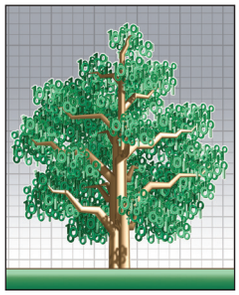
Editor: Munindar P. Singh • *m.singh@ieee.org*



Engineering Privacy in Social Applications

Pradeep K. Murukannaiah, Nirav Ajmeri, and Munindar P. Singh • *North Carolina State University*

Experience with a social application depends crucially upon how it supports or interferes with the users' social expectations. Because privacy is central to the user's experience, the authors introduce Danio, a methodology based on modeling users' expectations in various contexts.

William Prosser, in his classic paper, describes how privacy involves diverse aspects centered on the idea of being let alone (p. 389).[1] We view privacy as crucial to the user experience (UX) in a sociotechnical setting, wherein users engage socially through information technology. In previous work, we addressed privacy by limiting information disclosure to location-based (and not necessarily social) applications by computing context at an abstract level that enhances usability but hides details.[2]

Here, we approach privacy from the standpoint of engineering social applications, wherein interactions among users are central and thus privacy matters for more than just disclosure. Specifically, we investigate how to develop applications that deal with two foundational aspects of privacy.[1] Intrusion into someone's solitude, which originally meant physical intrusion into a person's space, we also take to include making a noise or otherwise interrupting someone's life. Disapprobation of someone's peers would mean loss of face, which can arise not only with a person's inner circle but also with strangers looking askance. Avoiding intrusion and disapprobation is overshadowed in computing research by concerns of information leakage — arguably, information leakage involves confidentiality more than privacy per se.

A social application caters to multiple users: primary users, who directly interact with it, and secondary users, who might not directly interact with the application but are affected by it. The lowly ringer manager on a cell phone is a social application: its primary user is the phone's owner and the secondary users are callers and those within earshot. The ringer manager helps the owner set a ringer mode (loud, silent, or vibrate) for incoming phone calls. A rigid design yields poor privacy and experience: The phone might ring loudly when the owner is in an important meeting (causing a nuisance) or stay silent even when the owner's spouse calls in an emergency (losing value).

Traditionally, UX design concentrates on primary users and disregards secondary users.[3] This attitude can lead to suboptimal experiences for both primary and secondary users, specifically, because privacy presupposes interaction between users.

Intrusion is a prominent aspect of privacy in the ringer scenario: Does the caller intrude upon the callee and does the callee intrude upon people nearby (by taking a call or by letting a phone ring)? The UX for all concerned parties depends upon whether the phone rings: the caller could be stymied by a phone set on silent and the privacy of the other users might be violated otherwise. Also, improper ringer settings expose another privacy risk — that of disapprobation, causing the owner embarrassment. Imagine if your phone went off during a classical concert!

Social Expectations

Producing privacy-enhancing controls is nontrivial. Setting a fixed ringer mode for all incoming calls (as is common today) ignores secondary users, but asks the owner to anticipate contexts and secondary users. But setting appropriate ringer policies is too complex to be viable. Accordingly, we adopt the idea of modeling social interactions.

