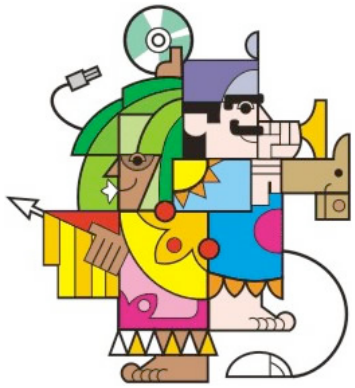
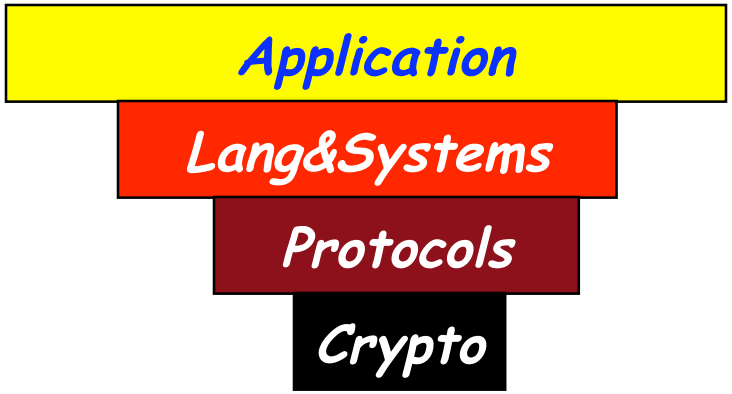
Security Requirements Engineering

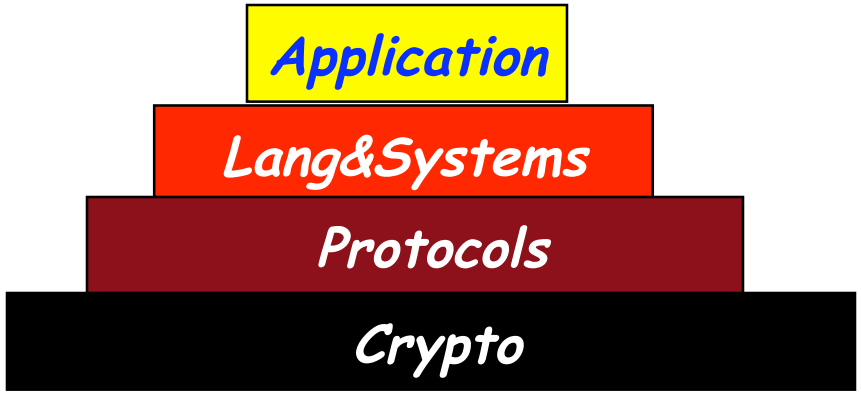
Nicola Zannone, Fabio Massacci
and John Mylopoulos



Security in the Internet Age

The number of security incidents reported to CERT has been growing exponentially, from 250 in 1990, 2,500 in 1995, 10,000 in 1999, 22,000 in 2000, 80,000 in 2002.
The strength of security guarantees provided by technology is inversely proportional to the size of the software layer security technology is applied to [Wing98]





Security in SE Practice

- The usual approach to security for a software system is to identify security requirements *after* system design.
- Most SE proposals focus on
  - ✓ protection aspects of security and explicitly deal with a series of security services (integrity, availability, etc.)
  - ✓ related protection mechanisms (password, crypto, etc.)
- Security mechanisms have to be fitted into a pre-existing design
  - ✓ may not be able to accommodate them
  - ✓ security requirements can generate conflicts with functional requirements of the system
- Big gap between solutions and the requirements of the entire system

Security and Requirements Engineering

- Traditional RE approaches treat security as a non-functional requirement.
- According to this view, security requirements are modeled as quality constraints under which the system must operate
- These need to be accommodated along with other non-functional requirements (e.g., reliability and performance).

Security Requirements Engineering

- Introduce security requirements analysis in the early phases of the software development process
- This allows us to
  - ✓ elicit security requirements from the organizational environment
  - ✓ analyze security requirements within the organizational environment in which the software will operate
  - ✓ motivate the use of specific security mechanisms

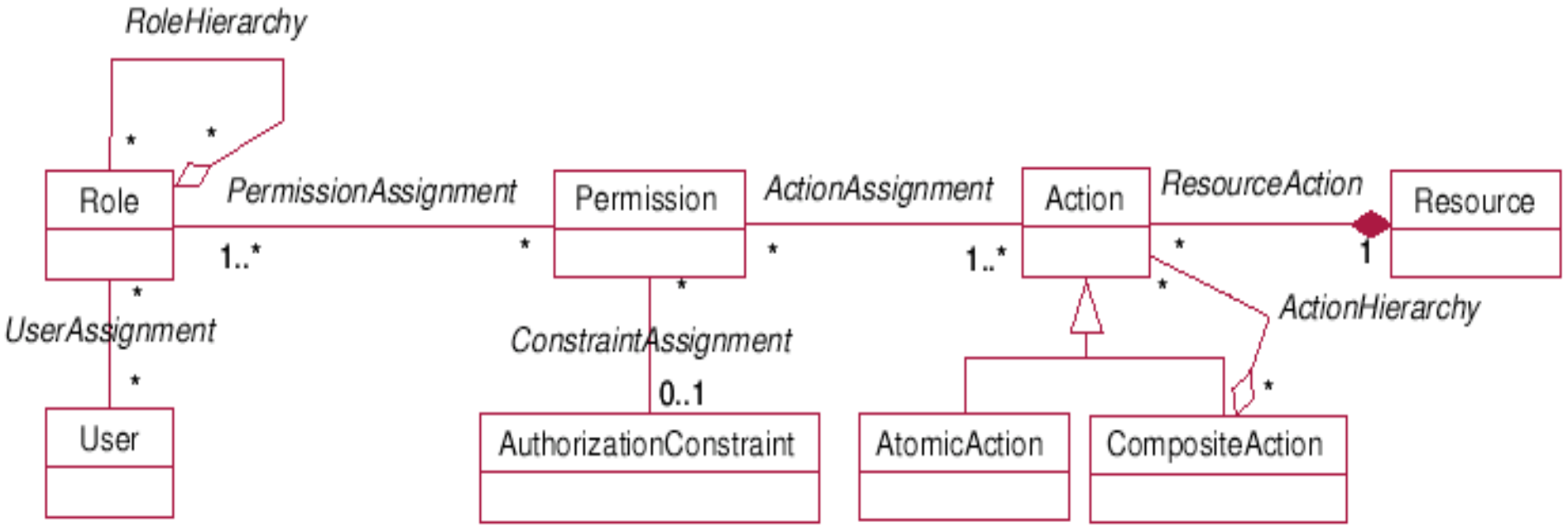
Related Work

- UML Proposals
  - ✓ SecureUML, Model-Driven Architecture [Basin et al.]
  - ✓ UMLsec [Juriens]
  - ✓ Abuse Cases [McDermott & Fox]
  - ✓ Misuse Cases [Sindre & Opdhal]
- Early Requirements Proposals
  - ✓ Anti-requirements [van Lamsweerde et al., Crook et al.],
  - ✓ Problem-Frames, Abuse Frames [Hall et al., Lin et al]
  - ✓ Security Patterns [Giorgini & Mouratidis]
  - ✓ Privacy Modelling [Liu et al., Anton et al,]

SecureUML

- D. Basin, J. Doser, and T. Lodderstedt, 2003
- Provides support for specifying access control policies
- The concepts of RBAC (Role-Based Access Control) are represented as metamodel types
  - ✓ User, Role, Permission, Action are types
  - ✓ UserAssignment, PermissionAssignment, RoleHierarchy are relations
  - ✓ AuthorizationConstraint is a predicate attached to a permission by the association ConstraintAssignement
    - ✓ Authorization constraints expressed in first-order logic
    - ✓ Used to establish the validity of the permission

SecureUML Metamodel



D. Basin, J. Doser, and T. Lodderstedt. Model driven security for process-oriented systems. In Proc. of SACMAT '03, 100-109. ACM Press, 2003.

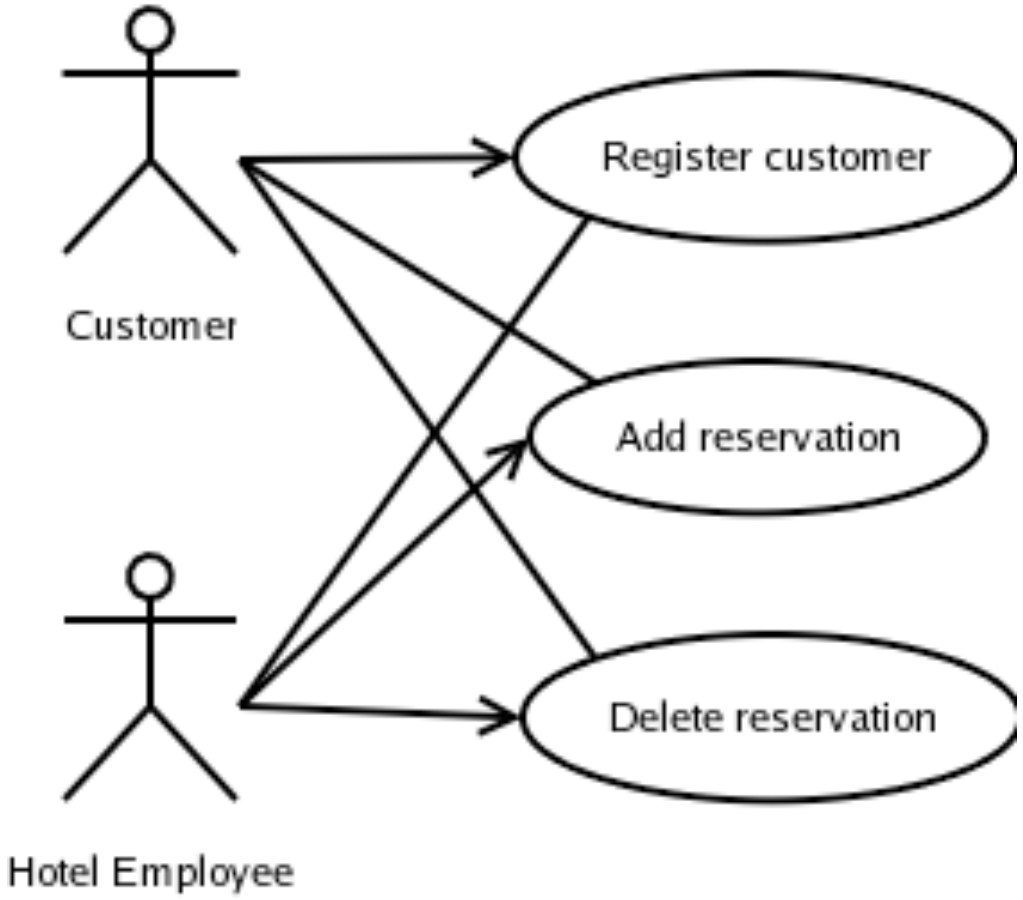
SecureUML Semantics and Limits

- An access control configuration is an assignment of users and permissions to roles
- SecureUML makes access control decisions based on an access control configuration and on the validity of authorization constraints in a certain system state
- Does NOT analyze security requirements within an organizational environment within which the software system will operate
- Requires knowledge of conflicting roles a priori
  - ✓ Does NOT detect conflicts in an organizational context.

Use Case Diagram

- Build a first sketch of system use.
- Offer a notation for describing the intended functionality of a system
  - ✓ Actors: an abstraction of a role that interacts with the system;
  - ✓ Use cases: define a function of interaction between the system and external actors;
  - ✓ Association links: connect actors with the use cases they use .

