CAI. We can say the same thing for cardholder credentials. These observations and the security assumptions SA1–SA4 lead us to make the following trust assumptions:

- TA1-1—satisfaction of SR2: As the credentials are stored on the machine, and as there is no apparent

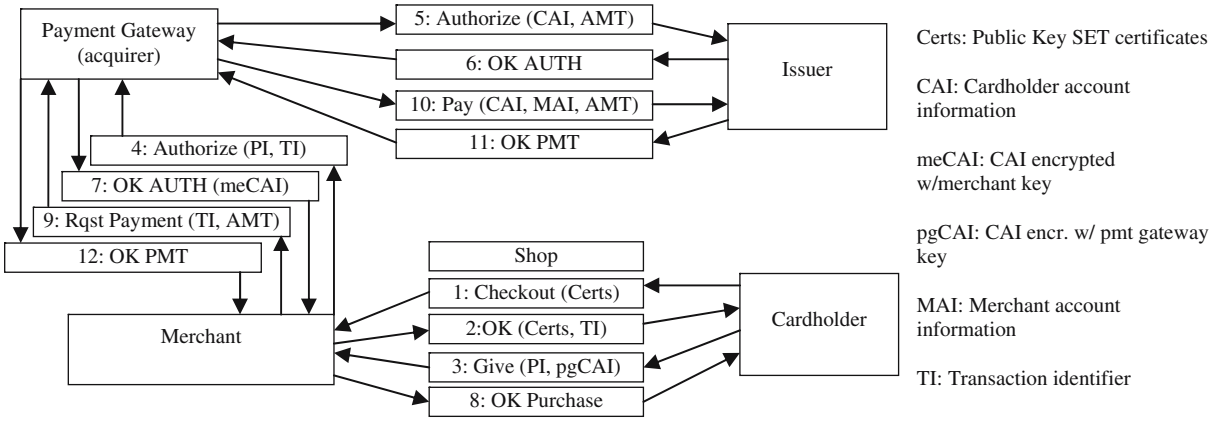

Fig. 6 Simplified SET processing flows

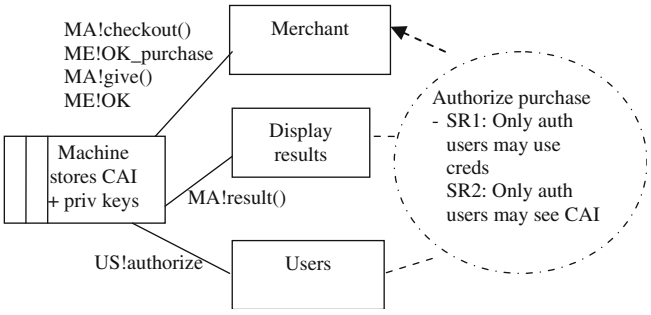

Fig. 7 Purchase problem

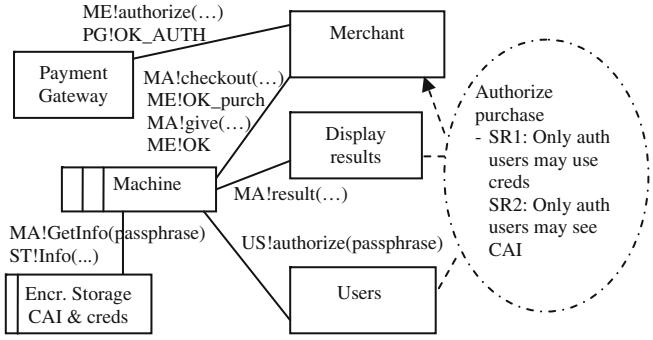

Fig. 8 Purchase problem (again)

way to limit who can access these credentials, SA1 forces us to assume that the domain *Users* in the problem contains only individuals authorized to use the credentials.

- TA1-2—satisfaction of SR1: The CAI and credentials are not visible outside the machine (SA2).
- TA1-3—satisfaction of SR1: The generated symmetric encryption keys are cryptographically secure (SA3).
- TA1-4—satisfaction of SR1 and SR2: The merchant cannot know the cardholder's private key, and therefore cannot access the CAI that it passes through to the payment gateway.