 Published by the IEEE Computer Society 1089-7801/16/$33.00 © 2016 IEEE

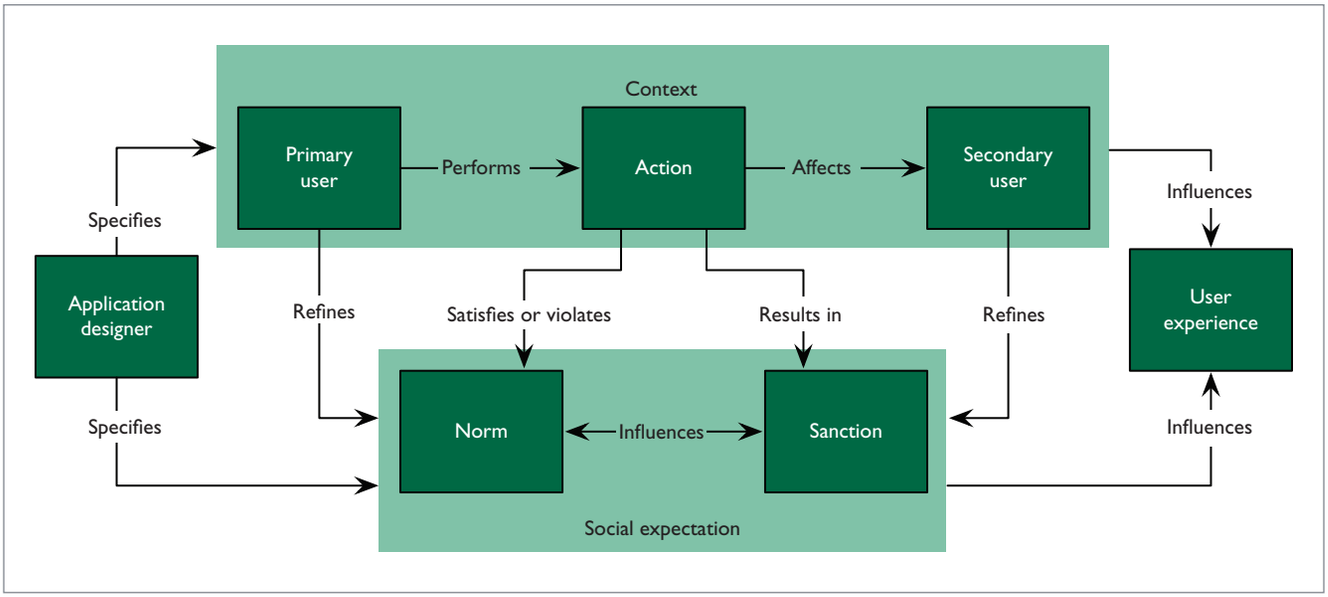


Figure 1. A conceptual model showing how a primary user's action within a certain context affects the secondary user, and how contexts and expectations influence the user experience. A traditional design emphasizes only the primary users' actions and contexts, but this model additionally captures how a primary user's action might affect another (secondary) user.

Katja Battarbee[4] motivates *co-experience* as a seamless blend of UX and social interactions.

However, social interaction is a loose concept. We propose instead to capture users' *social expectations* of others as central to delivering an optimal experience. For example, the ringer manager's UX depends on whether the phone owner meets their spouse's expectation by answering important calls, which in turn might depend on whether the ringer manager lets the phone ring. Thus, we seek to systematically incorporate social expectations in the UX as geared toward privacy.

Broadly, looking beyond privacy, social expectations and their influences are abundant in real-life interactions. Expectations arise both explicitly (as in text-based interactions) and implicitly (as via gestures). Yet, current UX techniques largely disregard expectations. How can we computationally represent and reason about expectations?

Sarah Spiekermann and Lorrie Faith Cranor[5] describe a "joint sphere" of privacy responsibilities involving data senders and recipients. Social expectations fit into this sphere. Spiekermann and Cranor identify the key engineering challenge in this sphere as providing individuals' control over access to themselves. In contrast, we argue that a key challenge in this sphere, perhaps more important than control, is to engineer solutions where both senders and recipients are accountable to each other and thus vested in enhancing each other's privacy.

We adopt two interrelated computational abstractions to capture social expectations. First, a *norm* characterizes a user's expected ("normal") behavior from the perspective of another user.[6] Second, a *sanction* characterizes a user's response to another user's satisfaction or violation of a norm.[7,8] Norms and sanctions arising in an application's context can be computationally represented and reasoned about.

Figure 1 captures a model in which a user's actions in a usage context influence social expectations, and both contexts and expectations influence the user experience. The application designer specifies both contexts and social expectations. This model emphasizes *contextual design* — an important part of the well-known user-centered design process. Whereas traditional design emphasizes the primary user's actions and contexts, this model additionally captures how a primary user's action might affect another (hence, secondary) user. A consequence of sociality is that the context helps determine whether a user is primary or secondary.

The lower part of this model captures social expectations. A primary user's action, within a context, might satisfy or violate a norm directed toward a secondary user, leading the latter to sanction (reward or punish) the primary user.

Based on this information, we understand — in a manner largely unique to our approach — that a norm is a *directed normative relationship*. (For the remainder of this article, we use "norm" in this sense.) A norm helps capture application requirements in terms of what one stakeholder expects from another.[6] A norm is directed from a subject to an object and is constructed as a conditional relationship involving an antecedent

