Protos: Foundations for Engineering Innovative Sociotechnical Systems

Amit K. Chopra*, Fabiano Dalpiaz†, F. Başak Aydemir‡, Paolo Giorgini‡, John Mylopoulos‡, Munindar P. Singh§
*Lancaster University, Lancaster, UK; email: a.chopra1@lancaster.ac.uk
†Utrecht University, Utrecht, The Netherlands; email: f.dalpiaz@uu.nl
‡University of Trento, Trento, Italy; email: {aydemir, pg, jm}@disi.unitn.it
§North Carolina State University, Raleigh, NC, USA; email: m.singh@ieee.org

***Abstract*—We address the challenge of requirements engineering for *sociotechnical* systems, wherein humans and organizations supported by technical artifacts such as software interact with one another. Traditional requirements models emphasize the goals of the stakeholders above their interactions. However, the participants in a sociotechnical system may not adopt the goals of the stakeholders involved in its specification. We motivate, Protos, a requirements engineering approach that gives prominence to the interactions of autonomous parties and specifies a sociotechnical system in terms of its participants' social relationships, specifically, commitments. The participants can adopt any goal they like, a key basis for innovative behavior, as long as they interact according to the commitments. Protos describes an abstract requirements engineering process as a series of refinements that seek to satisfy stakeholder requirements by incrementally expanding a specification set and an assumption set, and reducing requirements until all requirements are accommodated. We demonstrate this process via the London Ambulance System described in the literature.**

***Index Terms*—Requirements, refinement, architecture, commitments, teams, protocols**

I. Introduction

We define a *SocioTechnical System* (STS) as one involving interactions between humans and organizations (*principals*) facilitated by *technical artifacts*, including software. STSs under this definition include two or more autonomous parties, who ordinarily act and interact in ways that promote their respective agendas. Thus the participants may act in ways that are unexpected by others.

We further restrict our attention to STSs whose rules of encounter are formally represented, whether created via explicit engineering or otherwise. The benefit of an "institutionalized" STS is that its explicit rules of encounter facilitate reasoning by (prospective) participants about whether and how to participate in the STS. Examples of such STSs can be found in many domains, such as healthcare, finance, transportation, and commerce. STSs, especially those with explicit rules of encounter, provide a conceptual basis for innovation by the participants. (Other STSs may also support innovation but we limit our claims to institutional STSs.)

Innovations in STSs include the emergence of new social practices. For example, beginning from goals of passengers and transporters, we might design an airport as a hub where they can execute their transactions. However, an enterprising person could invent the notion of an airport as a venue for shopping in general (as Brendan O'Regan did in 1947). Similarly, the goals of providing capital for new ventures yield a market for public offerings of stocks, which have morphed into the financial systems of today. Notice that not every innovation is desirable and in general some participants would gain and some would lose from any innovation.

Classical Requirements Engineering (RE) approaches fail to sufficiently support innovation and thus do not help realize the innovative potential of STSs. Classical RE begins from an elicitation of the goals of stakeholders, followed by an analysis phase that produces a specification of a software system that would satisfy those goals given some assumptions about its operating environment. A limitation of such approaches is that the goals of the stakeholders used as a basis for modeling may have little in common with the goals of the participants in the STS (whom we call "principals"). We posit that a goal-based treatment of STSs undercuts innovation. It offers up the following dilemma. Either the participants innovate but in an ad hoc manner (such as on social media today, where memes and conventions such as emoticons emerge sporadically) or the participants are regimented to the original goals and do not innovate at all. Regimentation is not viable for autonomous participants. In practice, ad hoc innovation is what we see most often.

Motivation. Our research question is as follows: *How can RE facilitate innovation in STSs? That is, how may the stakeholders of an STS systematically produce a specification that facilitates innovation*?

We posit that the RE effort be broken into two logically distinct phases: (1) coming up with the rules of encounter in an STS and (2) given the rules of encounter, coming up with models of participants (e.g., their policies) that determine how they participate. A common representation of the rules of encounter ties these two phases together. The first phase is carried out jointly by (or on behalf of) the stakeholders specifying the STS. The second phase is carried out separately by (or on behalf of) stakeholders for each participant.

A major benefit of the aforementioned common representation is that not only does it enable the interoperation needed to realize the STS but also decouples the participants with respect to what is not specified, and thus frees them up to innovate. As long as they comply with the rules of encounter, the innovation is not ad hoc.

STSs are exemplified by many practical settings. Let us consider a healthcare system, which we understand as involving interactions among physicians, nurses, patients, hospitals, insurance sellers, and regulatory bodies. There are two potential ways one can approach the requirements engineering of such a system given stakeholder requirements. The traditional or *regimented* approach is to specify a software module that all the principals would use. The other or *interaction-oriented* approach involves is to specify the interaction protocols that would support meeting the requirements.