The first trust assumption (TA1-1), that the domain *Users* contains only authorized individuals, is clearly risky, making the argument that SR2 is satisfied very problematic. There is no information available to justify the claim. The analyst should change the problem to eliminate the trust assumption and reduce the risk. A similar statement must be made about TA1-2, because nothing is said that allows the engineer to claim that the storage is secure. If the information can be read without supplying a key that is not stored on the machine, then the existence of viruses, spyware, and other programs/users make the trust assumption's claim ludicrous. Vulnerabilities to the threats still exist, and appropriate domains and phenomena must be added to close the vulnerabilities and satisfy the requirement.

Verifying TA1-3 is probably not necessary, assuming that the cryptographic software comes from a company that the requirements engineer believes has verified its applications. If the engineer is uncomfortable with this belief, then a domain representing the encryption software must be added to the problem, and then analyzed appropriately.

TA1-4 serves to limit the scope of the analysis, stating that nothing on the other side of the merchant can expose CAI to the merchant. Unfortunately, the SET 'processing flows' diagram (step 7) shows that the payment gateway can give the CAI back to the merchant. The trust assumption is invalid and must be removed.

Because TA1-1 was rejected, a *passphrase* has been added to verify that the user is authorized. The passphrase is used to encrypt the CAI and certificate storage. Use of the passphrase and encryption protects the CAI against both viruses and other users of the machine. Spyware that can capture the entry of the passphrase is still a problem, not further discussed in this paper.

Figure 8 presents the modified problem. The context has been expanded to include the payment gateway.

Thinking about the satisfaction argument using the new problem diagram exposes the need for the following trust assumptions:

- TA2-1—satisfaction of SR1 and SR2: Users will not expose the passphrase.
- TA2-2—satisfaction of SR2: The merchant implements the SET recommendations and securely stores the CAI. There is no practical way to bypass this security, regardless of storage medium (operational, backup, etc.)
- TA2-3—satisfaction of SR2: The merchant's employees authorized to see the CAI will not reveal it.
- TA2-4—satisfaction of SR2: The CAI never appears in the clear on the merchant's internal network.
- The same trust assumptions that apply to the merchant also apply to the payment gateway.

Figure 9 presents the solution along with the four trust assumptions. To reduce the complexity of the diagram, the phenomena and the trust assumptions applied to the payment gateway are not shown.

The risk presented by TA2-1, that the passphrase will remain confidential, may or may not be acceptable. For example, a French bank decided the risk was too high, and included a smartcard reader in its implementation. The user must both know the passphrase and insert the appropriate smartcard into the reader. This solution greatly reduced the risk presented by spyware.

The remaining trust assumptions are problematic. There is no practical way for a requirements engineer to examine every merchant and payment gateway company, so the assumptions must be accepted at face value.

The trust assumptions required to fulfill the security requirement might provoke a debate about whether a customer-side product based SET is worth constructing. Given that the CAI can be stored on the merchant's machine, the difference between a SET solution and the ubiquitous solution based on secure sockets layer (SSL)

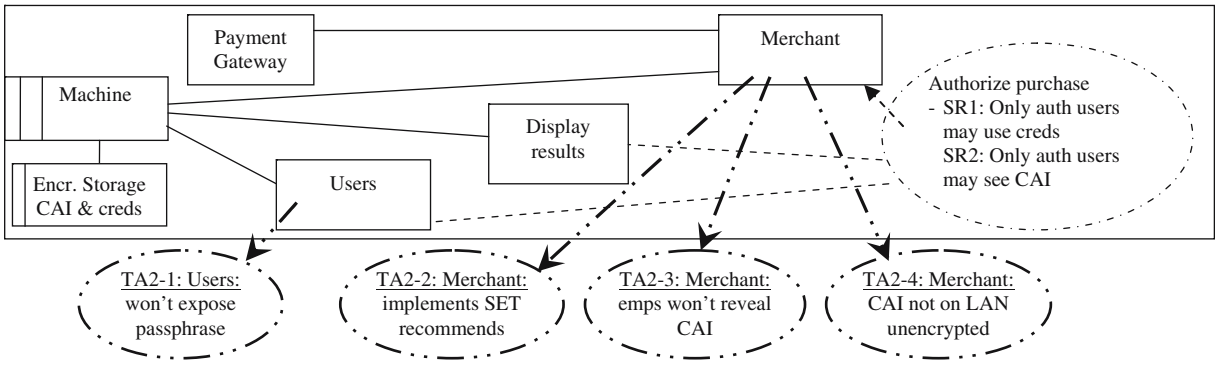

Fig. 9 Purchase problem (again)

is not large. Using SET, it is more difficult for a merchant to change an order, but a dishonest merchant would have no problem creating new non-SET orders charged to the customer. Dishonest merchants and employees could sell the account information. Hackers could steal it. There is nothing the engineer can do to mitigate the problems exposed by these trust assumptions. The customer/stakeholders must decide whether the risks are acceptable.

