# Security and Privacy Requirements Analysis within a Social Setting

Lin Liu[1] Eric Yu[2] John Mylopoulos[1]

[1]Department of Computer Science, University of Toronto, Toronto, Canada, M5S 1A4
{liu, jm}@cs.toronto.edu
[2]Faculty of Information Studies, University of Toronto, Toronto, Canada, M5S 3G6
yu@fis.utoronto.ca

## Abstract

*Security issues for software systems ultimately concern relationships among social actors - stakeholders, system users, potential attackers - and the software acting on their behalf. This paper proposes a methodological framework for dealing with security and privacy requirements based on i*, an agent-oriented requirements modeling language. The framework supports a set of analysis techniques. In particular, attacker analysis helps identify potential system abusers and their malicious intents. Dependency vulnerability analysis helps detect vulnerabilities in terms of organizational relationships among stakeholders. Countermeasure analysis supports the dynamic decision-making process of defensive system players in addressing vulnerabilities and threats. Finally, access control analysis bridges the gap between security requirement models and security implementation models. The framework is illustrated with an example involving security and privacy concerns in the design of agent-based health information systems. In addition, we discuss model evaluation techniques, including qualitative goal model analysis and property verification techniques based on model checking.*

## 1. INTRODUCTION

Security, and privacy to a lesser extent, have been active research areas in computing for a long time. Methods and techniques have been developed to protect data, programs, and more recently networks, from attacks or other infringements through mechanisms such as access controls and firewalls. However, most techniques were developed for earlier generations of computing environments that were largely within a single, closed jurisdictional control -- such as a single enterprise with a well-defined boundary. The open Internet environment, together with new business and organizational practices, has increased the complexity of security and privacy considerations dramatically. In such a setting, a system could potentially be interacting and sharing information with a large number of other systems, often on ad hoc and dynamically negotiated configurations. Traditional models and techniques for characterizing and analyzing security and privacy are ill-equipped to deal with the much higher social complexity that is implicit in this new internet-based setting.

In this paper, we propose a methodological framework for analyzing security and privacy requirements based on the concept of strategic social actors. The framework offers a set of security requirements analysis facilities to help users, administrators and designers better understand the various threats and vulnerabilities they face, the countermeasures they can take, and how these can be combined to achieve the desired security and privacy results within the broader picture of system design and the business environment. Moreover, the analysis process is integrated into the usual requirements process, so that security and privacy are taken into account from the very start all at once.

This paper builds on our work on designing trust and role-based pattern analysis on security requirements. In [13, 23], we use role-based mechanism to study patterns of relationships such as trust relations, attacker-defender relations at various levels of abstraction. These patterns can be selectively applied and combined for analyzing specific system configurations later on. This idea has been integrated and extended in the attacker analysis discussed below.

Based on our previous works in agent-oriented software engineering [22] and non-functional requirements [4], we recognize that, as with other non-functional requirements, security and privacy goals must be identified and dealt with starting from the earliest stages of a software engineering process [24,23]. Security and privacy issues originate from human concerns and intents, and thus should be modeled through social concepts [24,13]. Social concepts are extended to cover relationships among software systems and components. Agent-based models enable richer descriptions and analysis techniques about internet-based environments, especially ones involving intelligent agents. Based on these models, knowledge-based decision support tools can help identify alternatives, detect conflicts and synergies, understand related implications and consequences, and through a systematic

process, eventually arrive at appropriate combinations of proven policies, procedures, devices, and mechanisms to achieve the desired levels of security and privacy.

The proposed security requirements analysis is illustrated with the example of designing software agents supporting patient-doctor interactions. Design of security and privacy in health care information systems is a challenging task due to the influences of complex factors in multiple dimensions. For instance, in the social dimension, there are both patient-physician and user-system trust relationships. There are also regulations and constraints along medical and financial dimensions. Besides, adding unfamiliar new technologies such as unified electronic medical records and software agents is bound to make the design task even more challenging, since problems that arise from these new dimensions need to be taken into account.

Designing for security and privacy amounts to answering questions such as: " who is likely to attack the system? By what means might a specific attacker attack the system? Whose privacy is at risk? How to defend the system from these threats? What are the side effects of adding particular countermeasures?" Yet, there is no systematic analysis technique through which one can go from answers to these questions to particular security and privacy solutions. Our proposal is intended to provide mechanisms that explicitly relate social concerns with the technologies and policies addressing these concerns.

Section 2 introduces the basic requirement analysis process supported by *i**. We base our example on the Guardian Angel (GA) project [20], a patient and physician supporting system using software agents. Section 3 discusses the extended modeling process and a set of security- and privacy-related analysis techniques. Section 4 describes two particular model evaluation techniques – goal-based evaluation and model property checking. Section 5 and section 6 discuss related work and summarize the results of the paper.

## 2. Domain Requirements Analysis with *i**

The solid lines and boxes on the left-hand side of Figure 1 indicate a series of *basic* domain requirements analysis steps.

***Actor identification*** answers the questions of " who is involved in the system?" In *i** [22], an *actor* is used to refer generically to any unit to which intentional dependencies can be ascribed. Figure 2 shows some actors in the GA domain. Actors may be further differentiated into roles, agents, and positions. A *role* is an abstract actor embodying expectations and responsibilities, e.g., Owner, Primary User, and Administrator of Patient Information, Guardian of Patient and Provider of Health Care Service. An *agent* is a concrete actor, human or machine, with specific capabilities and funcationalities, e.g., Abby Kaye, Dr. Anthony, Ms. Young, GA-PDA and GA-Hospital Module. An agent can play one or more roles. A set of roles packaged together to be assigned to an agent is called a *position*. In Figure 2, Patient is modeled as a position which bridges the multiple abstract roles it covers, and the real world agents occupying it. As a simplification, other examples in this paper omit the use of the position concept. Initially, human actors representing stakeholders in the domain are identified together with existing machine actors (step ① in Figure 1). As the analysis proceeds (step ⑤ in Figure 1), more actors are identified, including new system agents such as GA System, GA-PDA, GA-HomePC, and GA Hospital Module, when certain design choices have been made, and new functional entities are added.