An interaction protocol would describe how any principal adopting one or more *roles*, such as hospital or physician, would interact with others. In particular, the protocol would specify the social expectations that principals playing one of the roles could have of principals playing other roles. Each principal would be free to develop its own software in accordance with its requirements. For example, different principals playing the role of hospital could adopt different implementations and policies by which they conduct their business. We refer to each principal's software as its *agent*.

Traditional works on requirements engineering (RE) and methodologies [1], [2] generally tend to be regimented: they address the specification of software conceptualized as a machine that resides within STSs, not the sociotechnical system itself. Approaches such as Tropos [3] begin from modeling the sociotechnical system, but end up with a regimented software system.

We refine the above question further to emphasize the foundational aspects of RE, namely, *What is a suitable formulation and formalization of the STS design space that is conducive to the autonomy both of stakeholders and principals?* To focus on the representational aspects, we place as out of our scope the important challenges of negotiation among stakeholders and of collaborative and concurrent engineering of an STS.

Contributions. We make the following contributions:

- We introduce a way of specifying an STS as a protocol in terms of roles and their social commitments [4] to each other. Our approach applies to social expectations generally, though for concreteness, we focus on commitments here.
- We present a model for requirements satisfaction in which stakeholder requirements are satisfied by a protocol specification, modulo any stated assumptions.
- We propose a generalization of one of the foundational axioms of RE that accommodates the separation of concerns between specifying an STS and an individual principal specifying its agent.
- We present an abstract design process for STSs that extends the classical idea of refinement. We refer to this extended notion of refinement as *social refinement* to emphasize the refinement of requirements into commitment-based specifications. We illustrate the process in a systematic case study of the London Ambulance System.

Organization. The rest of the paper is organized as follows. Section II introduces background on protocols and requirements refinement. Section III revisits the traditional RE problem to accommodate STSs and shows that specifying STSs and specifying agents are two independent problems. Section IV motivates and formalizes a set of refinement reductions to obtain STS specifications from stakeholders' requirements. Section V applies our approach to the London Ambulance System. Section VI discusses related work and Section VII summarizes our contributions, and outlines future directions.

II. Background

We adopt a scenario from automobile insurance (based on Browne and Kellet's [5] real-life description).

A. Classical Formulation of Requirements

Zave and Jackson [1] characterize RE in terms of A (a set of domain assumptions), M (a software (machine) specification), and R (a set of stakeholder requirements). A requirements engineer's task is to come up with a software specification and the domain assumptions that together guarantee that the requirements are met, which Eq. 1 shows:

$$A, M \vdash R \tag{1}$$

B. Commitment Models

A commitment represents a directed social expectation between principals. We express a commitment as a four-tuple $\mathsf{C}(debtor, creditor, antecedent, consequent)$: the debtor is committed to the creditor that if the antecedent holds, the consequent will hold [6]. A commitment is detached when its antecedent holds, unless it has timed out. It is discharged when its consequent holds, unless it has timed out.

Example 1. The insurance company commits to the car owner to reimburse any damages if the insurance policy is valid: $\mathsf{C}(\textsc{Insurer}, \textsc{Car Owner}, \mathsf{hasValidInsurance}, \mathsf{getReimbursed})$.

Example 2. If the Car Owner has a valid policy, the above commitment is detached and the following unconditional commitment holds: $\mathsf{C}(\textsc{Insurer}, \textsc{Car Owner}, \mathsf{true}, \mathsf{getReimbursed})$.

If the car owner is reimbursed, the insurer's commitment is discharged. The commitment is violated if the car owner owns a valid policy but does not get reimbursed for damages.

A *protocol* specifies the rules of encounter among principals adopting roles in it. In general, a commitment is established by communication from its debtor to its creditor: it is thus autonomously created and becomes part of the social state of the interacting parties. However, we concentrate here on the commitments themselves without regard to the communications that bring them about. Thus, here, a protocol specifies a set of commitments, such as in Example 1. By adopting roles in a protocol, a principal would become the creditor of some commitments and debtor of others.

Commitments, like goals, are a high-level abstraction. However, unlike goals, commitments provide a standard of correctness for interactions among principals that is independent of their mental states. Thus, for example, a particular insurance company may intend to avoid paying for the damages of a particular car owner even if the owner has valid coverage. If the insurance company goes through with its intention, though, it would violate its commitments to the car owner.

C. *Refinement*

Refinement is a foundational concept in software engineering [7]. Goal refinement, in particular, is a fundamental concept for systematically extracting a specification from a set of requirements, e.g., in Tropos.