7 Related work

We are not aware of other work investigating the capture of a requirements engineer's trust assumptions about the domains that make up the solution to the problem. However, several groups are looking at security requirements, some of which include incorporation of trust between domains in the context.

7.1 The *i** framework

The *i** framework [23, 24] takes an 'actor, intention, goal' approach where security and trust relationships within the model are modeled as "softgoals": goals that have no precise measure for satisfaction. In [25], Liu et al. extended the framework to better support security and privacy by modeling the attacker as a malicious stakeholder. Countermeasures, which are themselves goals, are added to thwart the attacker.

The Liu et al. [25] work focuses on the *attacker* as the primary point of analysis. One finds vulnerabilities by asking what an attacker might wish to gain while playing some role, and then looking for ways that the attacker might achieve the wish. As *i** is focused on the actor, it is difficult to perform an asset + threat-centered analysis. In addition, it could be that the attacker does not wish to cause harm, but harm is caused. For these reason, we consider the *i** approach to be complementary to but not equivalent to the one discussed in [7] and in this paper.

*i** can be used to demonstrate the need for certain trust assumptions, specifically those that restrict which agents are permitted to play particular roles, and those that exclude an agent performing exhibiting undesired behavior. There is, however, no convenient way to insert these trust assumptions into the model without expanding the scope of the analysis. For example, one of the countermeasures proposed in [25] is "user authentication mechanism"; this is a leaf task. The mechanisms that support, provide, and rescind authentication credentials are not mentioned, but are clearly being trusted by the analyst to be correct. The only way to express this trust is to add the actors who administer authentication, a process that is highly recursive.

The Tropos project [16, 26–30] uses the *i** framework, adding wider lifecycle coverage. The methodology focuses on connecting agent-oriented architecture and development with *i**, extending the *i** model to describe the details of the agents' behaviors. A formal specification language was added in [27]. Security, represented as constraints on the interactions between two agents, was added in [16] and [30], extending the specification language to express these constraints and agent interaction dependencies. Architectural styles beyond agent-orientation are also discussed in [30]. Trust and trust delegation were added in [29], along with appropriate extensions to the specification language.

Although Tropos has significantly enhanced *i**'s ability to represent security constraints and dependencies, it does not extend *i**'s ability to represent trust assumptions made by the analyst about the world. The authorization example described above also applies to Tropos; one finds authorization constraints and subgoals in [30], but one cannot easily indicate that administration of the authorization information is trusted, beyond extending the goal structure to include analysis of credential administration. One reasonable position is that certain trust assumptions are embedded in the definitions and conditions of the formal modeling language (just as can be said for KAOS below), but these assumptions would be implicit and not justified.

Other work has extended *i** in related directions. Gans et al. [31] add distrust and inter-agent communi-

cation (“speech acts”). Actors in the system decide dynamically to trust or to distrust each other. Yu and Cysneiros have focused on privacy in [32], exploring how privacy requirements fit into an *i** model. Both papers are concerned with analyzing trust relations between actors/agents in the running system, as opposed to capturing the requirements engineer’s assumptions.

Because *i** and its derivatives do not model the real-world components forming the connections between the agents of the system as it will eventually be built, certain classes of trust assumptions are difficult to make explicit. The best examples relate to unexpected connections between domains, such as information passing on paper through a mailroom, people hearing through walls, and security of backup media. A problem frames representation can make these “properties of the world” explicit, which is one reason that we have chosen to use it.

As noted above, we believe that our approach complements an *i**-based analysis. In addition to being used to determine the functional goals and requirements, the *i** approaches are useful for finding security goals and countermeasure tasks oriented around agent interaction (e.g., as in [25]). Trust assumptions (but not context and problem diagrams) and satisfaction arguments could be used to examine the countermeasure tasks for hidden assumptions that may or may not be justified, and to justify limiting the analysis at the points chosen.