***Goal/task identification*** answers the question of " what does the actor want to achieve?" (step ② in Figure1). As shown in Figure 3, answers to this question can be represented as goals capturing the high-level objectives of

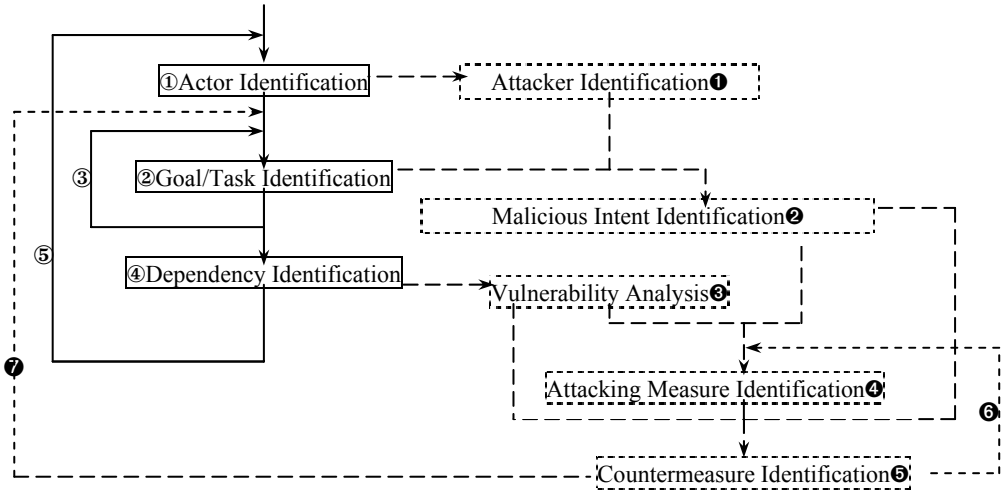


Figure 1. Requirements Elicitation Process with *i**

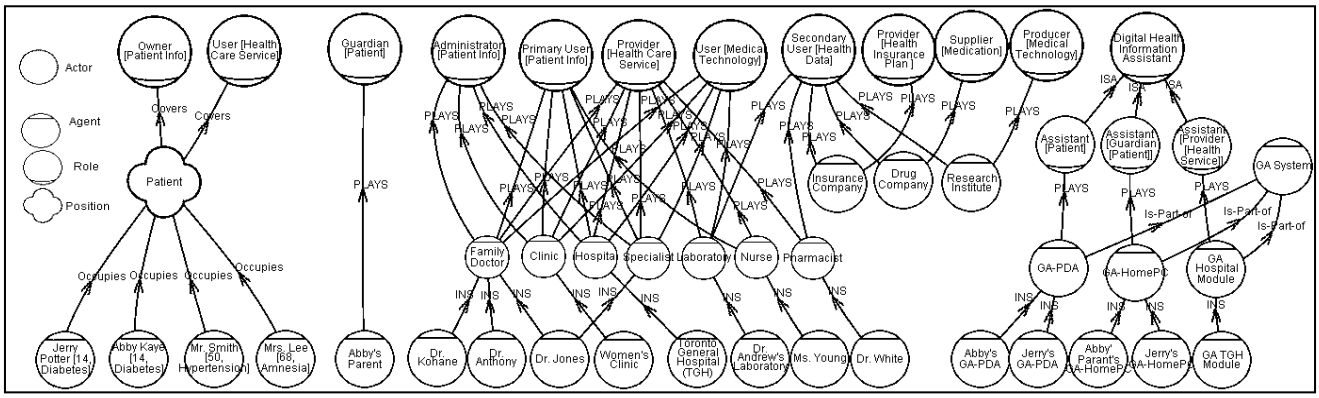


Figure 2. Actors (roles, agents and position) in the GA system

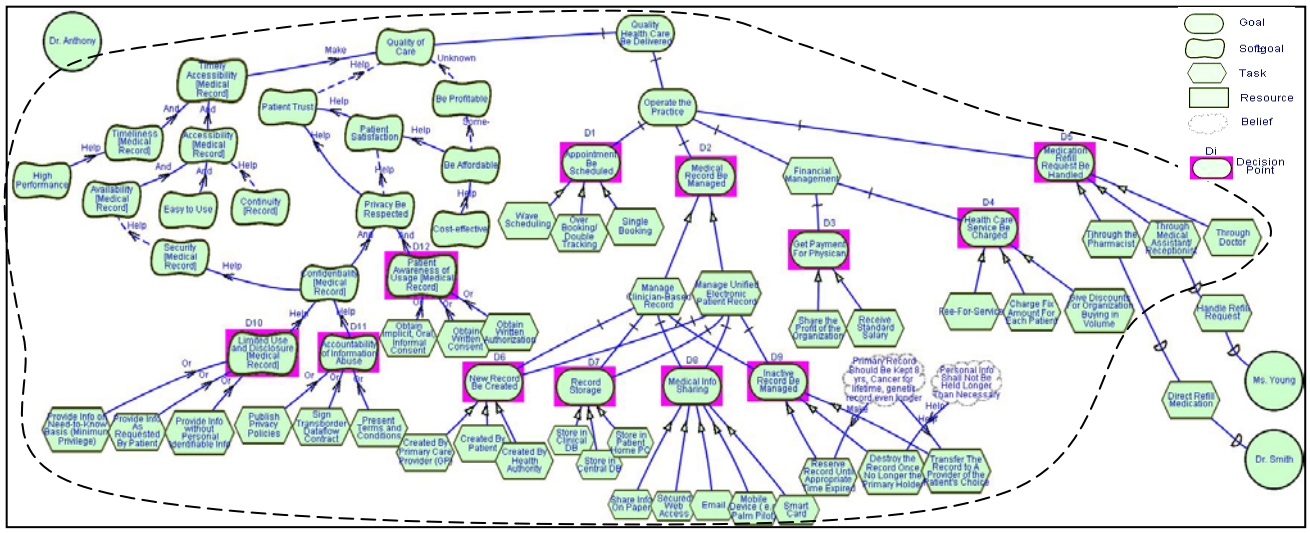


Figure 3. Goal/task elicitation in the space of alternatives for a physician opening a new practice (SR)

