# Revani: Revising and Verifying Normative Specifications for Privacy

**Özgür Kafalı, Nirav Ajmeri, and Munindar P. Singh,** *North Carolina State University*

*Privacy remains a major challenge today, partly because it brings together social and technical considerations. Yet, current software engineering focuses only on the technical aspects. Revani understands privacy from the standpoint of sociotechnical systems.*

Our investigation of concepts and techniques to enhance privacy begins from the recognition that privacy incorporates both human and social aspects. Accordingly, we approach privacy from the perspective of sociotechnical systems (STSs), which we view as systems composed of both social (people and organizations) and technical (computers and networks) elements.[1,2] Whereas traditional engineering approaches consider the social aspects in their early phases, they exclude them from the specifications they ultimately produce, including only the technical aspects therein.

There's a natural tension between functional and privacy requirements: typically, performing a work task reveals information, and restricting information obstructs a work task. Because mechanisms that are privacy-preserving at first glance often interfere with users' work, they force a choice on users of either failing to accomplish some goal or subverting those mechanisms, thereby compromising privacy. For example, a short session timeout will either interrupt a user's flow or force the user to seek workarounds, such as storing a password in a browser. The first case is effectively a denial of service and the second risks the password being stolen. A more subtle situation arises during disasters. The guidelines from the American College of Emergency Physicians (ACEP; http://goo.gl/HXWRnH) include expanding staff capacity and relaxing privacy requirements during a disaster. A hospital could take the initiative and assign temporary credentials to outside physicians to cope with the expected load (potentially even before the event is declared a federal disaster). However, doing so inevitably creates privacy threats.

How can we address such dilemmas? A sociotechnical view doesn't provide a magic bullet, but it makes these tradeoffs explicit and thereby helps produce specifications that hit the sweet spot between functional and privacy requirements. So how can we represent the social aspects formally, and how can we create an STS and verify whether it satisfies stakeholder requirements? Our contribution to this open research problem is Revani (which stands for Revision and Verification of Normative Specifications), an approach for

1541-1672/16/$33.00 © 2016 IEEE
*Published by the IEEE Computer Society*

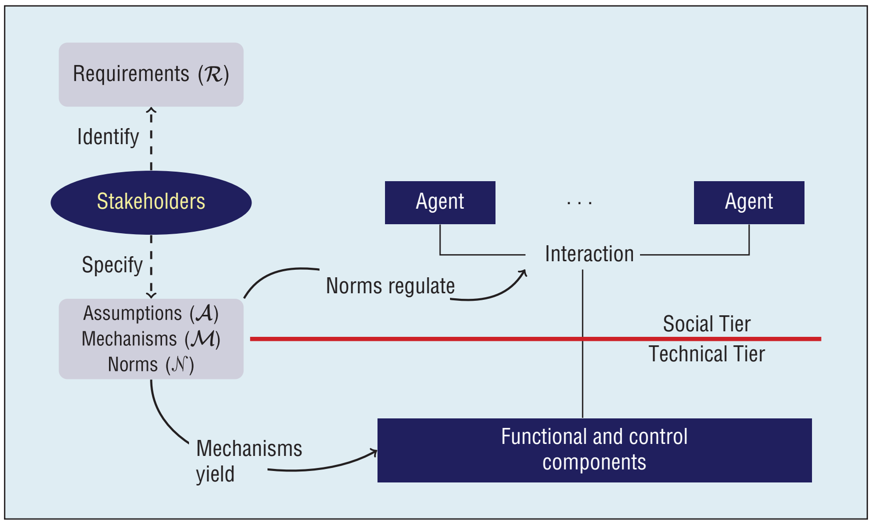

**Figure 1. Conception of a sociotechnical system (STS). The right part presents an STS as a two-tier system. The upper (social) tier includes agents representing the stakeholders, who interact with each other via the STS's technical elements. The lower (technical) tier includes software (functional and control) components that support agent interactions in the social tier. The left part of the figure shows that an STS can be specified by stakeholders based on their requirements.**

engineering STSs that promotes privacy by incorporating the social elements in a formal computational representation based on a particular view of norms, employing an approach that uses design patterns to create STS specifications that satisfy stakeholder requirements, and developing a revision tool based on temporal logic model checking that facilitates producing correct specifications.