Design refinement is a relation between design problems p and p', where p' is an incremental improvement of p, $p \hookrightarrow p'$, and a solution for problem p' also constitutes a solution for problem p. A *design space* $(P, R_{\hookrightarrow}, p_0)$ consists of an abstract set of design problems, P; a root problem, $p_0 \in P$; and a refinement relationship $R_{\hookrightarrow} \subseteq P \times P$. A root problem is one that is not a refinement of any other problem.

In Zave and Jackson's terminology an engineer could begin from the requirements in R and progressively refine them to produce an implementable M. Software development, then, is concerned with implementing M.

III. RE for Sociotechnical Systems

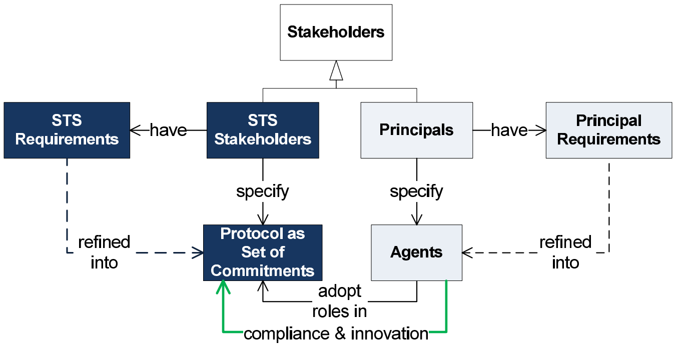

Fig. 1: The overall design space for sociotechnical systems.

Figure 1 illustrates the overall design space for STSs as discussed in Section I. Stakeholders are of two kinds: STS stakeholders and principals (that is, runtime participants in the STS). The principals of an STS might have been involved as STS stakeholders in its design, but that is not necessary. The innovation opportunity of STSs arises partly because its principals may never have been imagined as its stakeholders during design, though the principals should have something in common with the STS stakeholders. The dark boxes (and relationships) capture the design space for STS stakeholders. The light boxes (and relationships) capture an individual principal's design space. Whereas STS stakeholders specify a protocol by refining STS requirements, a principal specifies an agent from its requirements. The principal, via its agent, adopts one or more roles in the protocol. By decoupling STS requirements from principal requirements and concomitantly decoupling protocol (STS) specifications from agent specifications, we enable a principal to comply or not with a protocol. Additionally, a principal may comply with a protocol while functioning in novel ways, thereby promoting innovation.

Below, for clarity, we identify STS stakeholders with roles (based on the intuition that stakeholder types and STS roles would normally coincide) and use proper names to identify principals. Thus, for example, in an automobile insurance system, the stakeholders include Insurer, Mechanic, and Car Owner. Alessia and Cristina are particular mechanics; Great Insurance Co. is a particular insurance company.

A. *Rethinking RE Characterization*

The conception of the design space in Figure 1 necessitates a rethinking of Zave and Jackson's formulation. Eq. 1 is adequate for refining the stakeholders' requirements into a specification of a machine, i.e., the software that would reside in an STS. For example, we might specify software that would support a car owner to file a claim and an insurance staff member to approve the claim and assign a mechanic. The formulation captures the essence of regimented approaches, as illustrated by Example 3.

Example 3. Let stakeholders Insurer and Mechanic get together to design software for an automobile insurance STS (we neglect Car Owner for simplicity). Let R_1 be the set of Insurer's requirements, containing only one requirement: any replacement parts ordered by a mechanic that are priced above \$500 must be approved by the Insurer. This requirement, following Eq. 1, would be refined into a software specification that disables for mechanics the functionality of ordering parts priced over \$500 unless Insurer's approval is recorded. The specification can be implemented using an access control module.

If a principal who plays Mechanic wants to order parts without the insurance company's approval (e.g., because of urgency or because the approver—(staff member)—is on holiday), the above-mentioned software would not allow the mechanic to proceed. The mechanic would be forced to wait for approval—the software, effectively, regiments interactions [8].

What we would like instead is to come up with a specification that captures the essence of the stakeholder requirement and yet does not regiment the actions of any principal in the STS. This is possible if stakeholder requirements are refined into, not a machine, but a protocol specification. Any principal who adopts a role in the protocol is free to specify (and implement) its own agent in accordance with its own requirements. Eq. 2 captures the above intuition. Ag and S are the sets of agent and protocol specifications, respectively. Eq. 2 separates the essential nature of the interaction in the protocol from the autonomous decision-making of the participants involved in its enactment.

$$A, Ag, S \vdash R \tag{2}$$

Note that when Ag is a singleton and S is thus a null protocol, Eq. 2 reduces to Eq. 1. Therefore, Eq. 2 generalizes the traditional requirements specification problem equation, Eq. 1, to a setting of multiple autonomous principals. Example 4 illustrates Eq. 2.