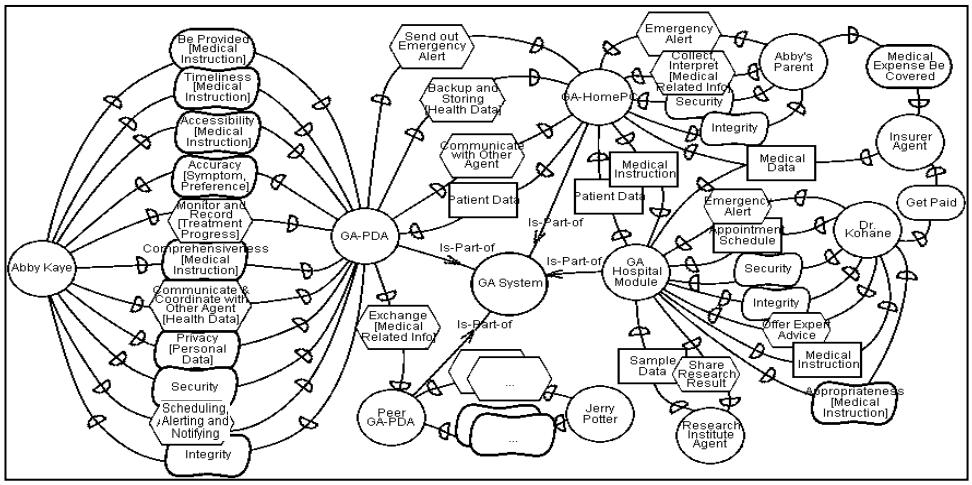


Figure 4. Dependency relationships in the GA system (SD)

agents. A goal may be " hard", referring to a function, e.g. Dr. Anthony wants Quality Health Care Be Delivered, or " soft", referring to a quality requirement, e.g. Timely Accessibility of Medical Record. *Task*s are used to represent the specific procedures to be performed by agents, e.g. Manage Clinician-based Record. A *resource* is a physical or informational entity, about which the main concern is whether it is available. A *belief* is used to represent a domain characteristic, a design assumption or an environmental condition.

A goal can be accomplished in different ways. For example, the goal Medical Record Be Managed can be achieved by performing the task Manage Clinician-Based Record or Manage Unified Electronic Record. The tasks are connected to the goal through means-ends links(⊢▷—). A goal is satisfied if any of its tasks is satisfied. A task may be detailed into subgoals, subtasks, resources and softgoals through *Decomposition* link (—┼). All subcomponents of a task must be satisfied in order to accomplish the task. Such goal models can represent the different alternatives for achieving a goal, elaborate the necessary components for carrying out a task, and evaluate the positive or negative contributions from tasks to softgoals. High-level abstract softgoals are reduced into lower-level, more specific softgoals or operationalized in terms of tasks through *contribution* link (→). The refinement of goals, tasks, and softgoals (step ③ in Figure 1) are considered to have reached an adequate level once all the necessary design decisions can be made based on the existing information in the model. The *i** model in Figure 3 are created by running through steps ②, ③, ④ in Figure 1 iteratively.

***Dependency relationship identification*** answers the question " how do the actors relate to each other?" In *i**, we focus on intentional relationships (e.g., one actor depends on another for a goal to be achieved) rather than on information exchanges or flows (e.g., what message an actor send to another). A strategic dependency (SD) model is a network of intentional dependencies (*dependency link*, —◊—), as shown in Figure 4. When the internal rationales of agents are made explicit (as in Figure 3), we call that a strategic rationale (SR) model. By analyzing the dependency network in an SD model, we can reason about opportunities and vulnerabilities.

The SD model in Figure 4 shows that Abby Kaye depends on GA-PDA to provide medical instruction (Be Provided [Medical Instruction]). This dependency is accompanied by expectations on Timeliness, Accessibility, and Comprehensiveness of the Medical Instruction. The model is generated by running steps ③, ④ and ⑤ in Figure 1 recursively. As explained above, by hiding the internal rationales of actors in an SR model, an SD model can be obtained. Thus, the goal, task, resource, softgoal dependencies presented in an SD model are not added arbitrarily, it always indicates a necessity of delegation relationship across the actor boundary.

Dependency types are used to differentiate the kinds of freedom allowed in a relationship. Be Provided [Medical Instruction], being modeled as a goal dependency, indicates that GA-PDA has full freedom to decide how to provide instruction to Abby Kaye. Scheduling, Alerting and Notifying, being a task dependency means that GA-PDA must follow a prescribed course of action. A resource dependency (e.g., Patient Data) means that the depended party (*dependee*) has to make it available to the depender.

In this paper, *i** models are shown graphically. Semantics and constraints of *i** are embedded in the *i** meta-framework described in *Telos*[15]. With the support of Telos, consistency checks between models, scalability management of large project, and various other knowledge-based reasoning techniques can be applied to *i** models.

The kind of analysis shown above answers questions such as " Who is involved in the system? What do they want? How can their expectations be fulfilled? And what are the inter-dependencies between them?". These answers initially provide a sketch of the social setting of the future system, and eventually result in a fairly elaborate behavioural model where certain design choices have already been made. However, another set of very important questions has yet to be answered, i.e., " What if things go wrong? What if the GA system does not behave as expected? How bad can things get? What prevention tactics can be considered?" These are some of the questions we want to answer in the security requirements analysis process.

## 3. Security Requirements Analysis with *i**

The dashed lines and boxes on the right hand side of Figure 1 indicate a series of security specific analysis steps. These steps are integrated into the basic domain requirements engineering process, such that threats from potential attackers are anticipated and countermeasures for system protection are sought and equipped wherever necessary. Each of the security related analysis steps (step ❶ to ❼) will be discussed in detail in the following subsections.

### 3.1 Attacker Analysis

Attacker analysis aims to identify potential system abusers and their malicious intents. The basic premise here is that all the actors are assumed " guilty until proven innocent". In other words, given the result of the basic *i** requirements modeling process, we now consider any one of the actors (roles, positions or agents) identified so far can be a potential attacker to the system or to other actors.

For example, we want to ask, " In what ways can a physician attack the system? How will he benefit from inappropriate information disclosure?"

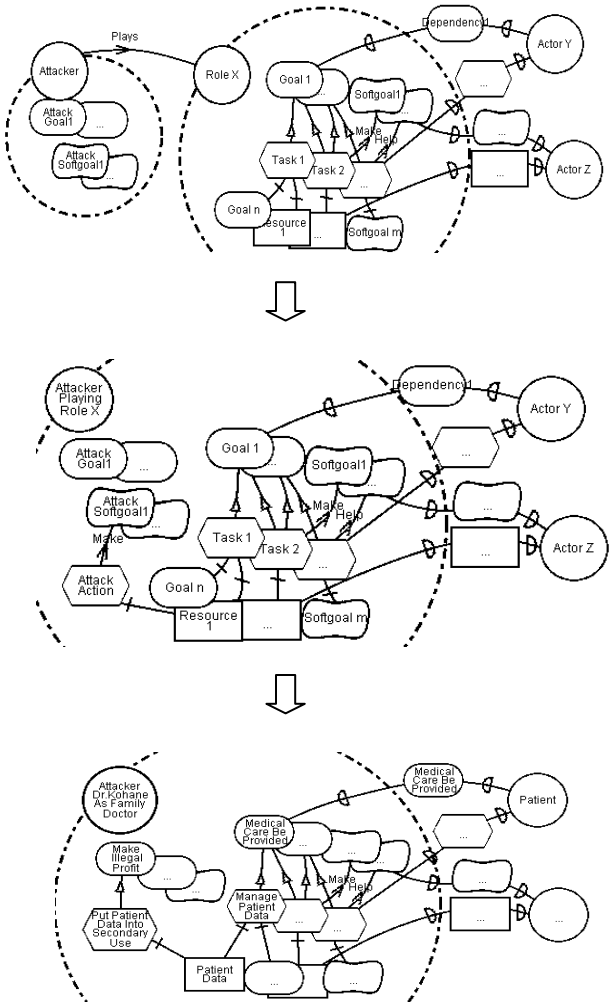


Figure 5. Attacker Analysis

In this analysis, each actor is considered in turn as an attacker. This attacker inherits the intentions, capabilities and social relationships of the corresponding legitimate actor (i.e., the internal goal hierarchy and external dependency relationships in the model). This may serve as a starting point of a forward direction security analysis (Step ❶ in Figure 1). A backward analysis starting from identifying possible malicious intents and business assets of value is also feasible here.