To demonstrate our approach, we adopt as a running example a little-studied part of the HIPAA (Health Insurance Portability and Accountability Act) law that focuses on the disclosure of patient information during disasters (www.hhs.gov/ocr/privacy/hipaa/understanding/special/emergency). In our scenario, a physician logs into the emergency department's computer, accesses and reviews a patient's electronic health record (EHR), and logs out of the computer. Ordinarily, the physician can't disclose the patient's protected health information (PHI) without consent, but during a national disaster, he or she is allowed to share this information with the patient's family without that consent. During a disaster, other emergency physicians (including those recruited from other hospitals to lend a helping hand) can access this patient data by authenticating on a computer in the emergency department.

Two established bodies of work complement our contribution. Traditional technical solutions such as access control can constrain who has access to what information, and modern access control models handle exceptional conditions well. For example, Rumpole[3] accommodates the idea that patient consent is waived during a disaster. Usable privacy and privacy engineering approaches[4,5] seek to improve user interfaces for authentication, policy configuration, photo sharing, and so on by tackling the human aspects of privacy and addressing biases in cognition and attention. They seek to "compile out" the human elements by producing better technical elements. However, unlike Revani, they don't incorporate computational models of privacy's social aspects.

In essence, both access control and usable privacy focus on an STS's technical elements and fail to encode any knowledge of its social elements. These approaches don't identify autonomous parties (agents) and what they're accountable for, even though it's precisely these autonomous parties whose requirements we're serving and whose interactions could lead to privacy violations. We model threats invisible to a technical approach, such as when a physician fails to log out and inadvertently enables an unauthorized person to access sensitive data or when a hospital assigns credentials to an outside physician. Access control doesn't capture what happens when an outside physician accesses patient data or a staff physician discloses it. Accordingly, our design process seeks to answer two important research questions: How can we design an STS that satisfies the given requirements? Answer: by first constructing a formal model of norms that determines STS enactments and then applying our design patterns to come up with an STS specification that satisfies the stated requirements. How can we determine which requirements are affected when a norm is violated? Answer: by first removing the assumption that agents are compliant with norms and then identifying which requirements (stated as verification properties) aren't satisfied when there's a norm violation. The prospect of agents violating norms is real in open systems with autonomous agents.

## Sociotechnical Systems

Figure 1 illustrates our conception of an STS and highlights two important points: the identification of coexisting social and technical tiers and the emphasis on norms and regulation. The right part of the figure presents an STS as a two-tier system. The upper (social) tier includes agents representing the stakeholders, who interact with each other via the STS's technical elements. The lower (technical) tier includes software (functional and control) components that support agent interactions in the social tier. The left part of the figure shows that an STS can be specified by stakeholders based on their requirements. An STS specification consists of norms that regulate interactions of its social elements and mechanisms that are realized in its nonautonomous (functional and control) components. For simplicity, we include the domain assumptions in the specification along with the norms and mechanisms.

Of the possible conceptions of STSs,[6] we confine ours to the present because it's adequate for demonstrating how we might address privacy requirements in a formal, norm-based manner. One benefit of our approach is the flexibility it accords agents via social elements. Some requirements can and should be realized through technical means—for example, an access control mechanism can check for appropriate consent, but such a requirement could be overridden during a disaster. An improved access control mechanism could handle this by dispensing with consent during a disaster. However, some requirements are inherently social—the fact that a hospital could legitimately dispense with patient consent during a disaster isn't captured in the technical tier because it's a property of the social tier, specifically, will physicians only access relevant patient data required for treatment? Similarly, a hospital could grant access to patient data to outside physicians during a disaster, which affects the technical tier even though the reasoning about its correctness lies in the social tier. In the same spirit, the interactions of outside physicians with hospital staff are regulated in the social tier, say, with hospital staff prohibited from discussing nonemergency patients with outside physicians. But what recourse do patients have when the prohibition is violated? Addressing these challenges requires a computational framework that synthesizes both technical and social aspects.

## Norms and Accountability

At the heart of our conception of an STS is the notion of social norms. Prior formulations treat norms as expected social properties,[7,8] usually enforced through (positive or negative) social sanctions.[9] Some approaches define to whom the norm applies but don't indicate a counterparty. Monitoring entities (centralized or distributed) are assumed to verify the compliance of agents to norms.

In contrast, we understand a norm as a conditional, directed relationship that indicates who's accountable to whom.[2] Our notion of norms is compatible with deontic concepts such as permissions and obligations introduced by von Wright's deontic logic.[10] We consider three types of norms: authorization, commitment, and prohibition. For brevity, we introduce Revani's syntax and semantics through examples: `EHR` is a proposition meaning that the physician accesses a patient's EHR, `EMERGENCY` is a proposition meaning that an emergency is declared in the hospital, `LOGGED_IN` is a proposition meaning that the physician logs in to a computer, and `PHI_DISCLOSED` is a proposition meaning the patient's PHI is disclosed.