Example 4. Consider R_1 from Example 3. Let S_1 be the protocol specification $\{\mathsf{C}(\textsc{Mechanic}, \textsc{Insurer}, \mathsf{price}(item) > 500, \mathsf{getApproval}(item) \text{ precedes } \mathsf{order}(item))\}$. For simplicity, let A_1 (the set of assumptions) be empty. Let $Ag_1 = \{i, m\}$; and i and m be the agents of Great Insurance Co. and Alessia, respectively. Suppose Alessia has designed m to obtain approvals before ordering any parts. The overall system is correct, that is, $A_1, Ag_1, S_1 \vdash R_1$.

Example 5. Modify Example 4 such that Alessia's agent m never asks for preapproval, so it would violate the above-mentioned commitment to obtain preapproval. Let's refer to this agent as m' and let $Ag'_1 = \{i, m'\}$. Then R_1 wouldn't be satisfied either, i.e., $A_1, Ag'_1, S_1 \nvdash R_1$.

B. Modularity of the Design Space

We can achieve the separation of design spaces illustrated in Figure 1. A protocol provides a solution for the requirements, independently of the agents who would ultimately play roles in that protocol to instantiate the STS. That is, given some assumptions A_S of the domain, protocol S ensures R under the assumption E that all principals adopting roles in the protocol satisfy their commitments. In other words, the following holds.

$$A_S, E, S \vdash R \tag{3}$$

Example 6. Returning to our example, under the assumption E (here meaning that any principal playing $\textsc{Mechanic}$ would satisfy the commitment to get approval), $A_1, E, S_1 \vdash R_1$.

Assuming E during STS design does not mean that any principal who wants to play a role in the STS must be compliant. Its purpose is to capture precisely the intuition that it is the *satisfaction* of the commitment (and not, for instance, its *creation*) that satisfies some requirement. Also note that E is not relativized to the particular STS (that is why there is no E_1 as there are R_1, S_1, and A_1). The assumption E is purely formal: it assumes the satisfaction of whichever commitments there are in the specification; in particular, assuming E is not a matter of choice for the stakeholders.

We are concerned with the process of designing protocols from requirements, as in Eq. 3 (and the dark boxes in Figure 1). Example 7 illustrates how agent specifications, as in the lighter boxes in Figure 1, fit into the overall requirements satisfaction argument.

Example 7. We know that $A_1, E, S_1 \vdash R_1$. Cristina wants to play $\textsc{Mechanic}$ in S_1. Cristina's requirements are R_c and she wants to design an agent c accordingly. R_c may or may not contain requirements imposed by the protocol. Let's say R_c contains the requirements and c is specified appropriately following Eq. 1, that is, $A_c, c \vdash R_c$. As before, let i (Great Insurance Co.'s agent) play $\textsc{Insurer}$ and, further, that $A_i, i \vdash R_i$. Putting everything together, we get $A_i, A_c, A_1, \{i, c\}, S_1 \vdash R_1$. This is like Eq. 2: $A = A_i \cup A_c \cup A_1$; $Ag = \{i, c\}$, and $S = S_1$. (The assumption E does not figure in the final equation as it is "replaced" by actual agent specifications.)

Example 7 demonstrates that even the simple modification of introducing a protocol yields interesting payoffs. The protocol provides a precise description of the nature of the interactions among principals but leaves open the possibility of enacting the protocol in multiple ways based on their own requirements, thus enabling innovation. For example, this approach accommodates heterogeneous software for different mechanics, and enables ordering before getting the approval (although a mechanic who does so may lose money if the insurance company denies the authorization).

IV. Social Refinement

We seek a theory of design for STSs founded on the concept of refinement. Since the concepts in terms of which STSs are conceived (e.g., roles, protocols, and commitments) are fundamentally different from those of traditional software engineering(e.g., goals, functions, and actions for requirements, statements and variables for programs), our task is to define new refinements that are specific to STS design problems.

Below, we introduce our model elements, use them to define a design configuration, and introduce a design process.

A. Model Elements

The following are the key primitives of Protos.

Proposition, which represents a state of the world in the domain of the STS. A proposition may be atomic or composite (representing the conjunction or disjunction of propositions). The set of all propositions is $\mathcal{P}$.

Stakeholder, an autonomous entity present during the design process of an STS.

Team, one or more stakeholders who function as a cohesive unit for design purposes. That is, no team is empty. We denote that τ_i is a subteam of τ as $\tau_i \sqsubseteq \tau$. We write a team τ that comprises subteams τ_i as $\tau = \bigsqcup_i \tau_i$. We do not consider the subtleties of organizational structures, though our approach could be enhanced to incorporate such models. The set of all teams is $\mathcal{T}$, and includes individual stakeholders as unary terms.