Proceeding to step ❷ of the process, for each attacker identified, we combine the capabilities and interests of the attacker with those of the legitimate actor (Figure 5). The analysis would reveal the commandeering of legitimate resources and capabilities for illicit use. For example, Dr. Kohane in playing the role of Family Doctor has access to certain patient data. While becoming an attacker (Attacker Dr. Kohane As Family Doctor), he will be able to Make Illegal Profit by Put Patient Data Into Secondary Use.

Applying the above reasoning to the *i** model in Figure 2, we may identify that potential attackers to the system are Patient Attacker, Patient Guardian Attacker, Care Provider Attacker, Business Associate (e.g., Insurance Company, Drug Company) Attacker and GA Software Agent Attacker. Here, we use the term attacker to refer to the source of any threat. Human attackers may attack deliberately, e.g., by committing insurance fraud, hiding malpractice evidence, and putting patient identifiable information into secondary use. An attack can also be accidental, e.g., accidental disclosure of embarrassing private information. Software agents can be threats as instruments of malicious human agents (e.g. they can be compromised through " hacking" or " sniffing") or simply through malfunctions, e.g., misunderstanding of user instructions, executing instructions improperly, perform tasks not intended by the user. In any case, software agents are considered as attackers to the system just the same as human attackers.

The attacker identification approach introduced above observes that all attackers are insider attackers. We set a system boundary, then exhaustively searches for possible attackers. In light of this, random attackers such as Internet hackers/crackers, or attackers breaking into a building can also be dealt within this framework by being represented as sharing the same territory as their victim. By conducting analysis on the infrastructure of the Internet, we may identify attackers by treating Internet resources as resources in *i** model. By conducting building security analysis, break-in attackers, or attackers sharing the same workspace can be identified. In [24], we have adopted an opposite assumption, i.e., assume there is a trusted perimeter for each agent, all the potential threats source within this trusted perimeter are ignored, only threats out of the perimeter will be protected.

### 3.2 Dependency Vulnerability Analysis

Dependency vulnerability analysis aims at identifying the vulnerable points in the dependency network (step ❸ in Figure 1). The basic idea is that dependency relationships bring vulnerabilities to the system and the depending actor (the *depender*). Potential attackers may exploit these vulnerabilities to actually attack the system, so that their malicious intents can be served. *i** dependency modeling allows a more specific vulnerability analysis because the potential failure of each dependency can be traced to a depender and to *its* dependers. The questions we want to answer here are " which dependency relationships are vulnerable to attack?", " What are the chain effects if one dependency link is compromised?"

Figure 6 shows some of the vulnerable points in the GA system. Dependency vulnerability analysis starts by substituting one of the actors in the basic dependency model with its corresponding attacker identified above, then referring to each *incoming* dependency link, it asks, " Is it possible that this actor, now as attacker, does something the depender does not want?" If the answer is " yes", a dependency attack link will be directed to that dependency. For example, attacker Insurer Agent may attack the GA Hospital Module by not Performing Insurance Transaction as expected, or by *hurt*ing its expectation on Privacy. Graphically, dependency attack links are represented with an arrow annotated by a link type. According to the different strength of the potential attack, the type of a dependency attack link can be *Break*, *Hurt*, *Some-*, or *Unknown*. Each dependency link associated with a dependency attack link indicates a vulnerable point of the future system.

The analysis of dependency vulnerabilities does not end with the identification of potential vulnerable points. We need to trace upstream in the dependency network, and see whether the attacked dependency relationship impacts other actors in the network. The model in Figure 6 shows that the Insurer Agent's attack eventually *hurt*s Abby Kaye's Privacy expectation through a dependency chain passing through GA-Hospital Module, GA-HomePC, and GA-PDA, all of which are parts of the GA System. To conduct this analysis, we also need means-ends and task structure information in the SR model. Another example given in Figure 6 is that, when Jerry Potter is playing attacker, he may *break* GA-PDA's expectation on his Integrity, (e.g., by flooding messages), which will hurt other agents in the GA system. This analysis process can be repeated for each role playing attacker in the *i** model, so that an exhausive search can be conducted to identify the vulnerabilities in the entire dependency network.

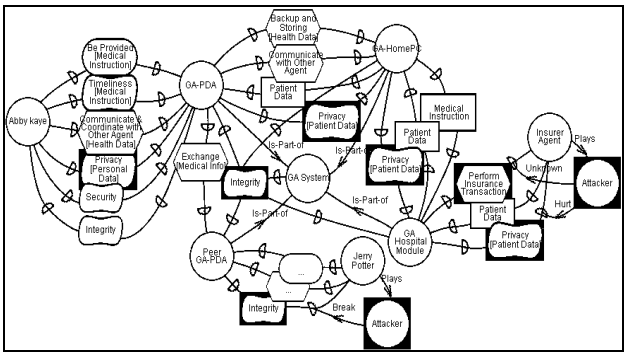

Figure 6. Dependency Vunerability Analysis

### 3.3 Countermeasure Analysis

During countermeasure analysis, system designers make decisions on how to protect security and privacy from potential attackers and vulnerabilities. This type of analysis covers general types of attacks, and formulates solutions by selectively applying, combining, or instantiating prototypical solutions to address the specific needs of various stakeholders. The general types of attacks and the prototypical solutions can be retrieved from a taxonomy or knowledge repository such as the ones in [2, 4].

Necessary factors for the success of an attack are attacker's motivations, vulnerabilities of the system, and attacker's capabilities to carry out the attack. Thus, to counteract a hypothetical attack, we seek measures that sufficiently negate these factors. Based on the above analysis, we already understand the attackers' possible malicious intentions and system vulnerabilities. Proceeding to step ❹, we now focus on how an attacker may attack the vulnerable points identified above by exploring the attacker's capacities.