Extrapolating further,

- *A*(*PHY*, *HOS*, `EMERGENCY`, `EHR`): a physician PHY is authorized (*A*) by the hospital *HOS* to access a patient's `EHR` in an emergency. Here, the object (*HOS*) is accountable to the subject (*PHY*). The hospital must ensure that a physician has access to the EHR when the authorization is detached (that is, `EMERGENCY` is true).
- *C*(*PHY*, *HOS*, `EHR`, ¬`LOGGED_IN`): a physician *PHY* is committed (*C*) to the hospital *HOS* to logging out of the computer (¬`LOGGED_IN`) after he or she accesses the patient's `EHR`. The physician is accountable to the hospital for this commitment.
- *P*(*PHY*, *HOS*, `true`, `PHI_DISCLOSED`): a physician *PHY* is prohibited (*P*) by the hospital *HOS* from disclosing a patient's *PHI* to others (`PHI_DISCLOSED`). This prohibition is unconditional because its condition is true. The physician is accountable to the hospital for this prohibition.

We specify an STS's social tier via norms that provide a standard of correctness with which to judge an implementation, such as an access control mechanism. For example, a traditional approach might realize a prohibition to read confidential information via access control—a prohibited party isn't allowed access. Other prohibitions, such as the one above that's against disclosing the patient's PHI, aren't represented at all because they can't be tackled purely at the technical tier. Whereas in traditional approaches, there's no representation other than the technical tier, here, norms capture the social tier independently of access control. A benefit of doing this is that we can reason about the social tier, both in understanding where the technical tier supports it and where it does not.

## Assumptions and Mechanisms

Assumptions characterize the STS's operating environment and describe what can or can't happen. An assumption is a pair ⟨`Head`, `Body`⟩, written `Head`←`Body` as an inference rule. For example, ¬`LOGGED_IN`← `POWER_FAILURE` means that it isn't possible to be logged in to a computer during a power failure. The correct working of Revani depends on these assumptions being consistent.

Agent actions are supported by underlying mechanisms. An example action, performed by the physician, is logging in to the emergency department computer. Mechanisms can impose enabling conditions on the actions—

for example, providing a correct password is the enabling condition for logging in to a computer.

We write a mechanism as `M(ENABLER, ADD, DELETE)`. When a mechanism is enabled (that is, `ENABLER` is true), its effect can take place. The effect consists of a set of atomic propositions to be added (`ADD list`) and a set of atomic propositions to be deleted (`DELETE list`). Here, `M(PASSWORD, {LOGGED_IN}, {})` describes the mechanism for logging in to a computer. Some mechanisms are always enabled.

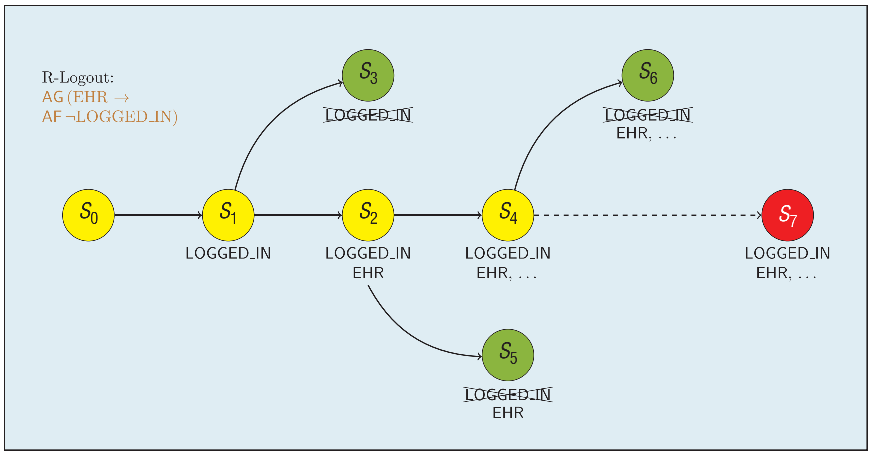

**Figure 2. Verification of computation tree logic formulas. R-Logout, the requirement being verified, states that if the physician accesses the electronic health record (EHR), he or she will eventually log out of the computer. Some branches (ending in $S_3$, $S_5$, and $S_6$) satisfy R-Logout. However, the existence of a counterexample (the branch leading to $S_7$) means this model violates R-Logout.**

## Requirements Engineering for STS

Researchers[11] formulate traditional requirements engineering (RE) as

`Assumptions, Mechanisms ⊢ Requirements.` (1)

The traditional RE problem is to determine (a specification of) mechanisms and domain assumptions such that any software implementation that follows the mechanism specification satisfies the given requirements, provided the assumptions hold. This formulation omits the social elements.