Commitment, a social relationship between a debtor team and a creditor team, referring to an antecedent proposition and a consequent proposition. The set of all commitments is $\mathcal{C}$. Thus, $\mathcal{C} \subseteq \mathcal{T} \times \mathcal{T} \times \mathcal{P} \times \mathcal{P}$.

Refinement, a reflexive, transitive, antisymmetric relation between propositions. We notate refinement as $\hookrightarrow$ where $a \hookrightarrow b$ means that a refines into b (that is, b is a refinement of a). Thus, $\hookrightarrow \subseteq \mathcal{P} \times \mathcal{P}$. The refinement of commitments and other constructs follows from the above. We lift refinement to design configurations where one configuration refines into another.

Conflict, an irreflexive, symmetric relationship over propositions. We notate conflict as $\oplus$ where $p \oplus q$ means that p conflicts with q. Thus, $\oplus \subseteq \mathcal{P} \times \mathcal{P}$.

Requirement, a representation of an expectation that a team would like to achieve a proposition. The set of all requirements is $\mathcal{R}$. $\mathcal{R} \subseteq \mathcal{T} \times \mathcal{P}$. The notation $\mathsf{R}(\tau, \pi)$ means that $(\tau, \pi) \in \mathcal{R}$.

Onus, a representation of the assumption that a team takes on the onus of ensuring a proposition. $\mathsf{O}(\tau, \pi)$ means that team τ takes on the onus for ensuring proposition π. We include onus assertions with other assumptions.

B. Design Configurations

We use these model elements to talk about design episodes. A design episode proceeds from one design configuration to another by systematically applying refinements. We define the following concepts to help us describe design episodes.

Requirements are given as R, a finite set of assertions describing what each team (stakeholder) wants to achieve in the STS. The teams are autonomous, each holding a stake in its own requirements, though a team's requirements may hold for its subteams. Thus, $R \subseteq \mathcal{R}$.

Specifications are given as S, a finite set of assertions describing how the STS to be will function. As motivated above, these assertions describe the interactions in the STS but not the internal details of any of its participants. The interactions are naturally captured as social relationships among the participants. Thus, $S \subseteq \mathcal{C}$.

Domain assumptions are given as A, a finite set of assertions that must hold true in order to ensure that the specifications will satisfy the requirements. The set of possible domain assumptions is $\mathcal{A} = \{p | p \in \mathcal{P}\} \cup \{\mathsf{O}(\tau, p) | \tau \in \mathcal{T}$ and $p \in \mathcal{P}\} \cup \{a \hookrightarrow b | a, b \in \mathcal{P}\}$.

Needs are given as N, a finite set of requirements. That is, $N \subseteq \mathcal{R}$. N represents the set of requirements that are yet to be addressed during the design episode.

The foregoing leads to a definition of a design configuration.

Definition 1. *Given a set of stakeholder teams $\mathcal{T}$ and a set of propositions $\mathcal{P}$, a* design configuration *is a tuple $\langle S, A, N\rangle$, where $N \subseteq \mathcal{R}$; $S \subseteq \mathcal{S}$; and $A \subseteq \mathcal{A}$.*

Table I contains an illustration of the complete process of deriving specifications from requirements for an automobile insurance scenario. C, I, M, G, and F represent the stakeholders—CAR OWNER, INSURER, MECHANIC, MANAGER, and FINANCE, respectively. MANAGER and FINANCE are subteams of INSURER; the former approves the payments and the latter makes the transactions. In Table I, the S, A, and N cells of each row constitute a design configuration. We refer to the table to illustrate our formal definitions.

The following definitions are needed in Section IV-C to specify how a design process may begin, terminate, and whether its outcome is consistent and meets the requirements.

Definition 2. *A design configuration $\langle S, A, N\rangle$ is* initial *for requirements R if and only if $N = R$, $S = \varnothing$, and $A = \varnothing$.*

In Table I, Row 1 refers to the initial design configuration. CAR OWNER's requirement is to be prepared for the emergencies: $\mathsf{R}(\mathsf{c}, \mathsf{prepared})$, INSURER has the requirement of selling insurance: $\mathsf{R}(\mathrm{I}, \mathit{sold})$, and MECHANIC wants to get paid for his repair services.

Definition 3. *A design configuration $\langle S, A, N\rangle$ is* final *if and only if $N = \varnothing$.*

In Table I, Row 8 refers to the final design configuration.

Definition 4. *A design configuration $\langle S, A, N\rangle$ is* consistent *if and only if $A \cup S \not\vdash \mathsf{false}$.*

Definition 5. *A design configuration $\langle S, A, N\rangle$* satisfies *requirements R if and only if $A \cup S \vdash R$.*

C. Design Process

The requirements of stakeholders feed a design (refinement) process, which we imagine as being conducted by stakeholders and facilitated by requirements engineers. The output of the process is an STS specification, and a set of domain assumptions. We model the design process as iteratively taking a design configuration and producing another, refined, design configuration through an application of one of the above reductions, beginning from an initial configuration and ending in a final configuration.