As shown in Figure 5, it is an attacker's normal roles that bring him/her the capabilities to perform system tasks and to access system resources. By abusing such legal capacities or exploiting certain design vulnerabilities, an attacker may achieve its undesirable objectives. Although software attackers pose different kinds of threats compared to human attackers, who actively perform tasks they are not supposed to, software agents may do this on behalf of humans manipulating them.

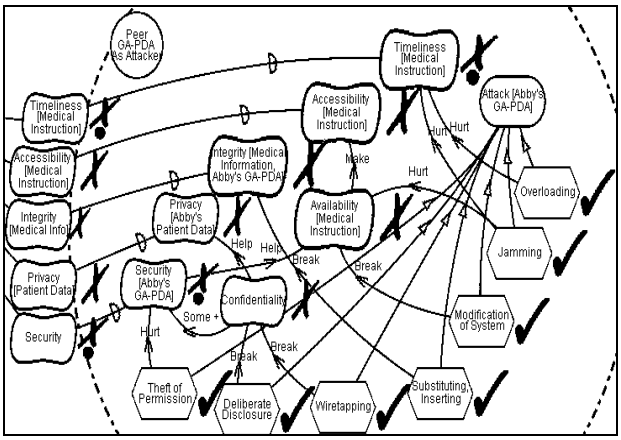

Figure 7. Attacks and Threats Identification

In Figure 7, Abby's GA-PDA depends on Peer GA-PDA to provide medical information and transmitting doctor's instructions. The impacts of a Peer GA-PDA As Attacker's various attacks are explored by doing softgoal refinement and evaluation, e.g., softgoal Privacy is refined to sub-softgoal Confidentiality, while Accessibility is refined to Availability ([4] provides more detailed refinements of softgoals). Their impacts to the dependencies are evaluated with a qualitative labeling algorithm [4] in the same way that goals in the basic *i** requirements models are addressed. The labels are defined as follows: *Satisficed*

(✓), *Weakly Satisficed* (✓), *Conflict/irresolvable* (✓), *Undecided* (?), *Weakly Denied* (✗), *Denied* (✗). When certain types of attacks are identified: Theft of Permission, Deliberate Disclosure, Wiretapping, Substituting & Inserting, Modification of System, Jamming, or Overloading, dependencies on Security, Privacy, correctness of Information, Accessibility, and Timeliness are compromised.

In Figure 8, the defender's countermeasures to these threats are sought by doing means-ends (" *how*" and " *how else*") analysis (step ❺ in the security analysis procedure). Prevention and protections such as User Authentication Mechanism, Requiring User Authorization for Information Passing, Transmitting Info in Encrypted Format, Auto-Recovery Mechanism, Using High Efficiency Network and Daily Refreshing of Medical Instruction are added into the system design. When each actor is considered a system defender, the overall evaluation results of security, privacy, timeliness, accessibility, and correctness of medical instruction will be changed. Here, the hypothetical threats are represented as beliefs, since their existence is based on the designer's assumption. By evaluating the effects of the countermeasures to the threats, we are able to decide if the impacts of the threats are reduced to an acceptable level. The countermeasure analysis process will iterate until a satisfying solution is found.

As shown in Figure 1, countermeasure analysis is an iterative process. Adding protective measures may bring new vulnerabilities to the system, so a new round of vulnerability analysis and countermeasure analysis will be triggered (step ❻).

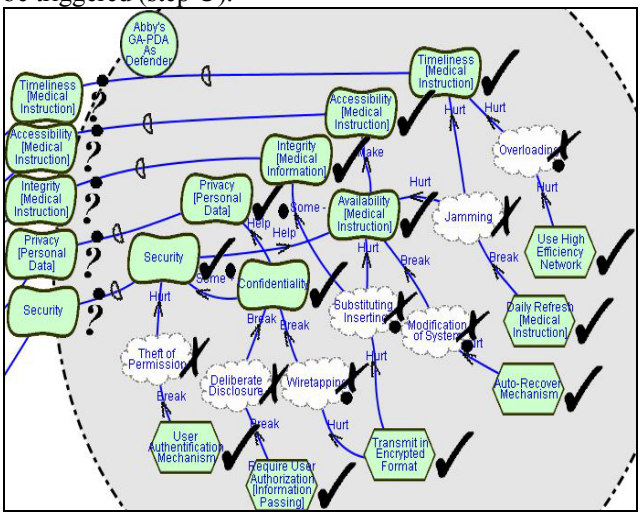

Figure 8. Countermeasure Analysis

### 3.4 Access Control Analysis

Access control analysis uses *i** models to refine a proposed solution and bring it closer to a system design. The role-based requirements analysis with *i** fits naturally to the role-based access control methodology in software design [16], and makes the transition from the former to the latter a smooth one. The basic idea for the access control analysis in *i** is that actor skills or capacities are encapsulated into abstract roles. In *i** model, skills and capacities are represented as tasks within the actor's boundary. Due to the differences in the tasks actors may perform, they will be assigned different access rights to the necessary resources accordingly. Graphically, access rights to data objects are represented as resources, whose name are defined as " DataObjectName[Access Privilege1, ...,n]". A more detailed model would treat an access right as a resource dependency on the role that grants that right.

The *i** model in Figure 9 shows that, Dr. Jones plays two roles: Family Doctor and Specialist [Amnesia]. He inherits the different capacities (skills) and access rights of the two roles. As a Family Doctor, he has access to Mr. Smith and Jerry Potter's full medical record. As a Specialist for Amnesia, he has access to Mrs. Lee's particular medical record: Amnesia Record [Patient, [Open, Read, Append, Transfer]].

An actor can play multiple roles. Within each role, there can be multiple ways for achieving a given goal. Each of these alternatives is composed of a different set of tasks that will also lead to different access privileges. These privileges need to be checked against principles such as Least Privilege and Separation of Duties discussed in section 4.2. For example, being the only Family Doctor of Jerry Potter, Dr. Jones have total access (open, read, append, transfer) to the complete medical record of Jerry Potter, while he has only read access to the old medical record of Mr. Smith created by his previous physician Dr. Anthony.