7.2 KAOS

KAOS [13, 33], a goal-oriented requirements engineering method, uses *obstacles* to analyze security and safety [34]. An obstacle to some goal “is a condition whose satisfaction may prevent the goal from being achieved” [35]. A recent addition is anti-goals, a refinement of obstacles, to discover and close vulnerabilities [35, 36]. One begins with a goal model for some system; the goal model includes a domain model expressed using temporal logic. Security goals for objects in the domain are enumerated using a catalog of general goals (e.g., confidentiality, integrity, etc.). One inverts these goals to express the goals of some attacker (anti-goals), and then looks for vulnerabilities in the original domain model that permit the anti-goals to be realized.

As in *i**, there are ways in KAOS to find and express some kinds of trust assumptions. One could argue that some *expectations*, terminal goals under the responsibility of non-software agents [35] (called *assumptions* in [13]), are expressions of trust assumptions, as the analyst is choosing to stop analysis at that point. Domain-specific axioms might also fall into the category of trust assumptions. For example, the *authorized* predicate described in [35] is clearly depending on knowing if an agent is an owner, a proxy, or a manager, but there is no expression of how it is known or managed. In any event, the assumptions are not explicit.

As is noted in [35], not all vulnerabilities must be eliminated, but instead may be mitigated or ignored. The choice varies with the context of the vulnerability—the level of harm being risked and the probability that the harm will occur. Using KAOS, one expresses security goals in terms of the vulnerability to be addressed, as opposed to the asset to be protected, losing information explaining the provenance of the goal (the context of the vulnerability). Goal refinement further distances the goal from its source. This distance creates difficulty when considering whether the cost of satisfying a security goal in a particular context is justified by the risk presented by the vulnerability in that context.

We believe our techniques are complementary to KAOS. Trust assumptions can describe behavior outside of the KAOS domain model. They can be used to limit the depth of a goal derivation graph. Operationalized security goals uncovered using the methods described in [35] can be added as constraints to the problem diagrams that represent the appropriate functional requirements, after which they must be accounted for by the satisfaction arguments, whether through modifications to the context or through addition of trust assumptions.

7.3 Other security and privacy work

He and Antón [37] are concentrating on privacy, working on mechanisms to assist trusting of privacy policies, for example on web sites. They propose a context-based access model. Context is determined using “purpose” (why is information being accessed), “conditions” (what conditions must be satisfied before access can be granted), and “obligations” (what actions must be taken before access can be granted). The framework, like *i**, describes run-time properties, not the requirements engineer’s assumptions about the domains forming the solution.

Security requirements have been added to SCR [38] and to the WinWin framework [39]. As with *i** and KAOS, one can locate some trust assumptions in both SCR and WinWin by looking for where the analyst stopped. The implicit decision to limit the context almost certainly has some number of trust assumptions behind it.

Alexander is looking at detecting vulnerabilities using misuse cases [40, 41], as are Sindre et al. [42]. McDermott uses ‘abuse cases’ [43, 44]. These techniques all assist with reasoning about security by postulating the existence of an attacker who attempts to exploit the system in a way that will cause harm. Sindre et al. [42] introduced the idea of *mis*use and *mis*actors into use cases to identify potential security flaws in a system. This work concentrated on simplicity, using the diagrams as a communications tool and saying that “misuse diagrams must only be seen as a support for eliciting threats”. Alexander extended the relations over those presented by Sindre et al., adding mitigation and restriction. McDermott et al. concentrated on exploring the details of an exploit, documenting the route and expertise needed to be successful. In all cases, an analyst’s trust assumptions are embedded in the diagrams and not made explicit; one can argue that the

very choices of the cases to analyze constitute trust assumptions. Although we believe that these techniques do not capture trust assumptions, they should be quite useful for testing validity of the satisfaction arguments built using trust assumptions.