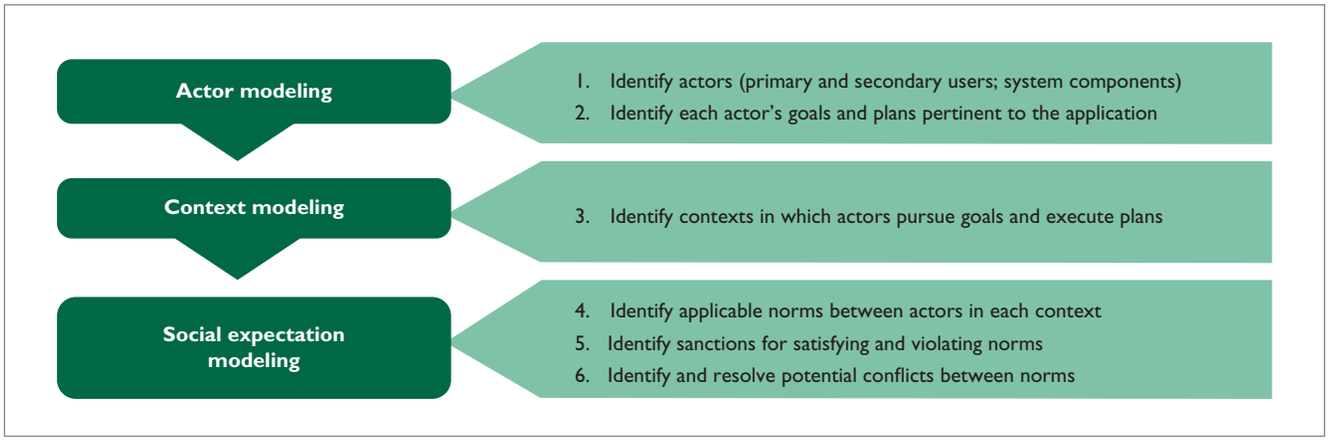


Figure 2. Danio summarized. The methodology helps systematically incorporate social expectations into application design.

(which brings the norm in force) and a consequent (which brings the norm to satisfaction). This representation yields clarity on who's accountable to whom. A norm follows this template:

N(SUBJECT, OBJECT, *antecedent*, *consequent*).

Each stakeholder is *autonomous*, meaning it can violate any norm. However, an application — and broadly speaking, society — operates under the assumption that most people respect these norms. How can we coordinate actions toward a predefined or an emergent social order? A sanction[7,8] specifies the consequences its subject faces from its object for satisfying or violating a norm, thereby promoting compliance. We write a sanction as

S(SUBJECT, OBJECT, *antecedent*, *consequent*).

A *positive* sanction rewards or encourages compliance and a *negative* sanction penalizes or discourages violation. Sanction types include the following: *autonomic*, where the consequence of the norm violated or satisfied is in itself a sanction; *material*, often financial; *social*, essentially, affecting a reputation; and *psychological*, such as guilt.[8]

Understanding an application in terms of social expectations yields key payoffs in terms of privacy.

- *Personalization.* Decentralized enactment facilitates modeling software functionality — independently of implementation — that respects users' (subjective) privacy.
- *Disclosure and control.* Explicit social expectations advise a user on precisely what information to provide others, thereby avoiding unnecessary disclosure.
- *Accountability.* Each norm determines who's accountable to whom, in what context, and what the concomitant sanctions are, thereby helping to operationalize the expectations to promote privacy.

Our approach presumes representing context and expectations. At one extreme, a designer could produce a complete specification; at the other, contexts and expectations could emerge at runtime through active usage. We adopt a pragmatic middle-ground solution: the designer formulates an incomplete specification, which users refine as they interact.

Danio

Keeping all of these criteria in mind, here we propose a software engineering methodology, called Danio (after the fish), for systematically incorporating social expectations into application design by extending Tropos[9] and Xipho.[10] As Figure 2 shows, Danio's key phases are modeling actors, contexts, and expectations.

Modeling Actors

Actor modeling identifies prospective users and their requirements. Danio adopts the following modeling constructs from Tropos.[9]

An actor is a user role or software agent.

A goal is an actor's strategic interest that the application would serve. Goals can decompose into subgoals.

A plan abstracts actions to satisfy a goal.

In the ringer scenario, the actors are the CALLEE (owner), CALLER (owner's spouse), CALLEE'S NEIGHBOR (meeting participants), and RINGER (the software). The CALLEE's goals include *don't disturb neighbors*. The RINGER's plans include *set silent mode*.

Modeling Context

We identify contexts in which actors act and interact. However, context is an all-encompassing notion. Which contexts are relevant to a given application?

We understand context as inherently related to the actors' goals and

plans,[10] which provides a systematic basis for eliciting relevant contexts. Specifically, situations such as the following provide grounds for eliciting contexts.

Conflicting goals. In this scenario, an actor can't satisfy all of the goals. For example, the CALLEE's goals *answer important calls* and *don't disturb neighbors* conflict. This choice might depend upon contextual elements such as locale as well as the CALLEE's *relationships* with the CALLER and NEIGHBOR.