We introduce norms as an additional component of the specification to extend it to accommodate STSs:

`Assumptions, Mechanisms, Norms ⊢ Requirements.` (2)

The sociotechnical RE problem is to find mechanisms and norms such that if the software implementation satisfies the mechanisms and the parties satisfy their norms, then provided the assumptions hold, the requirements are satisfied. The simplicity of the modification belies its subtlety. Because we introduce autonomous parties, they can adopt local mechanisms for their individual decision making. Moreover, the satisfaction of norms is nontrivial because norms can be overridden or sanctions can be applied.

## Computational Representation

Broadly speaking, Revani takes two inputs, an STS specification and stakeholder requirements, and produces a binary output, which indicates whether the specification satisfies the requirements.

To perform formal verification, we adopt model checking with branching-time temporal logic.[12] This paradigm posits a tree-like formal model based on the specification that's generated according to possible enactments of norms with regard to agents' actions and domain events. All enactments begin at the root; events occur serially on a branch, and each branch corresponds to a distinct possible enactment.

Figure 2 shows an example model. Enactments start from an initial state ($S_0$). Then, the physician logs in to the computer, which initiates a transition to $S_1$. The physician can view a patient's EHR ($S_2$); he or she can log out in any state, creating alternative branches ($S_3$, $S_5$, and $S_6$). If the physician performs other actions (such as passing through $S_4$) but never logs out, the branch leads to $S_7$.

## Specification and Requirements

Revani's specification is quite straightforward. It enumerates STS roles (here, `PHY` and `HOS`), domain propositions, and actions. Roles are design-time placeholders for agents—for example, `PHY` is instantiated with physician names at runtime. A domain event such as `cut_power` brings about the proposition `POWER_FAILURE`. Similarly, an agent action such as `log_in` brings about the proposition `LOGGED_IN`.

We express each stakeholder requirement as a formula of computation tree logic (CTL),[12] a branching-time logic based on a tree model, as in Figure 2. CTL enhances ordinary propositional logic with two temporal elements. A branch quantifier, A or E, respectively, indicates whether we're talking about all or some branches emanating from the current point. A linear temporal operator considers points on one branch; for a proposition $p$ (`LOGGED_IN`), $Fp$ means that $p$ occurs eventually on the current branch, and $Gp$ means that $p$ always occurs on the current branch.

Consider the following healthcare requirements:

- *R-Disclose*, which states the patient's PHI must never be disclosed on any branch. In CTL, this is AG(¬PHI_DISCLOSED). At each point on each branch PHI must not be disclosed.

- *R-Access*, which states that physicians can access an EHR. In CTL, this is EF (EHR). There must be a branch where EHR is eventually accessed.
- *R-Logout*, which states open sessions must be closed after reviewing an EHR. In CTL, this is AG(EHR→AF ¬LOGGED_IN). At any point when the EHR is accessed, the physician must eventually log out on all branches.
- *R-Share*, which states that in case of a national disaster, physicians must be allowed to share a patient's PHI with family members. In CTL, this is AG(DISASTER→EF PHI_SHARED). At any point when a disaster is declared, there must be a branch where PHI is eventually shared.

We assume that requirements are explicitly stated by the stakeholders and are mutually consistent. We don't address consistency checking of requirements.

## Verification

A model checker verifies whether the specification satisfies each requirement. If so, the specification is correct; otherwise, it's not. We adopt NuSMV, a model checker for CTL that provides a language for specifying finite state models (http://nusmv.fbk.eu). A NuSMV specification describes a set of variables and how these variables progress according to the possible enactments of the modeled system. Then, NuSMV verifies desired properties of the system expressed in CTL.

Consider the CTL formula in Figure 2 for R-Logout. The logical implication symbol means that whenever the antecedent of the formula holds (that is, whenever EHR is accessed or AG EHR), the consequent must hold (the physician must eventually log out of the computer or AG AF ¬LOGGED_IN). To determine whether this formula is satisfied at the tree root, we can examine each branch in turn. The branch ending in $S_3$ is acceptable because the EHR is never accessed on it; the branches ending in $S_5$ and $S_6$ are acceptable because a log out follows an access to the EHR. But R-Logout fails at the root because there's no log out ($S_7$) despite the EHR being accessed.