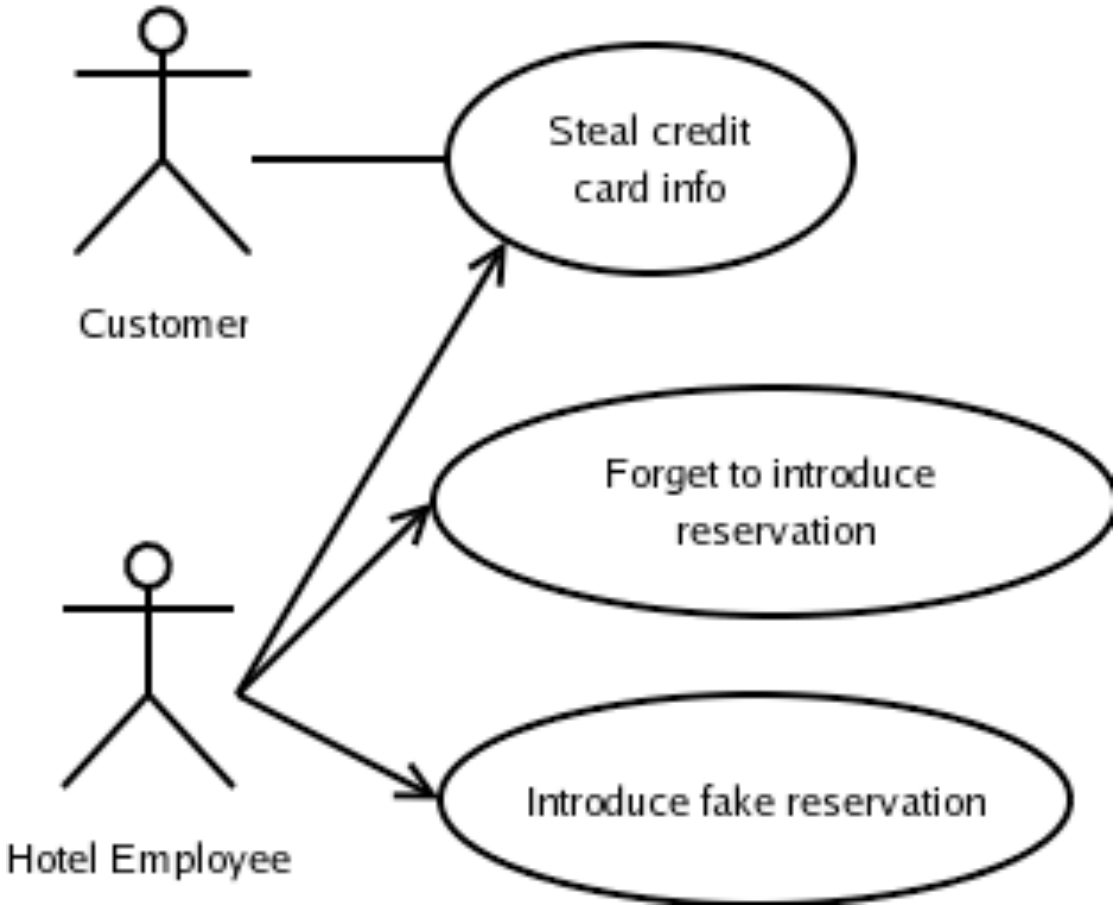
Reservation System



Abuse Cases

- McDermott & Fox, 1999
- Negative use cases for modeling security requirements
- Specify an interaction between a system and one or more actors, where the results of the interaction are harmful to the system or one of the actors of the system
- Actors are the same that participate in use cases

Reservation System Abuses



Limits of Abuse Cases

- Model security requirements separately from functional requirements
  - ✓ Abuse case diagrams show abuse only, not abuse together with normal use
  - ✓ Do not investigate relations between uses and abuses

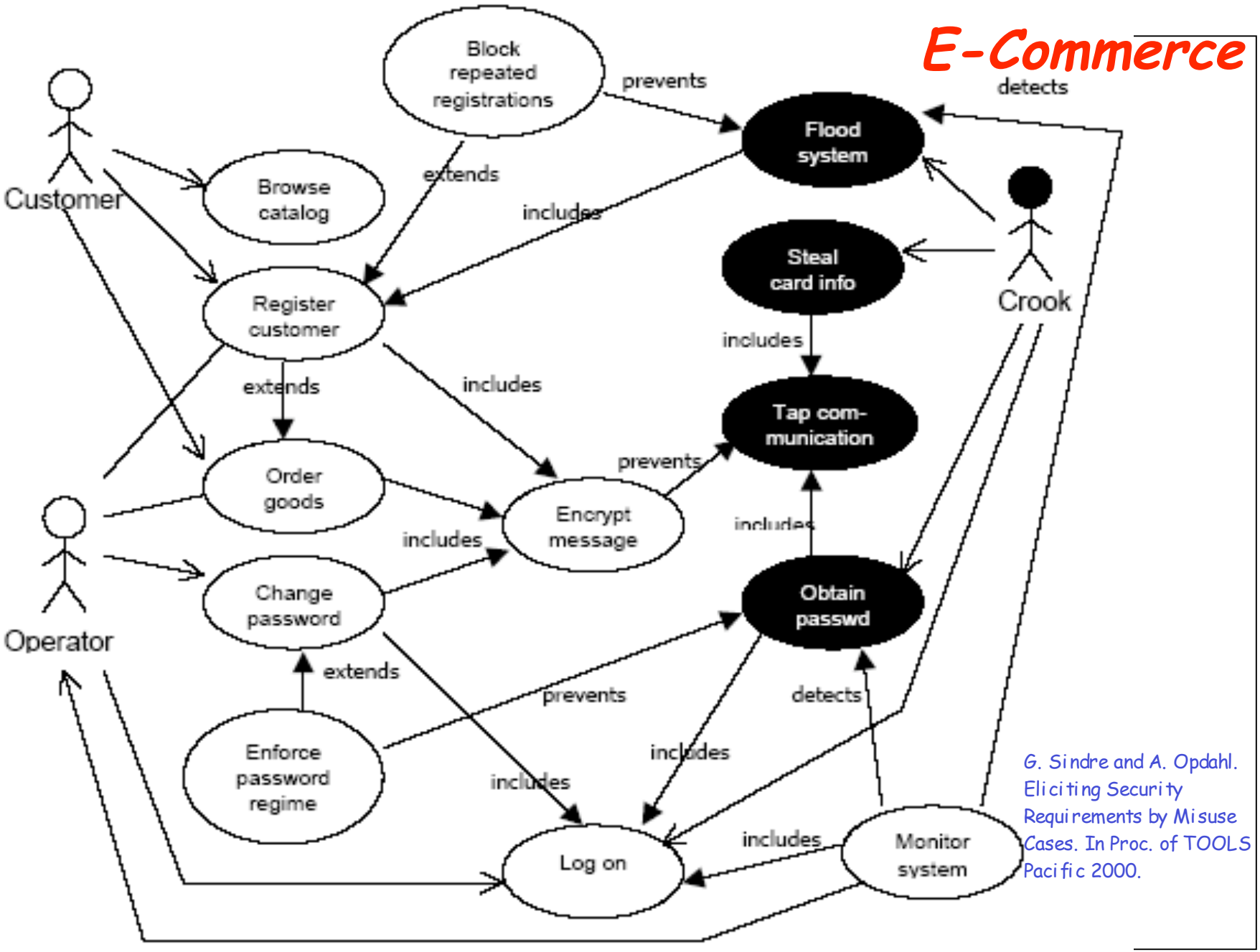
Misuse Cases

- Guttorm Sindre and Andreas Opdahl, 2000
- Extend use cases for modeling security requirements
- Specify behaviour that the system should prevent
- Specify how a misuser can damage the system

Concepts

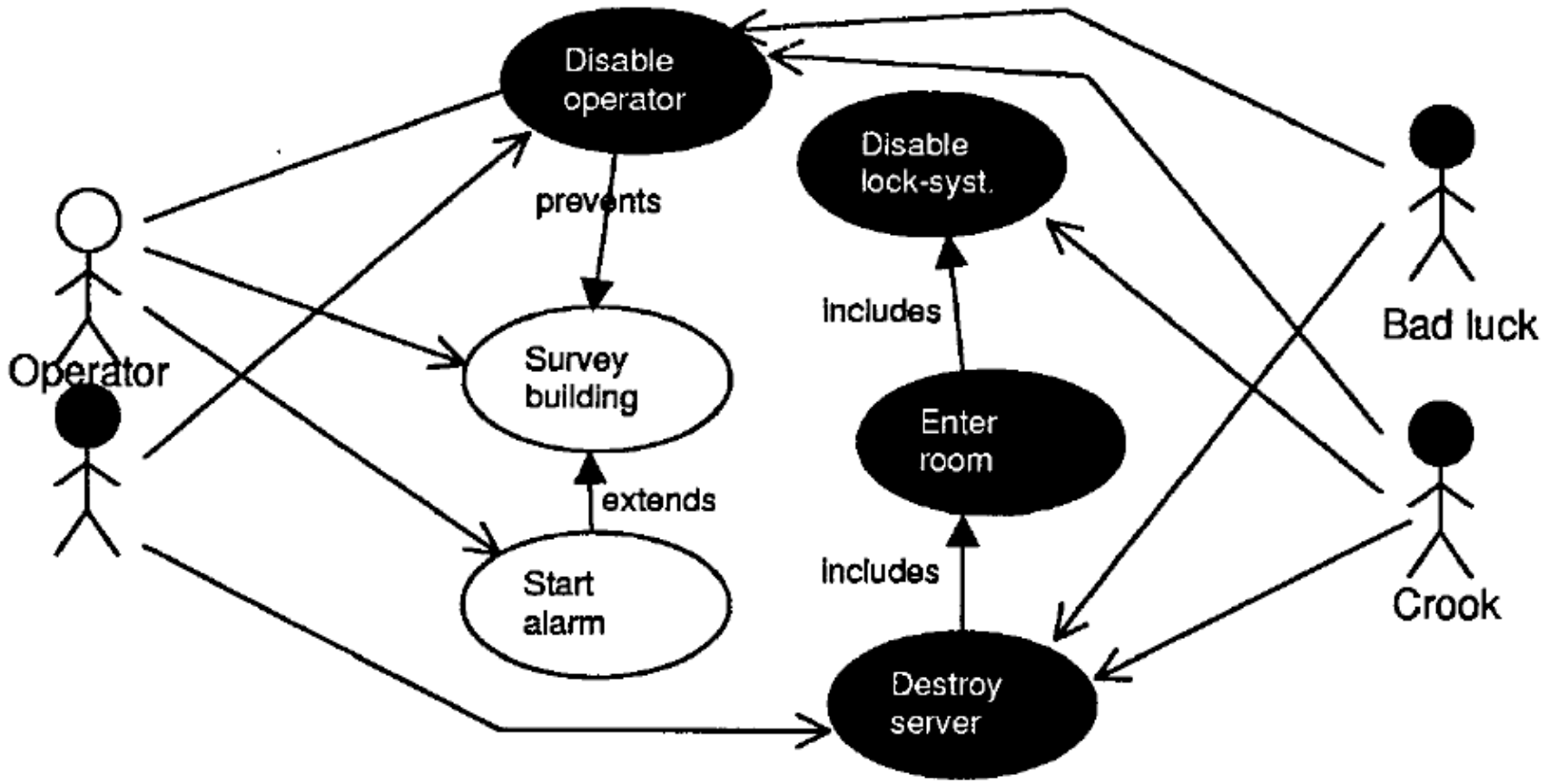
- Misuser
  - ✓ hostile actor
  - ✓ a similar notation as an actor in use cases, except the misuser has a black "head" instead of white
- Misuse case
  - ✓ course of actions intended to harm a stakeholder or the system
  - ✓ behavior that is not wanted in the system
  - ✓ illustrated by black circles
- Use cases
  - ✓ functionality of the system-to-be
  - ✓ countermeasure against misuse
- Relations
  - ✓ "includes" and "extends"
  - ✓ "prevents": use case prevents the activation of a misuse case
  - ✓ "detects": use case detects the activation of a misuse case