We initialize a design configuration from the requirements by treating each requirement as the technical debt of the relevant team. The design process iteratively addresses each need. A team may take on the onus for any of its needs. Alternatively, it may obtain a commitment from another team, in which case its need would change to the antecedent of the commitment. A design episode concludes when a configuration is obtained that resolves all needs of all teams.

Potentially, more than one reduction may apply on a given design configuration. That is, the design space can be large. Some explorations of it may end up in failure. An exploration is consistent when it constitutes a series of refinements from an initial to a final consistent configuration.

Definition 6. *A* design step *takes as input a configuration $\langle S, A, N\rangle$ and produces a configuration $\langle S', A', N'\rangle$ provided we can conclude $\langle S, A, N\rangle \hookrightarrow \langle S', A', N'\rangle$.*

In Table I, going from one row to the next is a design step (the latter row is annotated with the applied reduction from Section IV-D).

Definition 7. *A* design path *for requirements R is a finite series of configurations $\langle S_0, A_0, N_0\rangle \ldots \langle S_n, A_n, N_n\rangle$ where (1) $\langle S_0, A_0, N_0\rangle$ is initial for R; (2) $\langle S_n, A_n, N_n\rangle$ is final and consistent; and (3) for each i, $0 \leq i < n$, $\langle S_i, A_i, N_i\rangle \hookrightarrow \langle S_{i+1}, A_{i+1}, N_{i+1}\rangle$ is a design step.*

Table I shows a design path from Row 1 to Row 8.

Definition 8. *A design path $\langle S_0, A_0, N_0\rangle \ldots \langle S_n, A_n, N_n\rangle$ for requirements R is* sound *if and only if $\langle S_n, A_n, N_n\rangle$ is consistent and $\langle S_n, A_n, N_n\rangle$ satisfies R.*

In Table I, the design path from Row 1 to Row 8 is sound. Soundness relies on Theorem 1 that establishes that design paths following the reductions below are sound.

Recall Eq. 3 states that $A_S, E, S \vdash R$. It captures the problem of specifying a protocol from requirements. The