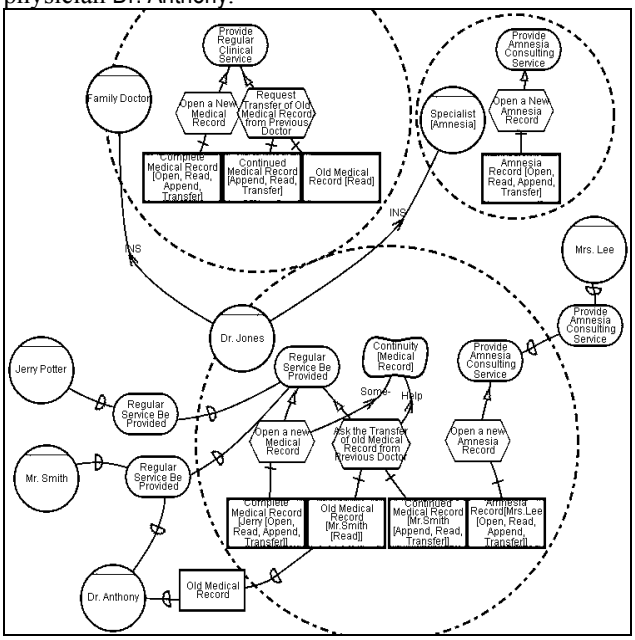

Figure 9. Access Control Analysis

## 4. Model Evaluation Techniques

### 4.1 Goal-Based Evaluation

To identify the best design solutions, goal-reasoning techniques such as qualitative goal labeling algorithms [4, 9, 10] can be used. Quantitative techniques, such as probability or other quantitative measures, can also be used [8]. With the help of *i** model, we are able to explore a space of design alternatives of considerable size. If there are *m* decision points (goals/softgoals with black rectangle shadow) and average *n* options at each point, there will be about $n^m$ alternatives to be chosen from. The model in Figure 3 shows 12 decision points, and there are 2-5 options at each decision point, which means that there are around $3^{12}$ alternative ways for a physician to operate his practice. Considering the presence of some external domain constraints, not all of these alternatives are workable. An example domain constraint is that, if a physician chooses to Manage Clinician Based Record, then it is not possible to Create (Record) By Patient. When there is a large space of alternatives to choose from, system designers will greatly appreciate automated support such as an approximate ranking according to some criteria.

The ranking of design alternatives is determined by the contributions to the softgoals of concern. When ranking design alternatives, various criteria can be adopted. We can then either rank alternatives according to their overall contributions to all softgoals, or rank according to user's specific preferences. Table 1. shows the labeling results of a qualitative evaluation. The contribution links are not fully presented in Figure 3 for the readability of the graph and limitation of space.

Below we present the two highest ranked combinations of options according to a ranking criteria as follows: Quality of Care > Easy to Use > Privacy > Security.

1. Manage Unified Electronic Patient Record / Created By Patient / Store in Patient Home PC / Secured Web Access / Reserve Record Until Appropriate Time Expired
2. Manage Clinician-based Record / Created By Clinician / Store in Clinical DB / Secured Web Access / Transfer The Record to A Provider of the Patient's Choice

The alternatives could also involve different inter-actor dependency relationships. Hence, goal-based evaluation may involve multiple actors and their dependencies. For example, when an unenforceable dependency relationship arises, the depender actor becomes more vulnerable to attacks. Thus, such alternatives should receive lower ranks, and when it has to be part of the design, countermeasures balancing the relationship need to be sought. Evaluation of dependency relationships are discussed in [24, 25].

Table 1. Labeling results of qualitative evaluation

| Decision Points | Alternatives | Labeling Result | | | |
|---|---|---|---|---|---|
| | | Quality of Care | Easy To Use | Privacy | Security |
| D2: Goal: Medical Record Be Managed | Manage Clinician-based Record | ✓ | ? | ? | ? |
| | Manage Unified Electronic Patient Record | ✓✓ | ? | ? | ? |
| D6: Goal: New Record Be Created | Created By Primary Care Provider | ? | ? | ? | ? |
| | Created By Patient | ✓✓ | ? | ✓ | ? |
| | Created by Health Authority | ✓ | ? | ✓ | ? |
| D4: Goal: Health Care Service Be Charged | Fee-For-Service | ✓ | ? | ? | ? |
| | Charge Fix Amount For Each Patient | ✓✓ | ? | ? | ? |
| | Give Discount to Organization Buying in Volume | ✓✓ | ? | ? | ? |
| D7: Goal: Record Storage | Store in Clinical DB | ✓ | ? | ✓✓ | ? |
| | Store in Central DB | ✓✓ | ? | ✓ | ✓ |
| | Store in Patient Home PC | ✗ | ? | ✓✓ | ✓✓ |
| D8: Goal: Medical Info Sharing | Share Info on Paper | ✗ | ✓ | ? | ✓ |
| | Secured Web Access | ✓ | ✓ | ? | ✓✓ |
| | Email | ✓ | ✓ | ? | ✓ |
| | Mobile Device (e.g. Palm Pilot) | ✓ | ✓ | ? | ✓✓ |
| | Smart Card | ✓✓ | ✓✓ | ? | ✓✓ |
| D9: Goal: Inactive Record Be Managed | Reserve Record Until Appropriate Time Expired | ✓✓ | ? | ✓ | ? |
| | Destroy the Record Once No Longer the Primary Care Provider | ✗ | ? | ✓ | ? |
| | Transfer a Record to A Provider of the Patient's Choice | ✓ | ? | ✓✓ | ? |
| D12: Softgoal: Patient Awareness of Usage [Medical Record] | Obtain Implicit, Oral, Informal Consent | ? | ? | ✓ | ? |
| | Obtain Written Consent | ? | ? | ✓ | ? |
| | Obtain Written Authorization | ? | ? | ✓ | ? |
| D11: Softgoal: Accountability of Information Abuse | Publish Privacy Policies | ? | ? | ✓ | ? |
| | Sign Transborder Dataflow Contract | ? | ? | ✓ | ? |
| | Present Terms and Conditions | ? | ? | ✓ | ? |
| D10: Softgoal: Limited Use and Disclosure [Medical Record] | Provide Info on Need-to-Know Basis | ? | ? | ✓ | ? |
| | Provide Info As Requested by Patient | ? | ? | ✓ | ? |
| | Provide Info without Personal Identifiable Info | ? | ? | ✓ | ? |

### 4.2 Access Permission Verification

It is often necessary to verify whether some expected properties are satisfied by the requirement model we have obtained in the various stages of the requirements process. Formal analysis techniques, such as model checking, are proven to have great potential in

requirement model specification and property verification [7]. Here we demonstrate how a lightweight object modeling notation Alloy [11] can be used for the specification of an access control requirements model respecting certain security and privacy relevant properties.