E-Commerce



G. Sindre and A. Opdahl. Eliciting Security Requirements by Misuse Cases. In Proc. of TOOLS Pacific 2000.

Special mis-actors



G. Sindre and A. Opdahl. Eliciting Security Requirements by Misuse Cases. In Proc. of TOOLS Pacific 2000.

Advantages

- Focus on security in the early phases of the software development process
- Increase the chance of discovering threats that otherwise would have been ignored
- Help trace and organize the requirements specification
- Help evaluate requirements
  - ✓ the real cost of implementing a use case includes the protection needed to mitigate all serious threats to it
- Easy to reuse in new development projects

Disadvantages

- Use/Misuse case are informal
  - ✓ No clearcut semantics
  - ✓ (Hence) NO formal analysis
- No knowledge on how to write *good quality* misuse cases
- Focus is on the system-to-be
- NOT suitable for all kinds of threats
- There is not always an identifiable misuser and the misuse case may not always consist of an identifiable sequence of actions

Obstacles in KAOS

- Identify goal violation scenarios
- An obstacle to some goal is a condition whose satisfaction may prevent the goal from being achieved
- An obstacle O is said to obstruct a goal G in domain Dom iff

  {O,Dom} |= ¬G Obstruction

  Dom |≠ ¬O Domain consistency

Obstacle Analysis

- Obstacle analysis takes a pessimistic view of goals … what could go wrong?
- Resolution techniques for obstacles: goal substitution, agent substitution, goal weakening, goal restoration, obstacle prevention and obstacle mitigation
- Obstacle analysis is an iterative process
  - ✓ it may produce new goals for which new obstacles may be generated and resolved

Security Goals

- High level of abstraction -- metaclasses
  - ✓ Authentication: "You are who you claim you are"
  - ✓ Integrity: "What you get is what I sent"
  - ✓ Confidentiality: "Accessible only to those authorized"
  - ✓ Availability: "We count on that it works"
  - ✓ Auditability: "Operations are transparent, provable"
  - ✓ Non-repudiation: 'We cannot deny we talked'
  - ✓ Certification: 'We are authorized to act'
- Each security goal has to be instantiated into application-specific security goals

Confidentiality

Goal Confidentiality

Confidentiality

Goal Confidentiality

Goal *Maintain*[SensitiveInfoNotKnownByUnauthorizedAgent]

FormalSpec ∀ ag: Agent, ob: Object

¬ Authorized(ag,ob.Info) ⇒ ¬ Knows(ag,ob.Info)

If agent ag is not authorized to access info about object ob, then he doesn't know info about object ob

Confidentiality

Goal Confidentiality

Goal *Maintain*[SensitiveInfoNotKnownByUnauthorizedAgent]

FormalSpec ∀ ag: Agent, ob: Object

¬ Authorized(ag,ob.Info) ⇒ ¬ Knows(ob.Info)

Goal *Maintain*[PaymentMediumNotKnownBy3rdParty]

FormalSpec ∀ p: Agent, acc: Account

¬ (Owns(p,acc) ∨ Manages(p,acc))

⇒ ¬ (Knows(p,acc.Acc#) ∧ Knows(p,acc.PIN))

If agent p is not the owner of account acc and he does not manage it, then he doesn't know number and PIN of the account

Obstacle Analysis

- Take the negation of the goal

  ∀ p: Agent, acc: Account

  (NG) ¬ (Owns(p,acc) ∨ Manages(p,acc))

  ∧ (Knows(p,acc.Acc#) ∧ Knows(p,acc.PIN))

- Suppose that the domain theory contains the following properties

  (D1) ∀ p,ag:Agent, acc:Account

  [Owns(p,acc) ∧ Knows(ag,p.name) ⇒ Knows(ag,acc.Acc#)]

  (D2) ∀ ag: Agent, acc: Account

  [Knows(ag,acc.Acc#) ⇒ Knows(ag,acc.PIN)]

- We can formally derive the following potential obstacle

  (O) ∀ p,ag: Agent, acc: Account

  ¬ (Owns(ag,acc) ∨ Manages(ag,acc))

  ∧ Knows(ag,p.name) ∧ Owns(p.acc)

Anti-goals

- Obstacles suffice for modeling and resolving *non-intentional* obstacles (e.g., accidental obstacles)
- Too limited for modeling and resolving *intentional* obstacles (malicious obstacles)
- Active attackers need to be modeled as well, together with their own goals, capabilities, and the vulnerabilities they can monitor or control (anti-models)
- Anti-goals represent intentional obstacles to security goals.

Building Anti-models

- Root anti-goal is obtained by negating a security goal.
- For each anti-goal, potential attackers are identified (WHO)
- For each anti-goal and corresponding attacker, higher level anti-goals are identified (WHY)
- For each anti-goal and corresponding attacker, lower level anti-goals are identified (HOW)
- AND/OR refinement process for anti-goals
  - ✓ realizable by the attacker (anti-requirements)
  - ✓ realizable by the attackee (vulnerabilities)
- Anti-models are derived from anti-goal formulations
- Anti-requirements are defined in terms of the capabilities of the corresponding attacker

Identifying Antigoals

Goal *Maintain*[PaymentMediumNotKnownBy3rdParty]

FormalSpec ∀ p: Agent, acc: Account

¬ (Owns(p,acc) ∨ Manages(p,acc))

⇒ ¬ (Knows(acc.Acc#) ∧ Knows(acc.PIN))

AntiGoal Achieve[PaymentMediumKnownBy3rdParty]

FormalSpec ∃ ag: Agent, ob: Object

¬ (Owns(p,acc) ∨ Manages(p,acc))

∧ Knows(acc.Acc#) ∧ Knows(acc.PIN)

Refining antigoals

By asking "what are sufficient conditions for someone unauthorized to know the number and PIN of an account simultaneously?"

AntiGoal *Achieve*[PaymentMediumKnownBy3rdPartyFromPinSearching]
FormalSpec ∃ ag: Agent, ob: Object
¬ (Owns(p,acc) ∨ Manages(p,acc)) ∧ Knows(acc.Acc#)
∧ (∃ x:PIN) [Find(p,x) ∧ Match(x,acc.Acc#)]

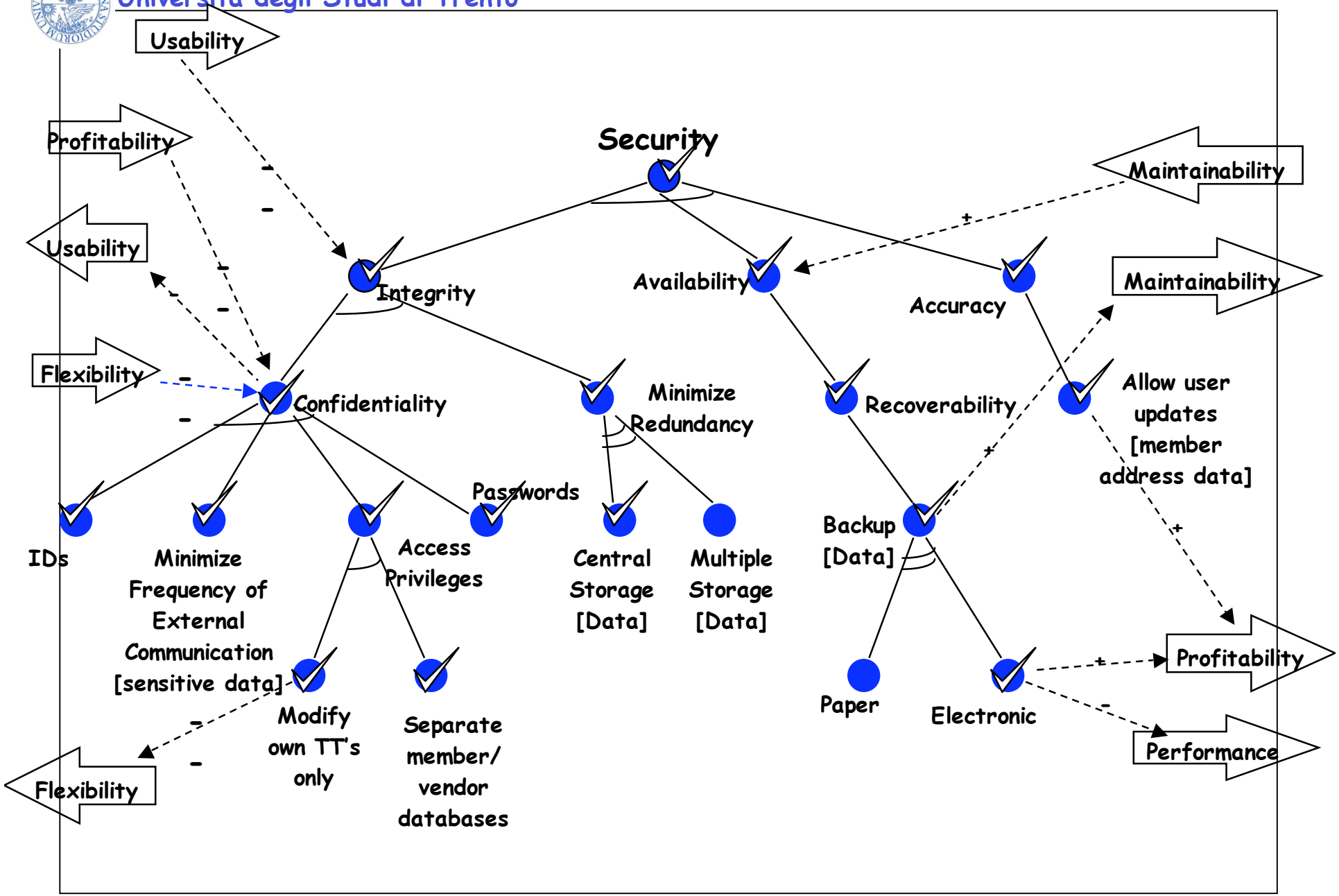
AntiGoal *Achieve*[PaymentMediumKnownBy3rdPartyFromAcc#Searching]
FormalSpec ∃ ag: Agent, ob: Object
¬ (Owns(p,acc) ∨ Manages(p,acc)) ∧ Knows(acc.PIN)
∧ (∃ y:Acc#) [Find(p,y) ∧ Match(acc.PIN,y)]

Limits of Antigoals

- Modeling attackers is difficult
- We have to consider all the possible obstacles even ones unknown
  - ✓ Many protocols for security have been proven to be incorrect several years after they were designed
- Many system vulnerabilities depend on implementation
- Software vulnerabilities are not completely known

i*/Tropos and Security

- Security is treated as a non-functional requirement, modelled and analyzed in terms of softgoals.
- These are goals that don't have a formal definition; consequently, they don't have a clear-cut criterion as to whether they are fulfilled or not (hence their name…)
- Softgoals are *satisficed*, rather than satisfied; in other words, softgoal fulfillment is relative and "good enough", rather than absolute and optimal.
- Goal analysis is then used to verify whether security goal are satisficed