Srivatanakul et al. [45] are combining use cases with risk analysis techniques taken from safety, specifically HAZOP. They extend the abuse case and misuse case work discussed above ([40–44]) by adding HAZOP 'guideword'-driven analysis of use cases to find potential abuses. One uses the guidewords to find *deviations* for the elements in a use case (e.g., actors, associations, event flow, pre- and post-conditions). These deviations represent potential violations of "security properties" of a system. If a security property is violated, the deviation represents a successful attack. One locates the vulnerabilities that were exploited, then takes appropriate steps to close or mitigate the vulnerabilities. The method, like misuse cases, abuse cases, and abuse frames [46], takes what might be considered a bottom-up approach; the methods locate vulnerabilities that lead to security requirements that, if satisfied, will ensure the closure the vulnerabilities. If no vulnerabilities are found, then the satisfaction argument has been bolstered. Although the technique as described employs use cases, we believe that a similar technique would work with the problem frames-based method we have used, adding another tool to help operationalize security goals into security requirements.

Some of the work in the aspect-oriented requirements engineering (AORE) community is related to identification of security requirements. Rashid et al. propose that ideas from aspect-oriented software development can be used when mapping non-functional requirements (NFRs) onto functional requirements [47, 48]. They start by identifying the NFRs that affect more than one functional requirement, determine what the effect of the overlap is, then model the composition of the requirements. In their work, security is treated identically to other NFRs. Their work is more general than the work presented in this paper. It focuses on managing the interplay and the results of composition of the requirements, not deriving requirements from the NFRs. Brito et al. [49] propose that non-functional requirements from an NFR catalog [15], be integrated with functional requirements using a composition process. The composition process connects security goals with functional requirements and permits specifying the priority of satisfaction arguments, but does not aid with the construction of these arguments. None of this work incorporates capture of the assumptions made by a requirements engineer when specifying a system.

7.4 Design rationale

Satisfaction arguments are clearly related to argumentation and design rationale. Our current position is that design rationale is principally concerned with capturing how one arrived at a decision, alternate decisions, or the parameters that went into making the decision [50]. For example, Buckingham Shum [51] focuses on how rationale (arguments) are visualized, especially in collaborative environments. Potts and Bruns [52], and later Burge and Brown [53], discuss capturing how decisions were made, which decisions were rejected or accepted, and the reasons behind these actions. Mylopoulos et al. [54] present a way to represent formally knowledge that was captured in some way, without focusing on the outcome of any decisions. Ramesh and Dhar [55] describe a system for "capturing history in the upstream part of the life cycle." Fischer et al. [56] suggest that the explicit process of argumentation can itself feed into and benefit design. Finkelstein and Fuks [57] suggest that the development of specifications by multiple stakeholders, who hold disparate views, may be achieved through an explicit dialogue that captures speech acts, such as assertions, questions, denials, challenges, etc. The representation of the dialogue is then a rationale for the specifications constructed.

To reiterate, our position is that the common element in all of the above work is the capture over time of the thoughts and reasons behind decisions. Whether the decisions satisfy the needs is not the primary question. We will be verifying this position in future work, looking at how our research fits in with other design rationale work, the representation of satisfaction arguments, and how one argues that trust assumptions are needed and sufficient. (For an example, see [58].)

8 Conclusions and future work

We have provided an approach for using trust assumptions when reasoning about the satisfaction of security requirements. The approach uses the strong distinction between system requirements and machine specifications found in problem frames, permitting the requirements engineer to choose how to conform to the requirements. The trust assumptions embedded in the solution inform requirements engineers, better enabling them to choose between alternate ways of satisfying the functional requirements while ensuring that vulnerabilities are removed or not created.

A prime area of future work is how our research fits in with argumentation and design rationale. See Sect. 7.4 for more discussion.

Another future focus will be a tighter coupling of trust assumptions and problem frames. When a larger problem is decomposed, the domains in the resulting problem diagrams are a projection of the context. The projection can combine domains into single entities, or it can split a domain into its component parts. Having such projections raises the question "to what, exactly, is the trust assumption connected?" The question is important because trust assumptions have impacts on the membership and phenomena of the projected domain, and we must determine how these impacts

affect other problems that reference any part of the projected domain. Finally, whether and how problem classes affect the derivation of trust assumptions is of significant interest.

Acknowledgements The financial support of the Royal Academy of Engineering and the Leverhulme Trust is gratefully acknowledged, as is the EU for supporting the E-LeGI project, number IST-002205. Thanks also go to Michael Jackson for many insights about problem frames and requirements. This paper is a revised and extended version of [59] and [60].