Multiple plans to satisfy the same goal. For example, the RINGER's plans *set silent mode* and *set vibrate mode* can each satisfy the goal *don't disturb neighbors*. The RINGER might choose at most one of these plans at runtime. Its choice could depend upon the CALLEE's ambiance.

Such scenarios help tailor the generic context model to an application-specific model. For example, the RINGER's context model can include the CALLEE's *relationships* with the CALLER and NEIGHBOR. Some aspects of context are user-specific and can be elicited from users at runtime via a contextual middleware.[2]

Modeling Social Expectations

We model social expectations between actors. Whereas Tropos and Xipho model broad-brush dependencies between actors, we derive a detailed specification of expectations in terms of norms and sanctions involving actors, actions, and contexts.

Norms. For each actor, we identify norms where the actor is the SUBJECT and another actor is the OBJECT. The context in which the norm applies can be captured in its *antecedent* and the expected behavior in its *consequent*.

Sanctions. For each norm, we identify the sanctions that apply when it's satisfied or violated. The sanction's SUBJECT and OBJECT are the corresponding norm's OBJECT and SUBJECT, respectively. Its *antecedent* captures the status of the norm and its *consequent* the sanctioning action.

The following are examples of norms and sanctions in our example.

- The CALLEE is committed to his spouse for answering the spouse's calls:
  C_1 (CALLER, CALLEE, *relationship = spouse* $\wedge$ *call, answer call*).

- The CALLEE is prohibited by his coworkers from answering calls in meetings:
  P_1 (CALLER, NEIGHBOR, *relationship = coworker* $\wedge$ *place = meeting* $\wedge$ *call, answer call*).

- Satisfaction of C_1 increases the trust (positive sanction) of the spouse toward the CALLEE:
  S_1 (CALLEE, CALLER, C_1 = *satisfied, increase trust*).

- Violation of P_1 yields a bad reputation (negative sanction) for the CALLEE among his coworkers:
  C_2 (NEIGHBOR, CALLER, P_1 = *violated, bad-mouth*).

Additional norms can be defined, limited only by the social context. For example, we might state that a university librarian has the power to declare reading rooms as quiet or as places where discussions are allowed.

Norms might conflict, as when the phone owner receives a call from their spouse during a meeting with coworkers. In this case, antecedents of both C_1 and P_1 hold, but satisfying one norm means violating the other. A designer must identify such conflicts and elicit additional contextual information to prioritize among the conflicting norms. Our conflict-resolution methodology,[11] based on an analysis of competing hypotheses, can guide a designer in prioritizing the alternatives.

Preliminary Evaluation

We conducted a developer study to evaluate our methodology. We asked subjects (34 graduate computer science students, working solo) to develop a ringer manager. We split the subjects into two groups: control (n = 16) and Danio (n = 18), providing identical application requirements to each, but providing our methodology (treatment) only to the Danio subjects. All subjects recorded their development time during the study and completed a post-survey.

Development Time

Figure 3 shows boxplots of times expended by subjects during different development phases. The diamond dots indicate mean values and the other dots indicate outliers.

We observed that the Danio subjects took slightly longer than the control subjects to understand requirements and prepare a specification. However, Danio subjects, on average, spent 17.8 and 11.4 percent less time than control subjects in implementing and testing the application, respectively. We posit that Danio's systematization of incorporating context, which costs extra time early, pays off during implementation and testing.

Post-Survey Data

We asked control subjects the extent to which a methodology would help them in developing a social application. We used a Likert scale of one (not helpful) to seven (extremely helpful) for each question; $\tilde{x}$ are mean values. The subjects responded as follows: that a methodology would be helpful for understanding requirements ($\tilde{x}$ = 5); implementation ($\tilde{x}$ = 4.5); testing ($\tilde{x}$ = 5.5); and specifications ($\tilde{x}$ = 3).