Usability
Profitability
Usability
Flexibility
Security
Integrity
Availability
Accuracy
Maintainability
Maintainability
Confidentiality
Minimize Redundancy
Recoverability
Allow user updates [member address data]
IDs
Minimize Frequency of External Communication [sensitive data]
Passwords
Access Privileges
Central Storage [Data]
Multiple Storage [Data]
Backup [Data]
Modify own TT's only
Separate member/ vendor databases
Paper
Electronic
Profitability
Performance
Flexibility

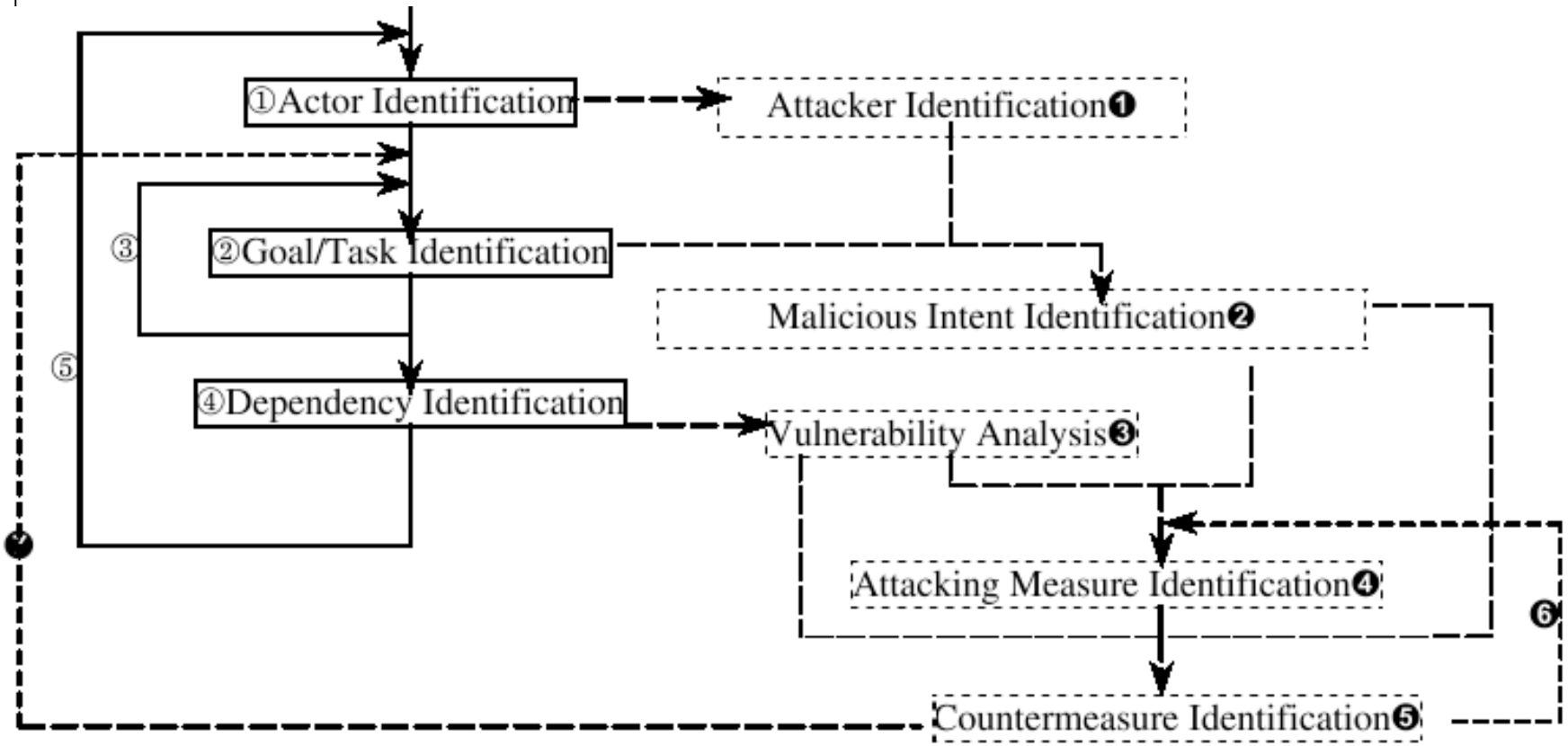
Modeling Security Concerns with i*

- A methodological framework for security requirements analysis, which builds on a strategic actors modeling framework - i*
- Offers facilities for analyzing threats, vulnerabilities, and countermeasures
- Relates social concerns with technology by explicitly integrating security analysis with the normal requirements analysis process

Security Requirements Analysis

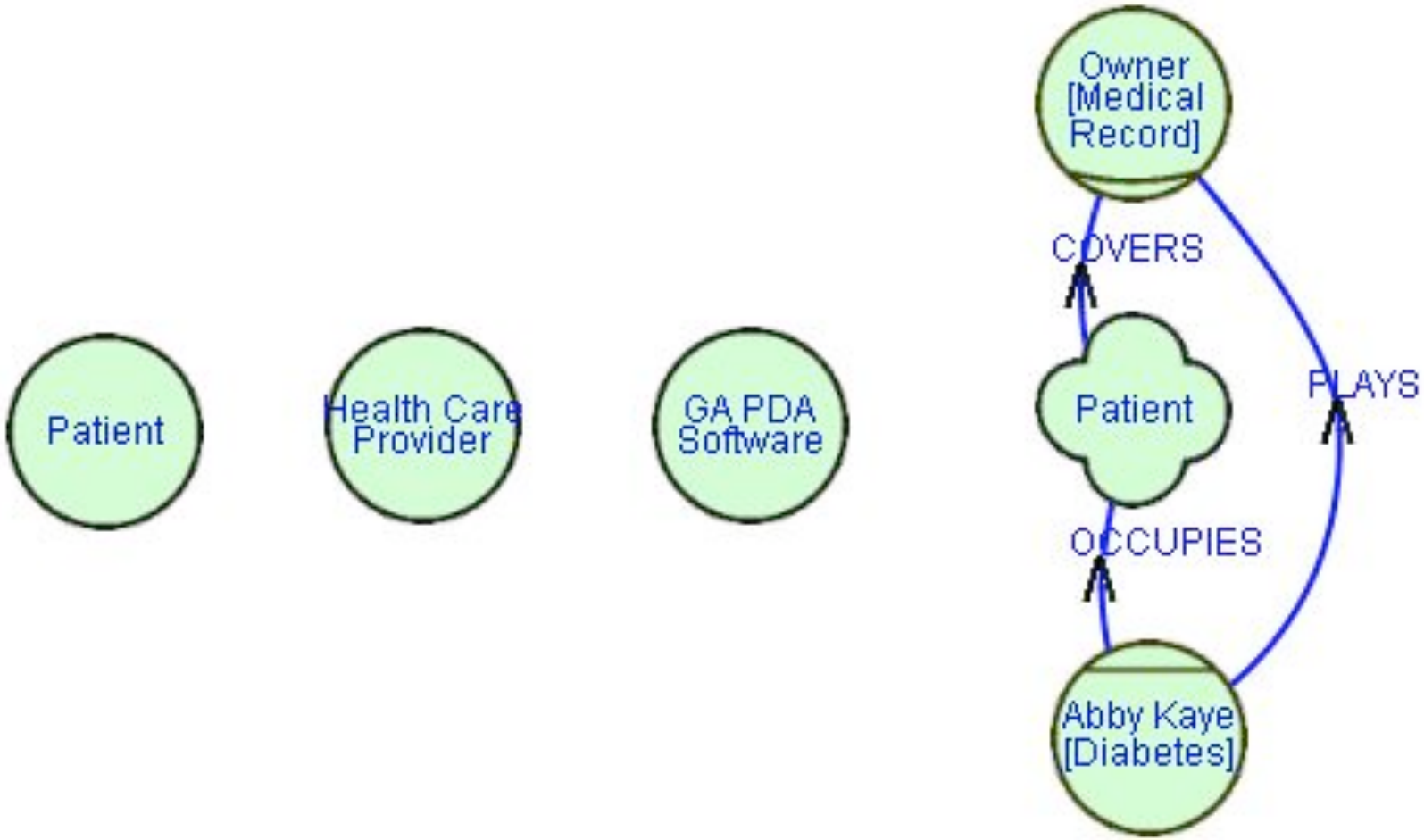
- Attacker analysis
  - ✓ Identify potential system abusers and their malicious intents
- Dependency Vulnerability Analysis
  - ✓ Identify vulnerability points in an actor dependency network
- Countermeasure Analysis
  - ✓ Make decisions on how to protect security from potential attackers and vulnerabilities
    - ✓ Identify attacks and threats
    - ✓ Introduce countermeasures
  - ✓ Countermeasure analysis is an iterative process
    - ✓ Adding protection measures may bring new vulnerabilities to the system

Requirements Elicitation Process

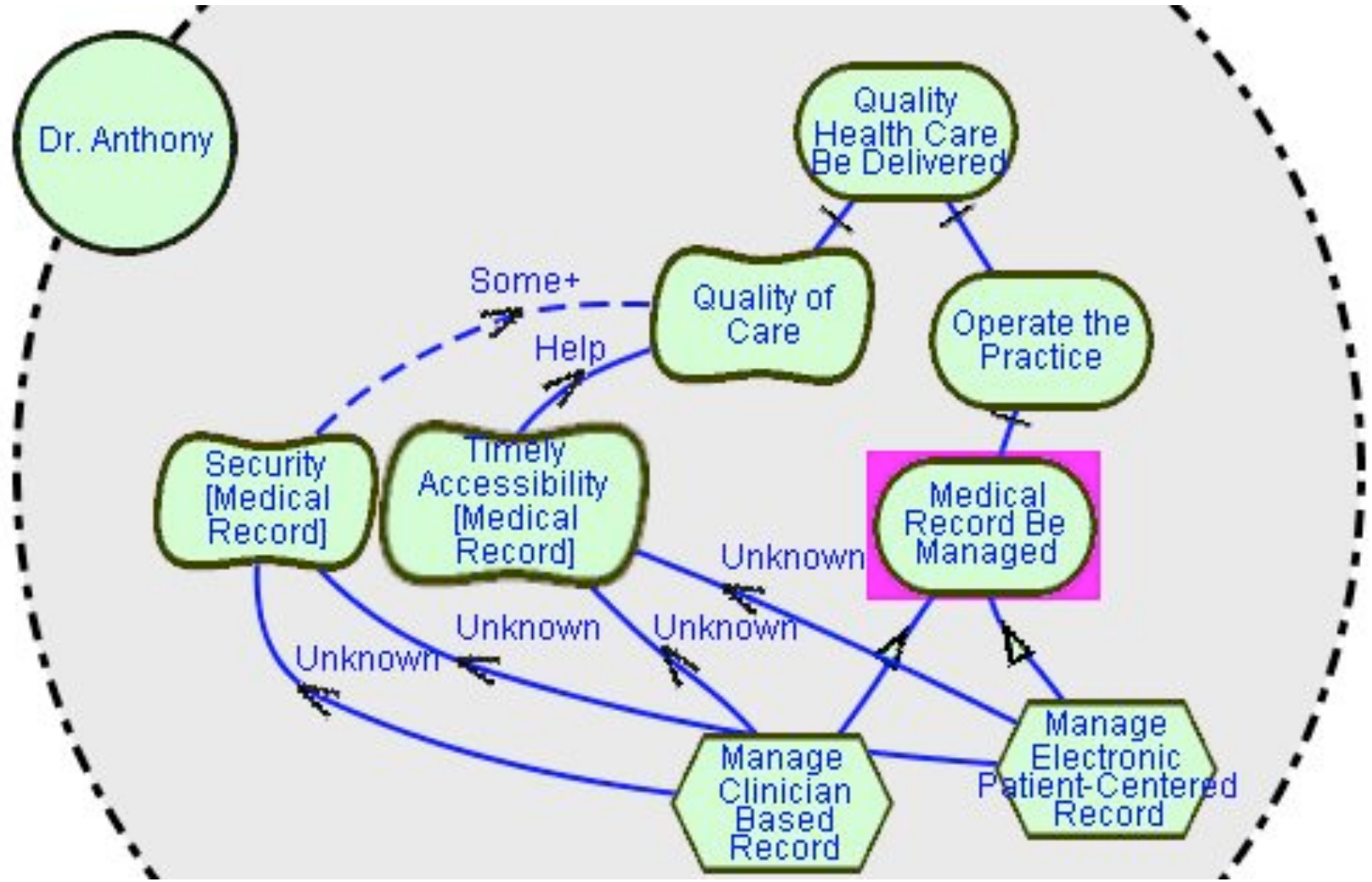


[Liu et al. 2003] Security and Privacy Requirements Analysis within a Social Setting. In Proc. of RE'03, pages 151-161.