References

1. ISO/IEC: Information Technology—Security Techniques—Evaluation Criteria for IT Security. Part 1: Introduction and general model. International Standard 15408–1, ISO/IEC, Geneva Switzerland, 1 Dec 1999
2. Zave P (1997) Classification of research efforts in requirements engineering. Comput Survey 29(4):315–321
3. van Lamsweerde A (2000) Requirements engineering in the year 00: a research perspective. In: Proceedings of the 22nd international conference on software engineering (ICSE'00), 4–11 June 2000. IEEE Computer Society Press
4. Greenspan SJ, Mylopoulos J, Borgida A (1982) Capturing more world knowledge in the requirements specification. In: Proceedings of the 6th international conference on software engineering (ICSE'82), Tokyo, 13–16 September 1982, pp 225–234
5. Devanbu P, Stubblebine S (2000) Software engineering for security: a roadmap. In: Finkelstein A (ed) The future of software engineering. ACM Press, New York
6. Firesmith DG (2003) Common concepts underlying safety, security, and survivability engineering. Technical Report CMU/SEI-2003-TN-033, Software Engineering Institute, Carnegie Mellon University, Pittsburgh
7. Moffett JD, Haley CB, Nuseibeh B (2004) Core security requirements artefacts. Technical Report 2004/23, Department of Computing, The Open University, Milton Keynes
8. Jackson M (1995) Software requirements and specifications. Addison Wesley, Reading
9. Jackson M (2001) Problem frames. Addison Wesley, Reading
10. Viega J, Kohno T, Potter B (2001) Trust (and mistrust) in secure applications. Commun ACM 44(2):31–36
11. Thompson K (1984) Reflections on trusting trust. Commun ACM 27(8):761–763
12. Haley CB, Laney RC, Nuseibeh B (2004) Deriving security requirements from crosscutting threat descriptions. In: Proceedings of the 3rd international conference on aspect-oriented software development (AOSD'04), Lancaster, 22–26 March 2004. ACM Press, New York, pp 112–121
13. van Lamsweerde A (2001) Goal-oriented requirements engineering: a guided tour. In: Proceedings of the 5th IEEE international symposium on requirements engineering (RE'01), Toronto, 27–31 August 2001. IEEE Computer Society Press, pp 249–263
14. Zave P, Jackson M (1997) Four dark corners of requirements engineering. Trans Softw Eng Method 6(1):1–30
15. Chung L, Nixon B, Yu E, Mylopoulos J (2000) Non-functional requirements in software engineering. Kluwer, Dordrecht
16. Gani A, Manson G, Giorgini P, Mouratidis H (2003) Analysing security requirements of information systems using Tropos. In: Proceedings of the 5th international conference on enterprise information systems (ICEIS'03), Angers, 23–26 April 2003
17. Kotonya G, Sommerville I (1998) Requirements engineering: processes and techniques. Wiley, United Kingdom
18. Pfleeger CP, Pfleeger SL (2002) Security in computing. Prentice Hall, Englewood Cliffs
19. Grandison T, Sloman M (2003) Trust management tools for internet applications. In: Proceedings of the 1st international conference on trust management, vol 2692, Heraklion, Crete, 28–30 May 2003. Springer, Berlin Heidelberg New York
20. Secure Electronic Transaction LLC: SET Secure Electronic Transaction Specification Book 1: Business description, version 1.0. Purchase NY, 31 May 1997
21. Secure Electronic Transaction LLC: SET Secure Electronic Transaction Specification Book 2: Programmer's guide, version 1.0. Purchase NY, 31 May 1997