TABLE I: Illustrating refinement on the insurance scenario

	Specification (S)	Assumptions (A)	Needs (N)	Refinement type
1	$\varnothing$	$\varnothing$	$\mathsf{R}(\textsc{c}, \mathit{prepared})$, $\mathsf{R}(\textsc{i}, \mathit{sold})$, $\mathsf{R}(\textsc{m}, \mathit{paid})$	
2	$\varnothing$	$A_1 = \{\mathit{prepared} \hookrightarrow \mathit{covered} \wedge \mathit{eme}\}$	$\mathsf{R}(\textsc{c}, \mathit{covered} \wedge \mathit{eme})$, $\mathsf{R}(\textsc{i}, \mathit{sold})$, $\mathsf{R}(\textsc{m}, \mathit{paid})$	Need refinement
3	$\mathsf{C}(\textsc{c}, \textsc{i}, \mathit{covered}, \mathit{sold})$, $\mathsf{C}(\textsc{i}, \textsc{c}, \mathit{sold}, \mathit{covered})$	A_1	$\mathsf{R}(\textsc{c}, \mathit{eme} \wedge \mathit{sold})$, $\mathsf{R}(\textsc{i}, \mathit{covered})$, $\mathsf{R}(\textsc{m}, \mathit{paid})$	Cyclic commitments
4	$\mathsf{C}(\textsc{c}, \textsc{i}, \mathit{repaired}, \mathit{data} \wedge \mathit{fee})$, $\mathsf{C}(\textsc{i}, \textsc{c}, \mathit{data} \wedge \mathit{fee}, \mathit{repaired})$	$A_2 = A_1 \cup \{\mathit{sold} \hookrightarrow \mathit{data} \wedge \mathit{fee},\ \mathit{covered} \hookrightarrow \mathit{repaired}\}$	$\mathsf{R}(\textsc{c}, \mathit{eme} \wedge \mathit{data} \wedge \mathit{fee})$, $\mathsf{R}(\textsc{i}, \mathit{repaired})$, $\mathsf{R}(\textsc{m}, \mathit{paid})$	Commitment refinement *i*
5	$\mathsf{C}(\textsc{c}, \textsc{i}, \mathit{found} \wedge \mathit{fixed}, \mathit{data} \wedge \mathit{fee})$, $\mathsf{C}(\textsc{i}, \textsc{c}, \mathit{data} \wedge \mathit{fee},\ \mathit{found} \wedge \mathit{fixed})$	$A_3 = A_2 \cup \{\mathit{repaired} \hookrightarrow \mathit{found} \wedge\ \mathit{fixed}\}$	$\mathsf{R}(\textsc{c}, \mathit{eme} \wedge \mathit{data} \wedge \mathit{fee})$, $\mathsf{R}(\textsc{i}, \mathit{found} \wedge \mathit{fixed})$, $\mathsf{R}(\textsc{m}, \mathit{paid})$	Commitment refinement *i*
6	$\mathsf{C}(\textsc{c}, \textsc{i}, \mathit{found} \wedge \mathit{fixed}, \mathit{data} \wedge \mathit{fee})$, $\mathsf{C}(\textsc{i}, \textsc{c}, \mathit{data} \wedge \mathit{fee}, \mathit{found} \wedge \mathit{fixed})$, $\mathsf{C}(\textsc{m}, \textsc{i}, \mathit{paid}, \mathit{fixed})$, $\mathsf{C}(\textsc{i}, \textsc{m}, \mathit{fixed}, \mathit{paid})$	A_3	$\mathsf{R}(\textsc{c}, \mathit{eme} \wedge \mathit{data} \wedge \mathit{fee})$, $\mathsf{R}(\textsc{i}, \mathit{found} \wedge \mathit{paid})$, $\mathsf{R}(\textsc{m}, \mathit{fixed})$	Cyclic commitments
7	$\mathsf{C}(\textsc{c}, \textsc{i}, \mathit{found} \wedge \mathit{fixed}, \mathit{data} \wedge \mathit{fee})$, $\mathsf{C}(\textsc{i}, \textsc{c}, \mathit{data} \wedge \mathit{fee}, \mathit{found} \wedge \mathit{fixed})$, $\mathsf{C}(\textsc{m}, \textsc{i}, \mathit{paid}, \mathit{fixed})$, $\mathsf{C}(\textsc{i}, \textsc{m}, \mathit{fixed}, \mathit{paid})$	$A_4 = A_3 \cup \{\textsc{g} \sqsubset \textsc{i}, \textsc{f} \sqsubset \textsc{i}, \mathit{paid} \hookrightarrow \mathit{transfer} \wedge \mathit{app}\}$	$\mathsf{R}(\textsc{c}, \mathit{eme} \wedge \mathit{data} \wedge \mathit{fee})$, $\mathsf{R}(\textsc{i}, \mathit{found})$, $\mathsf{R}(\textsc{m}, \mathit{fixed})$, $\mathsf{R}(\textsc{g}, \mathit{app})$, $\mathsf{R}(\textsc{f}, \mathit{transfer})$	Subteams
8	$\mathsf{C}(\textsc{c}, \textsc{i}, \mathit{found} \wedge \mathit{fixed}, \mathit{data} \wedge \mathit{fee})$, $\mathsf{C}(\textsc{i}, \textsc{c}, \mathit{data} \wedge \mathit{fee}, \mathit{found} \wedge \mathit{fixed})$, $\mathsf{C}(\textsc{m}, \textsc{i}, \mathit{paid}, \mathit{fixed})$, $\mathsf{C}(\textsc{i}, \textsc{m}, \mathit{fixed}, \mathit{paid})$	$A_5 = A_4 \cup \{O(\textsc{c}, \mathit{eme} \wedge \mathit{fee} \wedge \mathit{data}), O(\textsc{i}, \mathit{found}), O(\textsc{m}, \mathit{fixed}), O(\textsc{g}, \mathit{app}), O(\textsc{f}, \mathit{transfer})\}$	$\varnothing$	Onus

design process described above conforms to Eq. 3. S_n and A_n in the design path for R map to S and $A_S \cup E$, respectively. If the design path is sound, we obtain the relation $S_n, A_n \vdash R$.

D. Social Refinement Types

Below, we list social refinement types supported by Protos. (Here the set operators associate to the left and to reduce clutter we do not place assertions in quotations.)

Need refinement: Based on proposition refinement. The intuition is that if p refines to p', a need for p can be met by meeting a need for p'. For soundness, we must record the assumption that p refines to p'.

$$\langle S, A, N \cup \{\mathsf{R}(\tau, p)\}\rangle \hookrightarrow \langle S, A \cup \{p \hookrightarrow p'\}, N \setminus \{\mathsf{R}(\tau, p)\} \cup \{\mathsf{R}(\tau, p')\}\rangle$$

Notice that we write the needs set as $N \cup \{\mathsf{R}(\tau, p)$ to ensure that a need $\mathsf{R}(\tau, p)$ belongs to the needs set. In the resulting configuration, we remove $\mathsf{R}(\tau, p)$ from N to make sure it is not present in the resulting needs set and introduce the refined need $\mathsf{R}(\tau, p')$ explicitly.