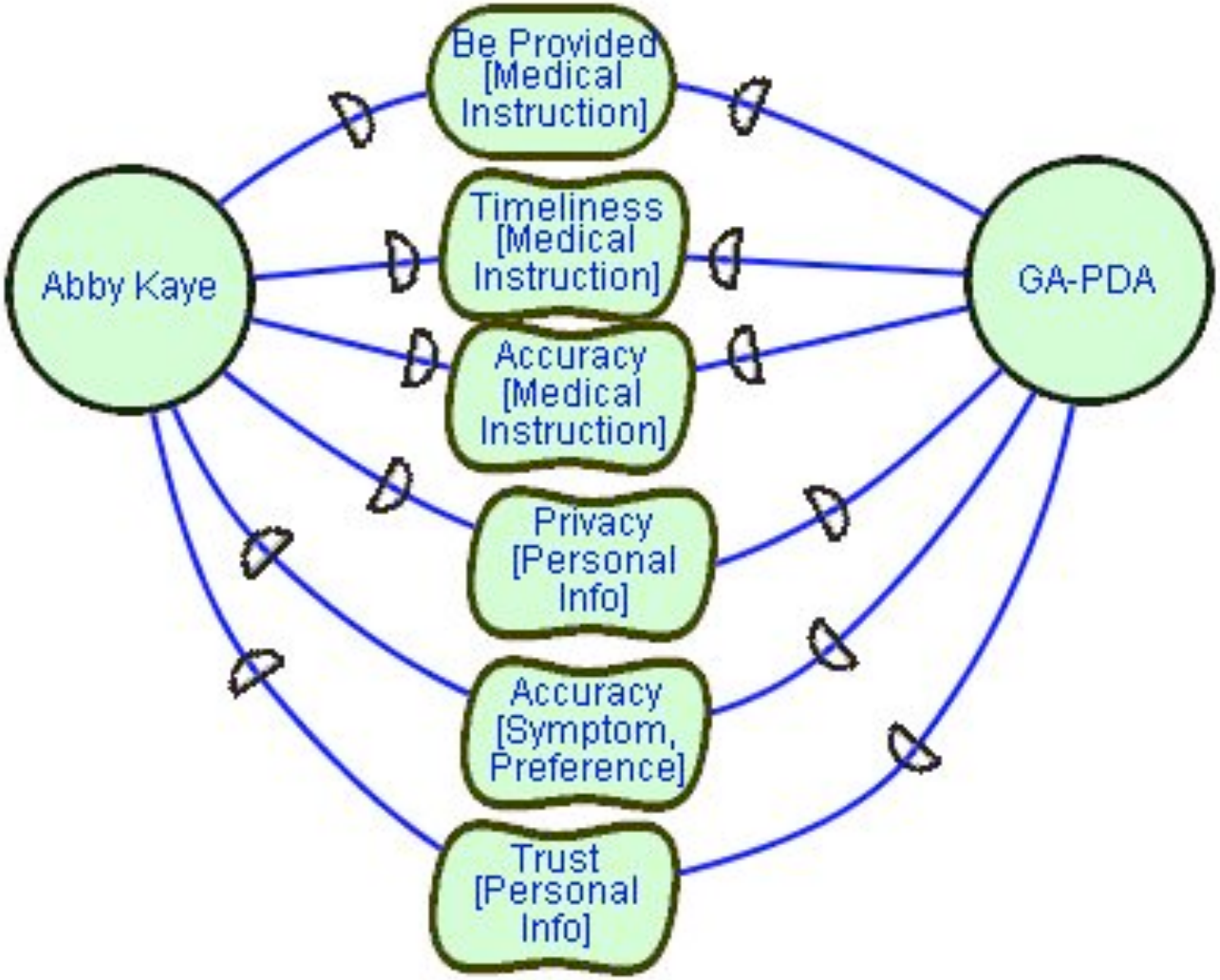
Who is Involved with the System?



What Do Actors Want?

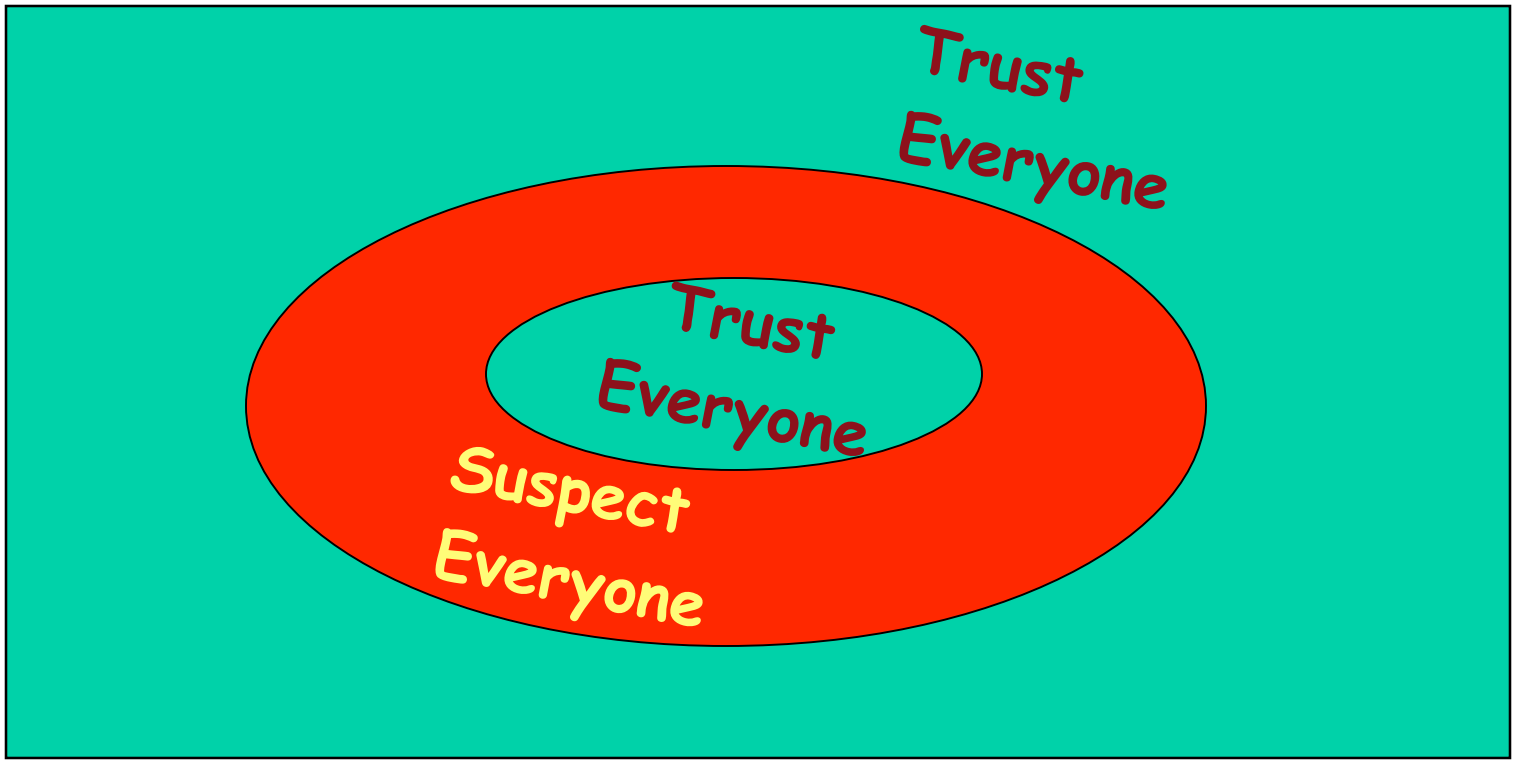


How Do Actors Relate to Each Other?



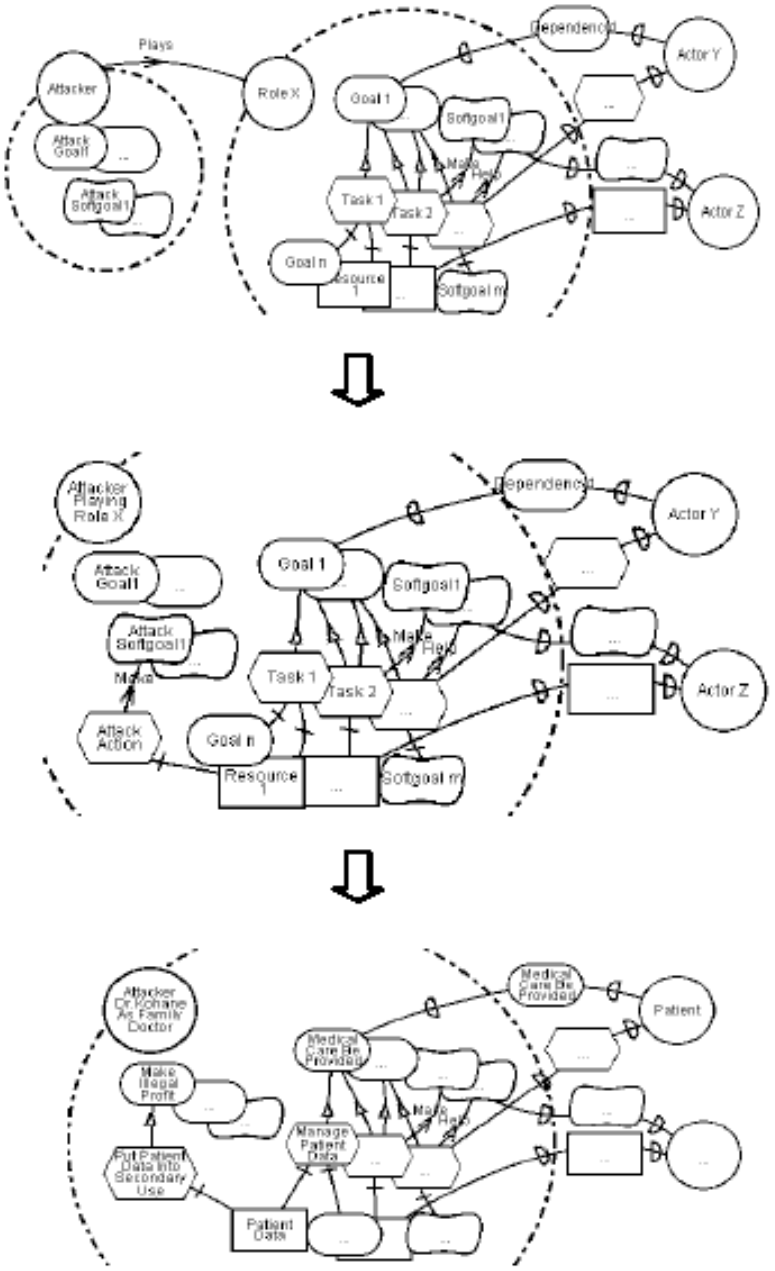
Attacker Identification

- Set an outer and an inner trust boundary
- All actors between the two boundaries are assumed "guilty until proven innocent".