22. Secure Electronic Transaction LLC: SET Secure Electronic Transaction Specification Book 3: Formal protocol definition, version 1.0. Purchase NY, 31 May 1997
23. Yu E (1997) Towards modelling and reasoning support for early-phase requirements engineering. In: Proceedings of the 3rd IEEE international symposium on requirements engineering (RE'97), Annapolis, 6–10 January 1997, pp 226–235
24. Yu E, Liu L (2001) Modelling trust for system design using the *i** strategic actors framework. In: Falcone R, Singh MP, Tan YH (eds) Trust in cyber-societies, integrating the human and artificial perspectives Springer, Berlin Heidelberg New York, 15–16 October 2002, pp 175–194
25. Liu L, Yu E, Mylopoulos J (2003) Security and privacy requirements analysis within a social setting. In: Proceedings of the 11th IEEE international requirements engineering conference (RE'03), Monteray Bay, 8–12 September 2003
26. Castro J, Kolp M, Mylopoulos J (2001) A requirements-driven development methodology. In: Proceedings of the 13th conference on advanced information systems engineering (CAiSE'01), Interlaken, Switzerland, 4–8 June 2001, pp 108–123
27. Fuxman A, Pistore M, Mylopoulos J, Traverso P (2001) Model checking early requirements specifications in Tropos. In: Proceedings of the 5th IEEE international symposium on requirements engineering, Toronto, pp 174–181
28. Giorgini P, Massacci F, Mylopoulos J (2003) Requirement engineering meets security: a case study on modelling secure electronic transactions by VISA and Mastercard. In: Proceedings of the 22nd international conference on conceptual modeling, Chicago, 13–16 October 2003. Springer, Berlin Heidelberg New York, pp 263–276
29. Giorgini P, Massacci F, Mylopoulos J, Zannone N (2004) Requirements engineering meets trust management: model, method, and reasoning. In: Proceedings of the 2nd international conference on trust management, Oxford, 28 March–1 April 2004. Lecture notes in computer science. Springer, Berlin Heidelberg New York
30. Mouratidis H, Giorgini P, Manson G (2003) Integrating security and systems engineering: toward the modelling of secure information systems. In: Proceedings of the 15th conference on advanced information systems engineering (CAiSE'03), Klagenfurt/Velden, 6–10 June 2003. Springer, Berlin Heidelberg New York
31. Gans G, Jarke M, Kethers S, Lakemeyer G, Ellrich L, Funken C, Meister M (2001) Requirements modeling for organization networks: a (dis)trust-based approach. In: Proceedings of the 5th IEEE international symposium on requirements engineering (RE'01), 27–31 August 2001. IEEE Computer Society Press, Toronto, pp 154–165
32. Yu E, Cysneiros LM (2002) Designing for privacy and other competing requirements. In: Proceedings of the 2nd symposium on requirements engineering for information security (SREIS'02), Raleigh
33. Dardenne A, van Lamsweerde A, Fickas S (1993) Goal-directed requirements acquisition. Sci Comput Program 20(1–2):3–50
34. van Lamsweerde A, Letier E (2000) Handling obstacles in goal-oriented requirements engineering. Transact Softw Eng (IEEE) 26(10):978–1005
35. van Lamsweerde A (2004) Elaborating security requirements by construction of intentional anti-models. In: Proceedings of the 26th international conference on software engineering (ICSE'04), Edinburgh, 26–28 May 2004, pp 148–157