Revani supports two verification scenarios:

- *Restrict an STS specification to include only correct enactments of norms.* For example, the correct enactment of a commitment C(PHY, HOS, EHR, ¬LOGGED_IN) would rule out state $S_7$. Ruling out such violating enactments enables us to understand what norms are necessary to achieve desired behavior in an STS. For example, the above commitment will satisfy R-Logout.
- *Include additional enactments (compliant and violating) in an STS specification and investigate what happens in a norm violation.* For example, verification of R-Logout in NuSMV would lead to a counterexample where the above commitment is never satisfied.

## Design Process

Determining that an STS fails its requirements isn't sufficient. We want to revise a specification so as to satisfy (possibly changing) requirements. To this end, we adopt an iterative design process that begins from a specification and revises the specification until it satisfies all requirements. Our process is centered on design patterns that exploit logical relationships supported by formalization of norms.

### Norm Strength

Our logical model enables determining which norm entails another (the ⊢ symbol denotes logical consequence). We formally define norm strength for each norm type as follows. Subscripts ($i$, $j$, 1, 2) represent instances of norms, such as the following.

***Authorization strength.*** $A_i(SBJ, OBJ, ANT_i, CON_i)$ is stronger than $A_j(SBJ, OBJ, ANT_j, CON_j)$, $A_i \gg A_j$, if and only if $ANT_j \vdash ANT_i$ and $CON_j \vdash CON_i$. Consider the following authorizations:
$A_1(PHY, HOS,$ CONSENT ∨ EMERGENCY, EHR ∨ ASK_PARENTS),
$A_2(PHY, HOS,$ CONSENT, EHR).
$A_1 \gg A_2$ because CONSENT ⊢ CONSENT ∨ EMERGENCY and EHR ⊢ EHR ∨ ASK_PARENTS.

***Commitment strength.*** $C_i(SBJ, OBJ, ANT_i, CON_i)$ is stronger than $C_j(SBJ, OBJ, ANT_j, CON_j)$, $C_i \gg C_j$, if and only if $ANT_j \vdash ANT_i$ and $CON_i \vdash CON_j$. Consider the following commitments:
$C_1(PHY, HOS, true,$ OPERATION ∧ CLINIC),
$C_2(PHY, HOS,$ EMERGENCY, OPERATION).
$C_1 \gg C_2$ because EMERGENCY ⊢ true and OPERATION ∧ CLINIC ⊢ OPERATION.

***Prohibition strength.*** $P_i(SBJ, OBJ, ANT_i, CON_i)$ is stronger than $P_j(SBJ, OBJ, ANT_j, CON_j)$, $P_i \gg P_j$, if and only if $ANT_j \vdash ANT_i$ and $CON_j \vdash CON_i$. Consider the following prohibitions:
$P_1(PHY, HOS, true,$ PHI_SHARED ∨ PHI_DISCLOSED),
$P_2(PHY, HOS,$ ¬EPIDEMIC, PHI_DISCLOSED).
$P_1 \gg P_2$ because ¬EPIDEMIC ⊢ true and PHI_DISCLOSED ⊢ PHI_SHARED ∨ PHI_DISCLOSED.

### Design Patterns

In each step of our iterative design process, we begin from a specification and systematically revise it into another specification. The benefit of reasoning about norm strength is that it ensures we can revise a specification into one that enhances functionality (by adding beneficial enactments) or privacy (by curtailing pernicious enactments). We capture revisions as design patterns.

The following relaxation patterns liberalize an STS and enable additional enactments:

- Expansion strengthens a given authorization specification $A_i$ by replacing it with $A_j$, where $A_j >> A_i$. Suppose a physician is authorized to access a minor patient's EHR. We can strengthen the authorization so that the physician is also authorized to talk to the minor's parents.
- Release of liability weakens a given commitment specification $C_i$ by replacing it with $C_j$, where $C_i >> C_j$. Suppose a physician is committed to the hospital to operating on patients as well as undertaking clinic duty. We can weaken the commitment so that the physician is committed only to operating on patients.
- Accessibility weakens a given prohibition specification $P_i$ by replacing it with $P_j$, where $P_i >> P_j$. Suppose a physician is prohibited by the hospital from sharing a patient's PHI with colleagues or publishing it online. We can weaken the prohibition so that the physician is prohibited from publishing a patient's PHI online but not from sharing it with a colleague.