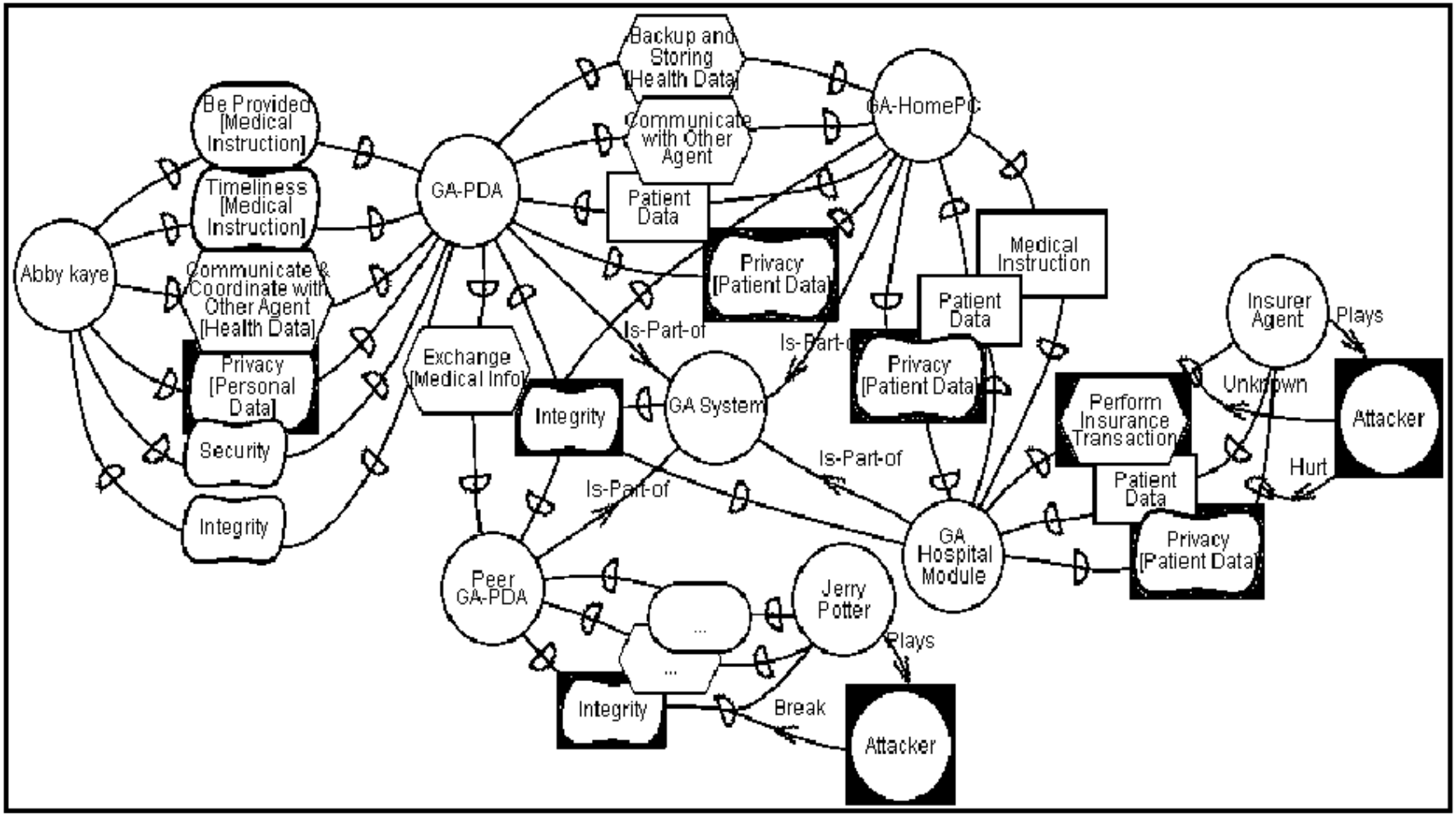
Attacker Analysis

- Actors are assumed guilty until proven innocent
- Any one of the actors identified can be a potential attacker
- The attacker inherits the intention, capabilities and social relations of the corresponding legitimate actor
- External attackers can be also considered



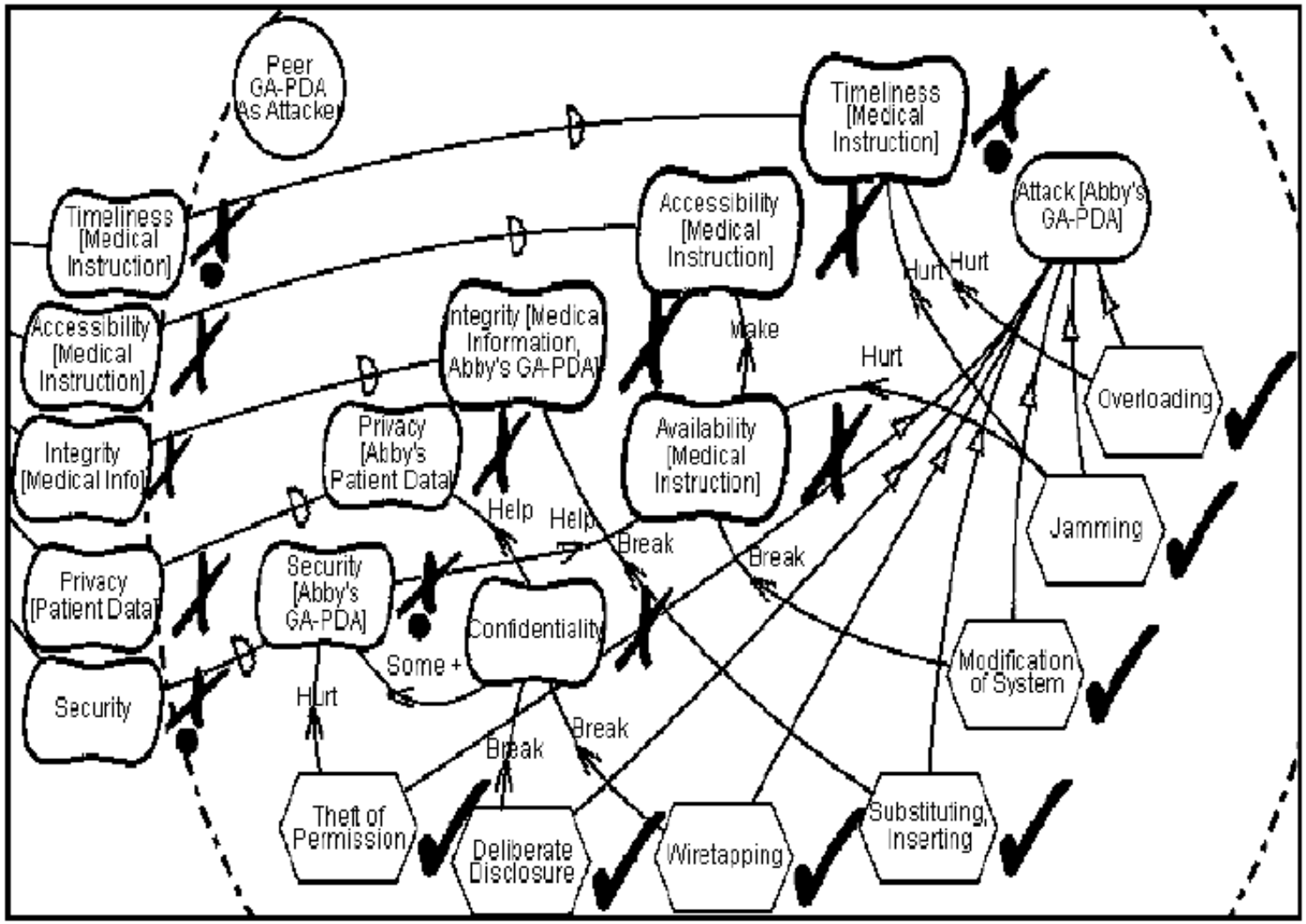
[Liu et al. 2003] Security and Privacy Requirements Analysis within a Social Setting. In Proc. of RE'03, pages 151-161.

Dependency Vulnerabilities Analysis



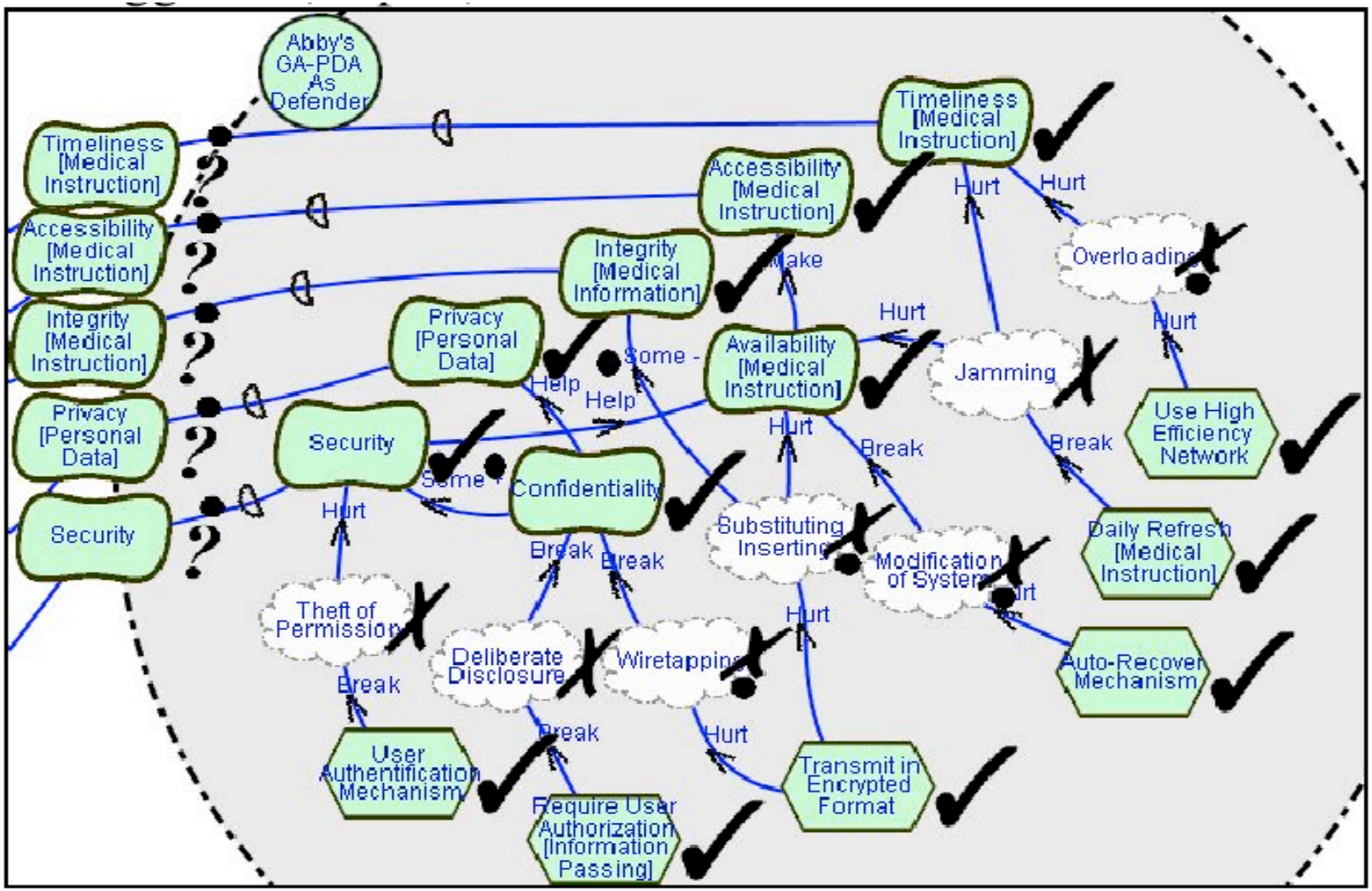
[Liu et al. 2003] Security and Privacy Requirements Analysis within a Social Setting. In Proc. of RE'03, pages 151-161.