36. van Lamsweerde A, Brohez S, De Landtsheer R, Janssens D (2003) From system goals to intruder anti-goals: attack generation and resolution for security requirements engineering. In: Requirements for high assurance systems workshop (RHAS'03), 11th international requirements engineering conference (RE'03), Monterey, 8 September 2003
37. He Q, Antón AI (2003) A framework for modeling privacy requirements in role engineering. In: Proceedings of the 9th international workshop on requirements engineering: foundation for software quality, the 15th conference on advanced information systems engineering (CAiSE'03), Klagenfurt/Velden, 16 June 2003
38. Heitmeyer CL (2001) Applying 'practical' formal methods to the specification and analysis of security properties. In: Proceedings of the international workshop on information assurance in computer networks: methods, models, and architectures for network computer security (MMM ACNS 2001), vol 2052, St. Petersburg, 21–23 May 2001. Springer, Berlin Heidelberg New York, pp 84–89
39. In H, Boehm BW (2001) Using WinWin quality requirements management tools: a case study. Ann Softw Eng 11(1):141–174
40. Alexander I (2002) Initial industrial experience of misuse cases in trade-off analysis. In: Proceedings of the IEEE joint international conference on requirements engineering (RE'02), Essen, pp 61–68
41. Alexander I (2002) Modelling the interplay of conflicting goals with use and misuse cases. In: Proceedings of 8th international workshop on requirements engineering: foundation for software quality (REFSQ'02), Essen, 9–10 September 2002, pp 145–152
42. Sindre G, Opdahl AL (2000) Eliciting security requirements by misuse cases. In: Proceedings of the 37th international conference on technology of object-oriented languages and systems (TOOLS-Pacific'00), Sydney, 20–23 November 2000, pp 120–131
43. McDermott J (2001) Abuse-case-based assurance arguments. In: Proceedings of the 17th computer security applications conference (ACSAC'01), New Orleans, 10–14 December 2001. IEEE Computer Society Press, pp 366–374
44. McDermott J, Fox C (1999) Using abuse case models for security requirements analysis. In: Proceedings of the 15th computer security applications conference (ACSAC'99), Phoenix, 6–10 December 1999. IEEE Computer Society Press, pp 55–64
45. Srivatanakul T, Clark JA, Polack F (2004) Writing effective security abuse cases. Technical Report YCS-2004–375, Department of Computer Science, University of York, York, 11 May 2004
46. Lin L, Nuseibeh B, Ince D, Jackson M, Moffett J (2003) Introducing abuse frames for analyzing security requirements. In: Proceedings of the 11th IEEE international requirements engineering conference (RE'03), Monterey, 8–12 September 2003, pp 371–372
47. Rashid A, Moreira AMD, Araújo J (2003) Modularisation and composition of aspectual requirements. In: Proceedings of the 2nd international conference on aspect-oriented software development (AOSD'03), Boston, 17–21 March 2003. ACM Press, New York, pp 11–20
48. Rashid A, Sawyer P, Moreira AMD, Araújo J (2002) Early aspects: a model for aspect-oriented requirements engineering. In: Proceedings of the IEEE joint international conference on requirements engineering (RE'02), Essen, 9–13 September 2002, pp 199–202
49. Brito I, Moreira A (2004) Integrating the NFR framework in a RE model. Presented at Early aspects 2004: aspect-oriented requirements engineering and architecture design (AORE'04), with the 3rd international conference on aspect-oriented software development (AOSD'04), Lancaster University, UK
50. Lee J, Lai KY (1991) What's in design rationale? Hum Comput Interact Spec Issue Design Rationale 6(3–4):251–280
51. Backingham Shum SJ (2003) The roots of computer supported argument visualization. In: Kirschner PA, Buckingham Shum SJ, Carr CS (eds) Visualizing argumentation: software tools for collaborative, educational sense-making. Springer, London, pp 3–24
52. Potts C, Bruns G (1988) Recording the reasons for design decisions. In: Proceedings of the 10th international conference on software engineering (ICSE'88), Singapore. IEEE Computer Society, pp 418–427
53. Burge JE, Brown DC (2004) An integrated approach for software design checking using design rationale. In: Gero JS (ed) Proceedings of the 1st international conference on design computing and cognition. Kluwer, Cambridge, pp 557–576
54. Mylopoulos J, Borgida A, Jarke M, Koubarakis M (1990) Telos: representing knowledge about information systems. ACM Trans Inf Syst (TOIS) 8(4):325–362
55. Ramesh B, Dhar V (1992) Supporting systems development by capturing deliberations during requirements engineering. IEEE Trans Softw Eng 18(6):498–510
56. Fischer G, Lemke AC, McCall R, Morch A (1996) Making argumentation serve design. In: Moran T, Carrol J (Eds) Design rationale concepts, techniques, and use. Lawrence Erlbaum and Associates, Mahwah, pp 267–293
57. Finkelstein A, Fuks H (1989) Multiparty specification. In: Proceedings of the 5th international workshop on software specification and design, Pittsburgh, pp 185–195
58. Haley CB, Laney RC, Nuseibeh B (2005) Arguing security: validating security requirements using structured argumentation. Technical Report 2005/04, Department of Computing, The Open University, Milton Keynes, 21 March 2005
59. Haley CB, Laney RC, Moffett JD, Nuseibeh B (2004) The effect of trust assumptions on the elaboration of security requirements. In: Proceedings of the 12th international requirements engineering conference (RE'04), Kyoto, 6–10 September 2004. IEEE Computer Society Press, pp 102–111
60. Haley CB, Laney RC, Moffett JD, Nuseibeh B (2004) Picking battles: the impact of trust assumptions on the elaboration of security requirements. In: Proceedings of the 2nd international conference on trust management (iTrust'04), vol 2995, St Anne's College, Oxford, 29 March–1April 2004. Lecture notes in computer science. Springer, Berlin Heidelberg New York, pp 347–354