Thanks to the simplicity of the Alloy language and its easy-to-use Alloy constraint analyzer, an *i** model can be easily rewritten into an Alloy specification. Following is a simplified Alloy translation of the i* model framework. Naturally, actors, goals, tasks, resources, dependency relationships can be considered as sets of objects (**signature, sig** ) with various attributes.

```
module istarRBAC

sig Agent {                 /* Agent class declaration */
  plays_role: Role,         /* "plays" link in /* */
  assigned: Permission      /* resource access permissions in the
                               system */
}
sig Role {                  /* Role class declaration */
  has_objective: Goal       /* goals within the boundary of role */}

sig Goal {                  /* Goal class declaration */
  achieved_by_task: Task    /* means ends link in I*  */}

sig Task {
need_access: Permission     /* task-resource link in I*  */}

sig Permission {            /* Resource class declarations that
                               represents access permissions */
  exclusive: Permission     /*mutual exclusive access permissions*/
}{                          /* mutual exclusion does not apply to itself */
  all p: Permission | not p::exclusive = p
}
```

Some global constraints and properties that are hard to represent with *i** can be described as facts or functions in Alloy. In the following, we specify two well-known and good-to-have properties for a security and privacy protected system.

- *Least privilege (need-to-know principle)* [16]

Least privilege requires that only those permissions required for the tasks conducted by roles that an actor is playing are assigned to the actor.

```
fact LeastPrivilege {
  all pm: Permission | all a: Agent |
   pm in a::assigned =>
   pm in
        a::plays_role::has_objective::achieved_by
_task::need_access
} /* For all agent a, all the permission pm assigned to a is needed by some
task to achieve some goal of some role that a is playing. */
```

- *Separation of duties (mutual exclusive roles principle)* [16]

Separation of duties ensures that a sensitive or critical process cannot be performed by a single actor, mutually exclusive roles must be invoked. It is one of the main mechanisms to control fraud and error in the context of automated systems.

```
fact SeparationOfDuties {
 all a: Agent | all r1: Role |
 all p1:Permission | some p2: Permission |
 some r2: Role |  r1 in a::plays_role
 && p1 in
r1::has_objective::achieved_by_task::need_access
 && p2 in p1::exclusive
 && p2 in
r2::has_objective::achieved_by_task::need_access
 => not  r2 in a::plays_role
}
/* For all agent a, if a plays role r1 which need permission p1, then a cannot
play another role r2, which needs p2, an exclusive permission of p1. */
```

By analyzing these properties on the requirements models with the Alloy analyzer, we were able to verify whether the defined access control models such as the one in Figure 9 respect these properties. With the instance editor in Alloy Analyzer, we may build an instance of the Alloy model. The model shown in Figure 10 corresponds to the *i** model in Figure 9, which is proved to be a valid solution. By defining the negation of the expected principles, we may obtain counter-examples of the model (e.g., an instance in which the agent is assigned permissions that are not needed for the agents tasks). By analyzing the counterexamples, we can trace back to the *i** model, see why the principles are violated by the instance. Then we can decide whether the problem originates from trivial modeling mistakes, or from conflicts between domain constraints and the general principle in question. The proposed model checking technique is generally applicable for the verification of requirement models. Since the size of the model is bounded by the scheme used, and it is not always necessary to generate the whole model when checking certain property, the model checking effort is tractable.

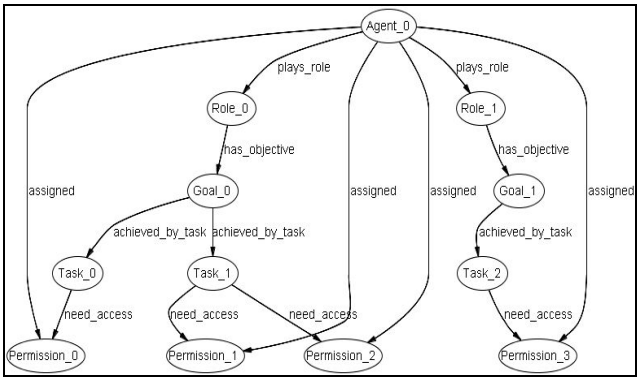


Figure 10. Representing *i** model in Alloy

## 5. Related Work

Security and privacy-related requirements engineering is a cross-disciplinary effort, with synergistic theories,

methods and techniques from several research areas, including requirements engineering, information and network security, policy development, and business process and organization management.

In the requirements engineering literature, [3, 23] treat security and privacy requirements together with other competing functional and non-functional requirements from the early requirements stage till concrete design choices are made. Our work adopts a similar viewpoint. The major difference with this earlier work is that, in our case, goal-oriented and agent-oriented analyses are conducted together. In [4] and [2], general catalogues/taxonomies for security and privacy goals are established, along with operationalization methods. These can serve as a general knowledge repository for a knowledge-based goal refinement/assessment process. In [21], a goal-oriented obstacle analysis method is used to capture exceptional requirements of a system, which can be used to model and analyze security requirements as well. Misuse cases [19] and abuse cases [14] demonstrate how negative scenarios can used to elicit and analyze security-related requirements. In [5], prospective visions on security requirements engineering of multiple dimensions are discussed. Our work is generally compatible with these visions. Generally speaking, the *i** approach encourages and facilitates the analysis of security and privacy-related issues within the broader social and organizational context of the relevant actors. In [12], past lessons has been studied intensively to demonstrate the importance of the tight coupling of safety engineering efforts with system design practice. Models that underlie particular approaches to safety problems are described. Our work targets at security problems in the new Internet-based systems based on similar observations, with a different set of modeling concepts.

The proposed approach is complementary to and benefits from the various theories and techniques currently being developed for information security protection. Traditional security engineering models and techniques such as: access control models [16], security policy models for specific application domain [1], authentication techniques [6], and trust evaluation and management mechanisms [17] are natural operationalizations or solutions for the security and privacy goals in the *i** requirements model.

Other relevant work effectively bridges the gap between information security and requirements engineering using fault trees [9], attack trees/threat trees [18] or other threat models to derive security requirements and develop security assured systems. These approaches are effective measures for security analysis of existing systems or systems in detailed design stage. The framework in this paper can be integrated with these mechanisms to deal with security at various abstraction levels. For example, we may use *i** models in a first-pass coarse-grained analysis when goals have not yet been operationalized. Once process-oriented operational details are obtained, other techniques may come into play.

## 6. Discussion

Security and privacy issues are becoming major concerns in the design of software systems. As new technologies and business models come into use in different socio-technical contexts, security and privacy issues becomes even more complex. Therefore, tools and methodologies providing systematic guidance and support to the design process are much needed.