Attacks and Threats Identification



[Liu et al. 2003] Security and Privacy Requirements Analysis within a Social Setting. In Proc. of RE'03, pages 151-161.

Countermeasure Analysis



[Liu et al. 2003] Security and Privacy Requirements Analysis within a Social Setting. In Proc. of RE'03, pages 151-161.

Evaluation of Alternatives

- Assume ranking criteria :

  Quality of Care > Easy of Use > Privacy > Security

- Top two best designs:
  - ✓ Manage Unified Electronic Patient Record + Created By Patient + Store in Patient Home PC + Secured Web Access + Reserve Record Until Time Expired
  - ✓ Manage Clinician-based Record + Created By Clinician + Store in Clinical DB + Secured Web Access + Transfer Record to Provider of Patient's Choice

So What?

- We have a method for identifying design features that can contribute to a security goal.
- We also have a method for evaluating the degree to which a security goal is satisficed, given a set of design decisions.
- But, formal analysis is minimal.
- Moreover, the method proposed by Chung is not specific to security!

i*/Tropos and Security Requirements

- i*/Tropos has not been designed with security in mind
- Lack of the ability to capture at the same time functional and security features of an organization
- The process of integrating security and functional requirements throughout the whole range of the development stages is quite ad hoc
- The concept of softgoal that Tropos uses to capture security requirements fails to adequately capture some constraints that security requirements often represent
  - ✓ REMARK: softgoals are goals that have no clear-cut definition and/or criteria for deciding whether they are satisfied or not
- The methodology fails to provide concepts and processes to model trust relationships

Actor Dependency

- A dependency between two actors means that the dependee will take responsibility for fulfilling the functional goal of a depender.
- Major assumption is that if you provide a service (fulfil a goal, carry out a task, deliver a resource) you also have the authority to use it, but...
- No way to specify or check whether the dependee is actually authorized to do so.
- It can happen that an actor depends on another for a service, but the dependee is neither the owner of the service nor authorized to provide the service.

Claim

- The concepts of *ownership* and *permission* are at the very foundation of all security concerns …
  - ✓ No ownership, no security to worry about.
  - ✓ If people didn't own human rights, privacy rights, physical property, security would be a meaningless word.
- Permission as a complementary notion to obligation is well-accepted in Deontic Logics
- So, we introduce it in our modeling framework

Secure Tropos

- Use the concepts offered by Tropos for actor, goal, task, and resource
- Make explicit who is the requester of a service, who is the legitimate owner of a service and who is able to provide a service
- Refinement of Tropos dependency
  - ✓ Trust relationship on Actor/service/Actor
  - ✓ Permission ≠ Execution

Requiring, Owning, and Provisioning

- Requiring
  - ✓ What actors want
- Owning
  - ✓ What actors own
  - ✓ The owner has full authority on the achievement of a goal, execution of a task, or use of a resource
  - ✓ Owner can also delegate this authority to other actors
- Provisioning
  - ✓ Identify actors who have the capability and are willing to achieve a goal, execute a plan, or deliver a resource.

Delegation

- Delegation of permission
  - ✓ Used to model formal passage of authority
  - ✓ The delegatee thinks "I have permission to fulfill the service (even if I do not need to)"
- Delegation of execution
  - ✓ Used to model formal passage of responsibility
  - ✓ The delegatee thinks "I have to deliver the service"

Trust

- Trust is a relation between two actors representing the expectation of one actor (the trustor) concerning the capabilities and behavior of the other (the trustee)
- Trust of permission
  - ✓ the trustor believes that the trustee will not misuse the goal, task, or resource
- Trust of execution
  - ✓ the trustor believes that the trustee will achieve the goal, execute the task, or deliver the resource
- Trust is a mental antecedent of delegation
  - ✓ Delegation is an action due to a decision, whereas trust is a mental state driving such decision

Comparing Tropos and Secure Tropos

- Tropos Model
  - ✓ Actor Properties
    - ✓ goals
  - ✓ Actor Relationships
    - ✓ dependency

- ➔Secure Tropos Model
  - ✓ Actor Properties
    - ✓ goals
    - ✓ entitlements
    - ✓ capabilities
  - ✓ Actor Relationships
    - ✓ trust of execution
    - ✓ delegation of execution
    - ✓ trust of permission
    - ✓ delegation of permission

Comparing Tropos and Secure Tropos

- Dependency = Delegation of Execution + Trust of Execution
  - ✓ if designer says A depends on B for G then A has actually delegated fulfillment of G to B and trusts that B will do it
  - ✓ if one depends on X to fulfill G, X is by default authorized to do G
- Wanting = Owning
  - ✓ if designer says that A wants G, of course A is authorized to fulfill G
- Implicit Provisioning
  - ✓ When designer stops dependency chain on goal G at agent B, it means that B will take care of it
- Trust vs Delegation
  - ✓ We want to also model scenarios where actors must delegate permission or execution to other actors they do not trust

Secure Tropos -- Methodology

- Actor Diagram
  - ✓ Goals an actor wants, *provides, or owns*
- Functional Requirements Model
  - ✓ Delegation of Execution
- *Trust Model*
  - ✓ *Trust of Execution and Permission Relations*
- *Trust Management Implementation*
  - ✓ *Delegation of Permission*
- Refinements by
  - ✓ Goal Decomposition within an Actor Diagram
  - ✓ Goal (Execution *or Permission*) Delegation to actors
  - ✓ *Modification of Trust Relationship*

Underlying Formal Model

- Formal Model
  - ✓ Answer Set Programming (aka Datalog)
  - ✓ Deduction, Satisfiability, Abduction
- Models (secure-i* Diagrams)
  - ✓ Extensional properties of classes (and instances)
- Axioms
  - ✓ Intensional properties and rules
- Properties
  - ✓ Specify conditions which must not be true in the model
  - ✓ Formulae that may be in true or may not be true

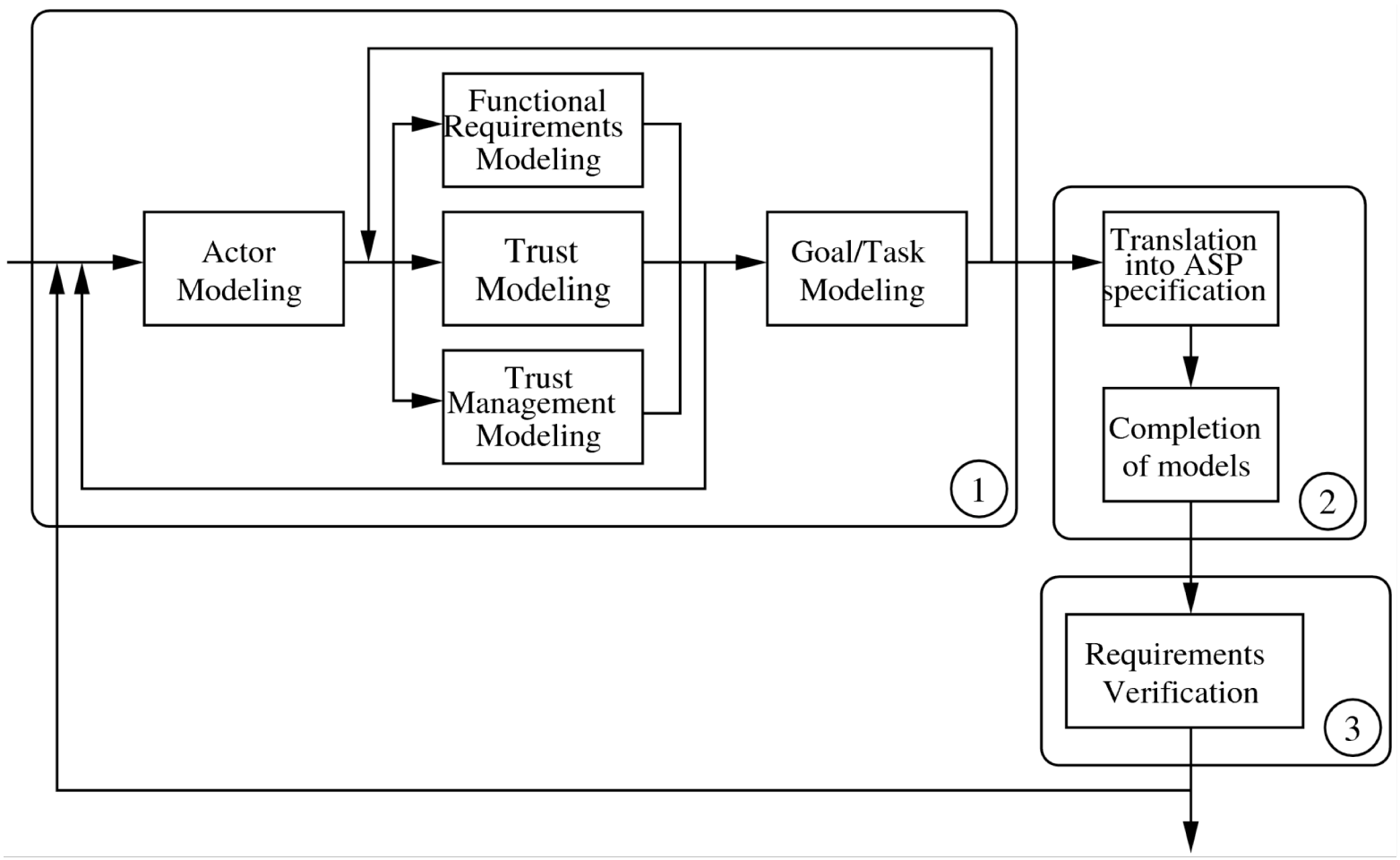
Examples

- Axioms
  - ✓ Intensional properties and rules
    - ✓ has_perm(A,S)<-- owns(A,S)
    - ✓ has_perm(A,S)<-- delegate(perm,B,A,S)&has_perm(B,S)
- Properties
  - ✓ Specify conditions which must not be true in the model
    - ✓ <-- delegation(perm,A,B,S)& not trustChain(perm,A,B,S)