Although the above patterns enhance functionality, they can yield erroneous specifications, e.g., if a potential norm violation that results from the added functionality isn't properly handled. The following amendment patterns address such cases:

- Responsibility specifies a complementary commitment to capture that the subject doesn't misuse the intended functionality provided by a relaxation pattern. Formally, it replaces $A_i(SBJ, OBJ, ANT_i, CON_i)$ with $A_j(SBJ, OBJ, ANT_j, CON_j)$, where $A_j >> A_i$, and adds $C_k(SBJ_k, OBJ_k, CON_j, CON_k)$. For example, extending the session duration (via the expansion pattern) increases the privacy risk if the physician forgets to log out. We can have the physician commit to logging out upon completing the task.
- Limitation specifies a complementary prohibition to provide compensation—that is, the additional functionality given by the relaxation pattern is bounded by the limits of the new prohibition. Formally, it replaces $P_i(SBJ, OBJ, ANT_i, CON_i)$ with $P_j(SBJ, OBJ, ANT_j, CON_j)$, where $P_i >> P_j$, and adds $P_k(SBJ_k, OBJ_k, CON_j, CON_k)$.

Note that the subjects ($SBJ_k$) and objects ($OBJ_k$) of the additional norms for the amendment patterns aren't necessarily the same as the original norms. For example, allowing the physician to share a patient's PHI with colleagues (via the accessibility pattern) increases the risk that the PHI is disclosed to parties that are prohibited from accessing the PHI. We can prohibit the physician's colleague (a new party as the subject of the norm) from publishing the PHI online.

The following pattern revises a mechanism to relax the enabling condition for an action: enabler replaces mechanism $M_i(ENABLER_i, ADD, DELETE)$ with $M_j(ENABLER_j, ADD, DELETE)$ if $ENABLER_i \vdash ENABLER_j$. For example, the EHR software for regular medical practice implements a mechanism that restricts access to an EHR without consent. We can relax the enabling condition for this mechanism to allow physicians to access any patient's EHR in an emergency.

## Demonstration and Evaluation

We built a tool for Revani to suggest revisions for a given STS specification. To demonstrate that the tool can come up with a norm specification compliant with the requirements for our scenario, we begin with an initial specification that doesn't satisfy some requirements. This initial specification reflects common practice in emergency medicine before the HIPAA privacy rule was revised in 2003 (www.hhs.gov/ocr/privacy/hipaa/understanding/summary).

Figure 3 shows the application of patterns through a series of specifications. Our design process is sound—that is, when our tool suggests a revision with respect to a requirement, the revised specification satisfies the requirement. However, our design process isn't complete in that our tool might not always compute a revised specification to satisfy a requirement. Figure 3 presents one solution among possible alternative revisions computed by our tool. Let's review each step:

- The initial specification satisfies R-Disclose because disclosing patients' PHI is prohibited. However, this specification is inflexible in disaster situations and fails R-Share. Moreover, access to the EHR isn't authorized without a patient's consent, which leaves R-Access unsatisfied. R-Logout fails as well, because the initial specification doesn't regulate computer usage in the emergency department.
- The authorization is substituted with a stronger one, which allows an alternative way for the physician to access a patient's EHR via the emergency department computer.
- An additional commitment is specified to improve privacy for patients' PHIs.