Surprisingly, control subjects felt that a methodology (which we didn't provide to them) would be least helpful for application specifications, whereas Danio subjects felt a methodology (which we did provide to them) was the most helpful for specifications

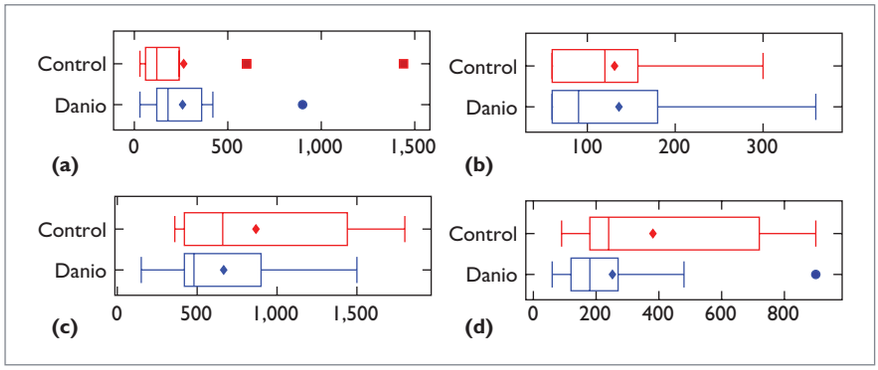


Figure 3. Time expended (in minutes) by subjects in developing the ringer manager. Time to (a) understand requirements, (b) prepare the specification, (c) implement, and (d) test.

($\tilde{x}$ = 5). The application specifications can be quite valuable in testing, and subsequently, for maintenance (although our study didn't include maintenance).

Incorporating social expectations enables an application to deliver a privacy-preserving experience by promoting personalization, disclosure and control, and accountability. Danio extends well-known design and engineering techniques to engineer privacy into social applications by incorporating social expectations. Our preliminary evaluation shows the merits of Danio and sets the stage for more extensive evaluations.

An interesting direction is extending Danio to tackle cases where user interactions are motivated by subtle tradeoffs between privacy and social utility. For example, a callee might accept a call to gloat to his neighbors about the caller having called him or to the caller about a meeting with the neighbors.

One contribution of Danio is to show how to engineer privacy in social applications in a way that accommodates aspects of privacy that are often de-emphasized. Synthesizing these diverse aspects of privacy and providing a context-sensitive way to engineer privacy with respect to diverse tradeoffs is an important challenge for future research.

Acknowledgment

We thank the US Department of Defense for support through the Science of Security Lablet.

References

1. W.L. Prosser, "Privacy," *California Law Rev.*, vol. 48, no. 3, 1960, pp. 383–423.
2. P.K. Murukannaiah and M.P. Singh, "Platys: An Active Learning Framework for Place-Aware Application Development and Its Evaluation," *ACM Trans. Software Eng. and Methodology*, vol. 24, no. 3, 2015, pp. 1–33.
3. O.A. Alsos and D. Svanæs, "Designing for the Secondary User Experience," P. Campos et al., eds., *Proc. Human-Computer Interaction*, LNCS 6949, Springer, 2011, pp. 84–91.
4. K. Battarbee, "Defining Co-Experience," *Proc. Int'l Conf. Designing Pleasurable Products and Interfaces*, 2003, pp. 109–113.
5. S. Spiekermann and L.F. Cranor, "Engineering Privacy," *IEEE Trans. Software Eng.*, vol. 35, no. 1, 2009, pp. 67–82.
6. 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.
7. P. Pasquier, R.A. Flores, and B. Chaib-draa, "An Ontology of Social Control Tools," *Proc. Int'l Joint Conf. Autonomous Agents and Multiagent Systems*, 2006, pp. 1369–1371.
8. L.G. Nardin et al., "Classifying Sanctions and Modelling a Sanctioning Process for Socio-Technical Systems," *The Knowledge Eng. Rev.*, vol. 31, no. 2 2016, pp. 1–24.
9. P. Bresciani et al., "Tropos: An Agent-Oriented Software Development Methodology," *Autonomous Agents and Multi-Agent Systems*, vol. 8, no. 3, 2004, pp. 203–236.
10. P.K. Murukannaiah and M. P. Singh, "Xipho: Extending Tropos to Engineer Context-Aware Personal Agents," *Proc. Int'l Conf. Autonomous Agents and Multi-Agent Systems*, 2014, pp. 309–316.
11. P.K. Murukannaiah et al., "Resolving Goal Conflicts via Argumentation-Based Analysis of Competing Hypotheses," *Proc. 23rd IEEE Int'l Requirements Eng. Conf.*, 2015, pp. 156–165.

Pradeep Murukannaiah is a computer science PhD candidate at North Carolina State University. His research interests include software engineering, social computing, and context-aware systems. Murukannaiah has an MS in computer science from North Carolina State University. Contact him at pmuruka@ncsu.edu.

Nirav Ajmeri is a computer science PhD student at North Carolina State University. His research interests include software engineering and multiagent systems. Ajmeri has a BE in computer engineering from Sardar Vallabhbhai Patel Institute of Technology, Gujarat University. Contact him at najmeri@ncsu.edu.

Munindar Singh is a computer science professor 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.