This paper proposes a methodological framework for dealing with security and privacy requirements within an agent-oriented modeling framework. The main objective of the work is to define a set of security and privacy-specific analysis mechanisms and integrate them into the usual requirements engineering process, so that security and privacy requirements can be addressed together with other functional and non-functional concerns early on. The concepts provided by the *i** language make it possible to analyze security issues within their (natural) social context, leading to a systematic way of deriving vulnerabilities and threats. Moreover, the combined use of goals and agents in *i** models makes it possible to conduct different countermeasure analyses, such as counteracting malicious intentions, addressing vulnerabilities, and defending against attacking techniques.

The proposed methodological framework can be used for a top-down security requirements analysis process, or a bottom-up process that helps assess existing designs. The qualitative goal-based evaluation techniques facilitate trade-off analysis of security and other competing quality requirements, such as cost and performance. Model checking techniques can be applied at various stages of the requirements process, so that desired properties can be checked as the requirements models unfold.

We are currently refining the proposed methodology and are preparing to use it in a real-life case study, such that the scalability and tractability of the techniques can be further studied. A next step of this work is to develop a series of formalisms based on the current methodology, and tools supporting the kinds of reasoning on security.

## References:


[1] Anderson, R. A Security Policy Model for Clinical Information Systems. IEEE Symposium on Security and Privacy, Los Alamitos, 1996. pp.30-43.

[2] Anton,A.I., Earp, J.B., Reese, A. Analyzing Website Privacy Requirements Using a Privacy Goal Taxonomy. IEEE Joint International Requirements Engineering Conference (RE'02). Essen, Germany, September 9-13, 2002.pp.23-31.

[3] Chung, L. Dealing with Security Requirements During the Development of Information Systems. Proc. CAiSE'93, 5th Int. Conf. Advanced Information Systems Engineering, Collette Rolland, Francois Bodart, Corine Cauvet (Eds.). Springer, 1993. pp.234-251

[4] Chung, L., Nixon, B. A., Yu, E. and Mylopoulos, J. Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers, 2000.

[5] Crook, R., Ince, D., Lin, L., Nuseibeh, B. Security Requirements Engineering: When Anti-requirements Hit the Fan. IEEE Joint International Requirements Engineering Conference (RE'02). Essen, Germany, September 9-13, 2002.pp.203-205.

[6] Devanbu, P., Gertz, M., Martel, C., Stubblebine, S.G. Authentic Data publication over the Internet. Journal of Computer Security. Also available at: http://sirius.cs.ucdavis.edu/publications/jcs1.pdf.

[7] Fuxman, A., Pistore, M., Mylopoulos, J., and Traverso, P. Model Checking Early Requirements Specifications in Tropos, 5th IEEE Int. Symposium on Requirements Engineering (RE'01), Toronto, August 2001. pp.174-181.

[8] Giorgini, P., Mylopoulos, J., Nicchiarelli, E., Sebastiani, R.. Reasoning with Goal Models. 21th International Conference on Conceptual Modeling (ER-2002). Tampere, Finland, October 167-181, 2002. LNCS 2503 Springer Verlag. pp. 232-246.

[9] Helmer, G., Wong, J., Slagell, M., Honavar, V. Miller, L., Lutz, R. A Software Fault Tree Approach to Requirements Analysis of an Intrusion Detection System. Anton, A. eds. Special Issue on Requirements Engineering for Information Security. Loucopoulos, P., Mylopoulos, J. eds. Requirements Engineering. Vol. 7, N0. 4, 2002. pp. 177-220.

[10] Hui, B., Liaskos, S., Mylopoulos, J. Requirements Analysis for Customizable Software: A Goal-Skills-Preferences Framework. The 11th IEEE Int. Requirements Engineering Conference (RE'03). Monterey Bay, California USA, Sept. 8-12, 2003.

[11] Jackson, D. Alloy: A Lightweight Object Modeling Notation. ACM Transactions on Software Engineering and Methodology (TOSEM), Vol. 11, Issue 2, April 2002. pp. 256-290.

[12] Leveson, N. Safeware: System Safety and Computers. Addison-Wesley Publishing Company. 1995.

[13] Liu, L., Yu, E., Mylopoulos, J. Analyzing Security Requirements as Relationships Among Strategic Actors. 2nd Symposium on Requirements Engineering for Information Security (SREIS'02). Raleigh, North Carolina, October 16, 2002.

[14] McDermott, J., Fox, C. Using Abuse Case Models for Security Requirements Analysis. Proceedings 15th IEEE Annual Computer Security Applications Conference. 1999.

[15] Mylopoulos, J., Borgida, A., Jarke, M. and Koubarakis, M. Telos: Representing Knowledge About Information Systems. ACM Transactions on Information Systems 8(4), October 1990.

[16] Sandhu, R. S., Coyne, E.J., Feinstein, H.L., Youman, C.E. Role-Based Access Control Models. IEEE Computer, Vol.29, No.2, Feburary 1996. pp. 38-47.

[17] Sandhu, R. eds. Special Issue: The Technology of Trust. IEEE Internet Computing, November/December 2002. pp. 28-53.

[18] Schneier, B. Attack Trees Modeling Security Threats. Dr. Dobb's Journal, December 1999. Also available at: http://www.counterpane.com/attacktrees-ddj-ft.html.

[19] Sindre, G., Opdahl, A.L. Templates for Misuse Case Description. Proceedings of the 7th International Workshop on Requirements Engineering, Foundation for Software Quality (REFSQ'2001), Switzerland, June 4-5, 2001.

[20] Szolovits, P., Doyle, J., Long, W.J. Guardian Angel: Patient-Centered Health Information Systems. Technical Report MIT/LCS/TR-604. Available at: http://www.ga.org/ga.

[21] van Lamsweerde, A. and Letier, E. Handling Obstacles in Goal-oriented Requirements Engineering. IEEE Transactions on Software Engineering, 26(10). 2000.

[22] Yu, E. Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering. Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97) Jan. 6-8, 1997, Washington D.C., USA. pp.226-235.

[23] Yu, E., Cysneiros, L. Designing for Privacy and Other Competing Requirements. 2nd Symposium on Requirements Engineering for Information Security (SREIS'02). Raleigh, North Carolina, October 16, 2002.

[24] Yu, E. and Liu, L. Modeling Trust for System Design Using the *i** Strategic Actors Framework. In: Trust in Cyber-Societies - Integrating the Human and Artificial Perspectives. R. Falcone, M. Singh, Y.H. Tan, eds. LNAI-2246. Springer,2001. pp.175-194.

[25] Yu, E., Liu, L., Li, Y. Modeling Strategic Actor Relationships to Support Intellectual Property Management. 20th International Conference on Conceptual Modeling (ER-2001). Yokohama, Japan, November 27-30, 2001. LNCS 2224 Spring Verlag. pp. 164-178.