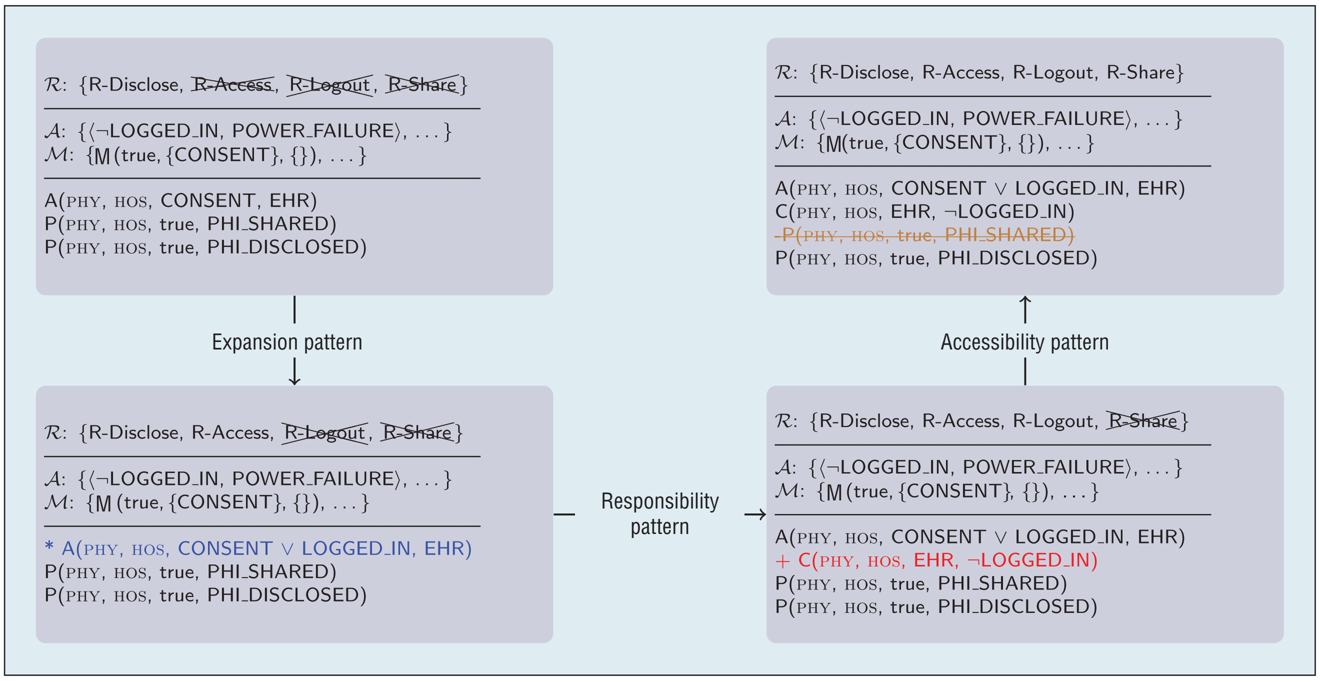
$\mathcal{R}$: {R-Disclose, ~~R-Access~~, ~~R-Logout~~, ~~R-Share~~}
$\mathcal{A}$: {⟨¬LOGGED_IN, POWER_FAILURE⟩, . . . }
$\mathcal{M}$: {M (true, {CONSENT}, {}), . . . }
A(PHY, HOS, CONSENT, EHR)
P(PHY, HOS, true, PHI_SHARED)
P(PHY, HOS, true, PHI_DISCLOSED)
Expansion pattern
$\mathcal{R}$: {R-Disclose, R-Access, ~~R-Logout~~, ~~R-Share~~}
$\mathcal{A}$: {⟨¬LOGGED_IN, POWER_FAILURE⟩, . . . }
$\mathcal{M}$: {M (true, {CONSENT}, {}), . . . }
* A(PHY, HOS, CONSENT ∨ LOGGED_IN, EHR)
P(PHY, HOS, true, PHI_SHARED)
P(PHY, HOS, true, PHI_DISCLOSED)
Responsibility pattern
$\mathcal{R}$: {R-Disclose, R-Access, R-Logout, ~~R-Share~~}
$\mathcal{A}$: {⟨¬LOGGED_IN, POWER_FAILURE⟩, . . . }
$\mathcal{M}$: {M (true, {CONSENT}, {}), . . . }
A(PHY, HOS, CONSENT ∨ LOGGED_IN, EHR)
+ C(PHY, HOS, EHR, ¬LOGGED_IN)
P(PHY, HOS, true, PHI_SHARED)
P(PHY, HOS, true, PHI_DISCLOSED)
Accessibility pattern
$\mathcal{R}$: {R-Disclose, R-Access, R-Logout, R-Share}
$\mathcal{A}$: {⟨¬LOGGED_IN, POWER_FAILURE⟩, . . . }
$\mathcal{M}$: {M(true, {CONSENT}, {}), . . . }
A(PHY, HOS, CONSENT ∨ LOGGED_IN, EHR)
C(PHY, HOS, EHR, ¬LOGGED_IN)
~~P(PHY, HOS, true, PHI_SHARED)~~
P(PHY, HOS, true, PHI_DISCLOSED)

**Figure 3. Applying the design patterns using the revision tool. We begin with an initial specification (top left box). Each box shows the revised specification after the application of a pattern. Crossed out requirements aren't satisfied by the corresponding specification. Some norms replace existing norms (* A(...)), some are added (+ C(...)), and some are removed ~~P(...)~~.**

The physician must log out of the computer after reviewing the EHR.

- The two prohibitions are relaxed into a weaker prohibition, which prohibits the physician only from disclosing patients' PHIs to outsiders.

Revani differs from other formal verification models because it incorporates the social dimension and thereby provides a computational basis to regulate interactions among agents.