Secure Tropos for Security Services

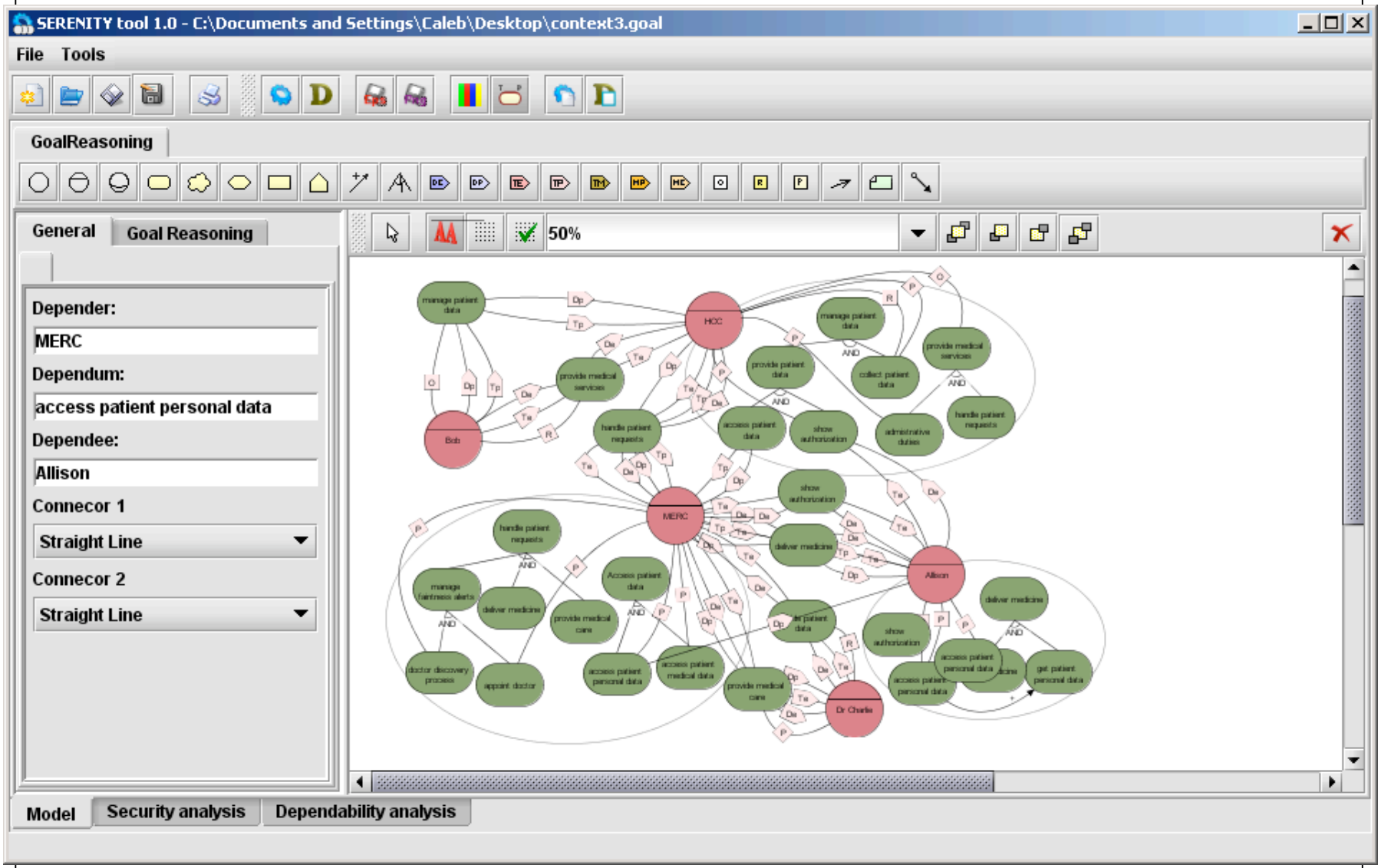
- Security Services as Patterns…
  - ✓ Authorization
    - ✓ The owner of a service is confident that his service will not be misused
  - ✓ Availability
    - ✓ Actors are confident they will satisfy their objectives
    - ✓ Actors who need to have permission for achieving their duties have such permission
  - ✓ Privacy
    - ✓ Actors have permission on a service only if they need such a permission for achieving their duties (Need-to-have principle)

Requirements Analysis Process



Computer-Aided SRE

- Draw the graphical (Secure) Tropos models
- Diagrams (automatically) mapped into a Formal Model
  - ✓ Datalog specifications
  - ✓ Formal Tropos specification
- Check the properties on the Formal Model
  - ✓ Integration within different Datalog solvers
    - ✓ DLV System, ASSAT, C-Models, S-Models


SERENITY tool 1.0 - C:\Documents and Settings\Caleb\Desktop\context3.goal
File Tools
GoalReasoning
General
Goal Reasoning
Depender:
MERC
Dependum:
access patient personal data
Dependee:
Allison
Connecor 1
Straight Line
Connecor 2
Straight Line
50%
Model
Security analysis
Dependability analysis

Social vs Individual

- Tropos involves two different levels of analysis
  - ✓ Social level: the structure of organizations are defined associating to every role (or position) objectives and responsibilities
  - ✓ Individual level: agents are not only defined with their objectives and responsibilities, but also they are associated to roles (or positions) they can play
- In Tropos there is no explicit separation between the two levels, and it is very difficult to maintain the consistency

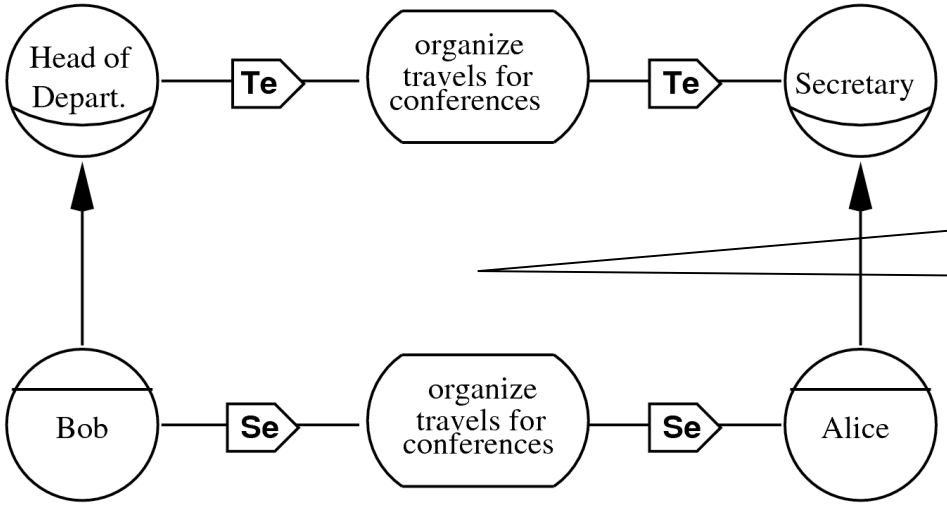
Social vs Individual (II)

- It is possible that requirements are given only at individual level or at social level
- Social => Individual
  - ✓ The agents playing a social role "should" inherit properties and social relations of that role
  - ✓ If Alice play R1 and R1 trusts R2 and Bob plays R2 then Alice trusts Bob…
- Useful feature to "complete" models in Computer-aided RE
  - ✓ Social relationships are always drawn in RE
    - ✓ After all they are among the system specifications
  - ✓ Designers must only draw social relationships and the reasoning system does the rest

Distrust

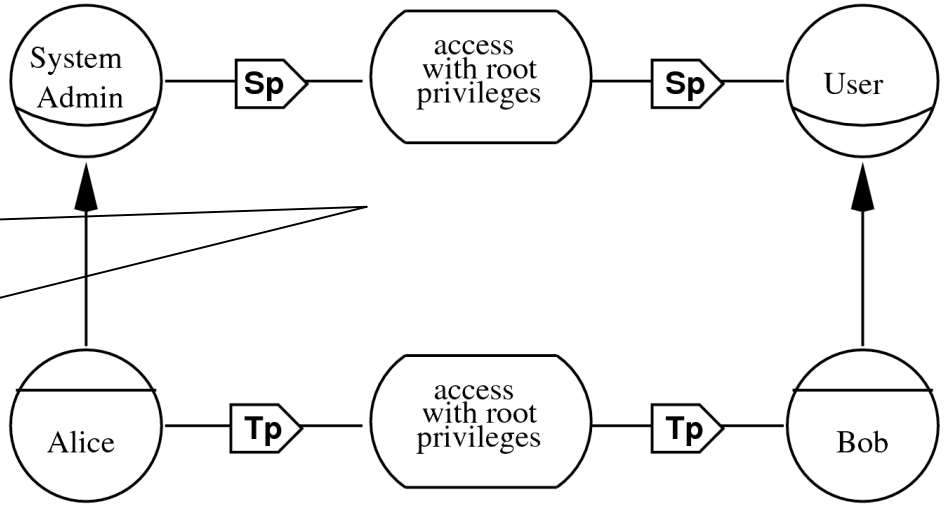
- Need for negative authorizations to help designers in shaping the perimeter of positive trust
- Distrust is a relation between two actors representing the expectation of one actor about the inabilities and misbehaviour of the other
- Used to identify illegitimate actors
- Distrust as a primitive
  - ✓ Model Trust and Distrust as independent primitive
  - ✓ Distrust ≠ absence of trust
- Trust Conflicts
  - ✓ The presence of positive and negative authorization at the same time could generate conflicts on trust relationships
  - ✓ Computer-Aided RE automatically detects such conflicts

Sample Conflicts



Trust at social level
Distrust at individual level

Distrust at social level
(eg procedures imposing restriction on roles)
Trust at individual level



Verification Process

- Design models at both social level and individual level, independently
- Verify correctness and consistency of social level
- Map relations at social level into models at individual level
- Solve conflicts if needed
- Verify correctness and consistency of models at individual level

Monitoring

- Trust is normally necessary for delegation.
- However, there may be delegations without trust
  - ✓ the delegator is not free to decide (coercive delegation)
  - ✓ the delegator has no knowledge and no alternative (blind delegation)
- Delegation without trust may carry risk
- Monitoring as surrogate for trust
  - ✓ An actor (the *monitor*) is appointed by the delegator to monitor whether the delegatee will not misuse services and fulfill assigned obligations
  - ✓ If you do not trust somebody, just monitor his work to ensure everything happens according to expectations.

Privacy ≠ Security

- Privacy is the right of individuals to determine for themselves when, how, and to what extent information about them is communicated to others.

  Alan Westin - Professor of Law at Columbia Univ.

- Contentious
  - ✓ We cannot use Software Engineering Methodologies to address privacy, they have different objectives (we cannot use Security Methodologies either)
  - ✓ Engineering and civil liberties don't mix …

In Real Life…

- Permissions can never be as fine grained as you would need them
  - ✓ Cleaning person has key to open room, rather than just access to waste basket …;
  - ✓ … Can empty waste basket or steal papers from desk.
- Real life contracts or data submissions have *purposes* tagged to permissions
  - ✓ Special power of attorney for contracts
  - ✓ Privacy laws in the US, EU, Italy, etc.
  - ✓ You get a permission for some action/goal.
- If you breach trust (use permission for other purposes) then you can be sued, fined, etc.

Privacy Links Permission to Purpose

- Fact of Life
  - ✓ We want something done
  - ✓ We give private information (or access to it) to get it done
- If private information is used for its given purpose
  - ✓ --> Happy customer
- If private information is used for other purposes
  - ✓ Consent must be sought (eg according Law)
  - ✓ --> Unhappy or unwilling customers

Future Work

- More Sophisticated Reasoning
- Privacy Concepts
  - ✓ Build formal theory + reasoning services
  - ✓ Relations with other frameworks (eg HippoDB)
- Completing the development process for secure designs
  - ✓ Expand the model beyond early requirements
- Ongoing case studies
  - ✓ Compliance with Privacy Legislation or ISO-17799
  - ✓ John Rusnak's fraud
- This work has been partially supported by MOSTRO, SERENITY, STAMPS and SENSORIA projects