Norms have been adopted for capturing and verifying privacy requirements in the contextual integrity framework.[13] Moreover, emerging software engineering approaches incorporate deontic concepts.[14] However, these approaches either focus on control components or treat norms as hard constraints, thus lacking the sociotechnical underpinnings of Revani, which are essential for computationally handling the human and social aspects of privacy.

Recent works from the multiagent systems literature[7,8,15] capture the normative dimension for STSs, but they lack Revani's formal design methodology. One approach[1] proposes a formalization for STS requirements engineering, but this formalization is limited to commitments; it doesn't formalize STS mechanisms or support verification.

Revani opens up several directions for future work. Of these, developing ways to measure the improvement a revision pattern provides to a given specification (such as the distance from an optimal specification) are particularly important for facilitating decision making by stakeholders when creating STS specifications. ■

## Acknowledgments

This research is supported by the US Department of Defense under the Science of Security Lablet grant. We thank Rahul Pandita, Jessica Staddon, Christopher Theisen, and the anonymous reviewers for their helpful comments on previous versions of this article.

## THE AUTHORS

**Özgür Kafalı** is a postdoctoral researcher in computer science at North Carolina State University. His research interests include multiagent systems and computational logic. Kafalı has a PhD in computer engineering from Bogazici University. Contact him at rkafali@ncsu.edu.

**Nirav Ajmeri** is a PhD student in computer science at North Carolina State University. His research interests include software engineering and multiagent systems, with a focus on security and privacy. Contact him at najmeri@ncsu.edu.

**Munindar P. Singh** is a professor in computer science and a co-director of the Science of Security Lablet at North Carolina State University. His research interests include the engineering and governance of sociotechnical systems. Singh is an IEEE Fellow, a former editor in chief of IEEE Internet Computing, and the current editor in chief of ACM Transactions on Internet Technology. Contact him at singh@ncsu.edu.

## References

1. A.K. Chopra et al., "Protos: Foundations for Engineering Innovative Sociotechnical Systems," *Proc. 18th IEEE Int'l Requirements Eng. Conf.*, 2014, pp. 53–62.
2. M.P. Singh, "Norms as a Basis for Governing Sociotechnical Systems," *ACM Trans. Intelligent Systems and Technology*, vol. 5, no. 1, 2013, pp. 21:1–21:23.
3. S. Marinovic, N. Dulay, and M. Sloman, "Rumpole: An Introspective Break-Glass Access Control Language," *ACM Trans. Information and System Security*, vol. 17, no. 1, 2014, pp. 2:1–2:31.
4. S. Spiekermann and L.F. Cranor, "Engineering Privacy," *IEEE Trans. Software Eng.*, vol. 35, no. 1, 2009, pp. 67–82.
5. A. Adams and M.A. Sasse, "Users Are Not the Enemy," *Comm. ACM*, vol. 42, no. 12, 1999, pp. 40–46.
6. I. Sommerville et al., "Large-Scale Complex IT Systems," *Comm. ACM*, vol. 55, no. 7, 2012, pp. 71–77.
7. N. Criado, E. Argente, and V. Botti, "Open Issues for Normative Multi-agent Systems," *AI Comm.*, vol. 24, no. 3, 2011, pp. 233–264.
8. F. Dechesne et al., "No Smoking Here: Values, Norms and Culture in Multi-agent Systems," *Artificial Intelligence and Law*, vol. 21, no. 1, 2013, pp. 79–107.
9. L.G. Nardin et al., "Classifying Sanctions and Designing a Conceptual Sanctioning Process Model for Socio-technical Systems," *Knowledge Eng. Rev.*, vol. 31, no. 3, 2016, pp. 142–166.
10. G.H. von Wright, "Deontic Logic: A Personal View," *Ratio Juris*, vol. 12, no. 1, 1999, pp. 26–38.
11. P. Zave and M. Jackson, "Four Dark Corners of Requirements Engineering," *ACM Trans. Software Eng. and Methodology*, vol. 6, no. 1, 1997, pp. 1–30.
12. E.M. Clarke, O. Grumberg, and D.A. Peled, *Model Checking*, MIT Press, 1999.
13. A. Barth et al., "Privacy and Contextual Integrity: Framework and Applications," *Proc. IEEE Symp. Security and Privacy*, 2006, pp. 184–198.
14. E. Letier and W. Heaven, "Requirements Modelling by Synthesis of Deontic Input-Output Automata," *Proc. 35th Int'l Conf. Software Eng.*, 2013, pp. 592–601.
15. N. Alechina, M. Dastani, and B. Logan, "Reasoning about Normative Update," *Proc. 23rd Int'l Joint Conf. Artificial Intelligence*, 2013, pp. 20–26.