Example 8. Row 2 in Table I is obtained from Row 1 by refining $\mathsf{R}(\mathsf{C}, \mathit{prepared})$ into two other needs, $\mathsf{R}(\mathsf{C}, \mathit{covered})$ and $\mathsf{R}(\mathsf{C}, \mathit{eme})$, namely, accident coverage and emergency response, respectively.

Commitment introduction i: If τ_1 has a need for q, τ_1 can address that need by obtaining a commitment from τ_0 to τ_1 whereby τ_0 commits to bringing about q provided p holds. Here, τ_1 takes on the need for p.

$$\langle S, A, N \cup \{\mathsf{R}(\tau_1, q)\}\rangle \hookrightarrow \langle S \cup \{\mathsf{C}(\tau_0, \tau_1, p, q)\}, A, N \setminus \{\mathsf{R}(\tau_1, q)\} \cup \{\mathsf{R}(\tau_1, p)\}\rangle$$

Commitment introduction ii: If τ_1 has a need for q, τ_1 can address that need by obtaining a commitment from τ_0 to τ_1 whereby τ_0 commits to bringing about q provided p holds. In this case, we add an assumption that p will hold.

$$\langle S, A, N \cup \{\mathsf{R}(\tau_1, q)\}\rangle \hookrightarrow \langle S \cup \{\mathsf{C}(\tau_0, \tau_1, p, q)\}, \{A \cup \{p\}\}, N \setminus \{\mathsf{R}(\tau_1, q)\}\rangle$$

Commitment refinement i: Based on the refinement of either or both of its antecedent and consequent. That is, if the antecedent or consequent of a commitment can be refined, then so can the commitment. As before, we record the refinements as assumptions.

$$\langle S \cup \{\mathsf{C}(\tau_0, \tau_1, p, q)\}, A, N \cup \{\mathsf{R}(\tau_1, p)\}\rangle \hookrightarrow \langle S \setminus \{\mathsf{C}(\tau_0, \tau_1, p, q)\} \cup \{\mathsf{C}(\tau_0, \tau_1, p', q')\}, A \cup \{p \hookrightarrow p', q \hookrightarrow q'\}, N \setminus \{\mathsf{R}(\tau_1, p)\} \cup \{\mathsf{R}(\tau_1, p')\}\rangle$$

Notice that, the resulting commitment need not logically entail the original commitment. For example, a commitment to provide coffee for payment may be refined into a commitment to provide coffee for payment of Euros, though the second commitment is weaker than the original.

Example 9. Row 4 in Table I is obtained by refining the commitments in Row 3. Specifically, buying a policy is refined into providing personal data and paying the fee, whereas providing coverage is refined into repairing the car. The commitments and needs in Row 4 reflect this refinement.

Commitment refinement ii: Based on the refinement of either or both of the commitment's creditor and debtor.

$$\langle S \cup \{\mathsf{C}(\tau_0, \tau_1, p, q)\}, A, N \cup \{\mathsf{R}(\tau_1, p)\}\rangle \hookrightarrow \langle S \setminus \{\mathsf{C}(\tau_0, \tau_1, p, q)\} \cup \{\mathsf{C}(\tau_{0'}, \tau_{1'}, p, q)\}, A \cup \{\tau_{0'} \sqsubseteq \tau_0, \tau_{1'} \sqsubseteq \tau_1\}, N \setminus \{\mathsf{R}(\tau_1, p)\} \cup \{\mathsf{R}(\tau_{1'}, p)\}\rangle$$

Cyclic commitments: Given n teams ($n \geq 2$) where for each i, $0 \leq i < n$, team τ_i has a need for p_i, these teams can address their needs via (cyclic) commitments from τ_i to $\tau_{(i+1 \mod n)}$ whereby τ_i commits to bringing about $p_{(i+1 \mod n)}$ provided p_i holds. Reciprocal commitments are a special case of cyclic commitments when $n = 2$.

$$\langle S, A, N \cup \bigcup_i \{\mathsf{R}(\tau_i, p_i)\}\rangle \hookrightarrow$$
$$\langle S \cup \bigcup_i \{\mathsf{C}(\tau_i, \tau_{(i+1 \mod n)}, p_i, p_{(i+1 \mod n)})\}, A,$$
$$N \setminus \bigcup_i \{\mathsf{R}(\tau_i, p_i)\} \cup \bigcup_i \{\mathsf{R}(\tau_{(i+1 \mod n)}, p_i)\}\rangle$$

Example 10. Row 3 in Table I is obtained from Row 2 by applying Cyclic Commitments. In Row 2, CAR OWNER and INSURER need coverage and sale of policies, respectively. To meet these needs, in Row 3, CAR OWNER commits to buying a policy if INSURER provides coverage for accidents and INSURER commits to providing coverage if a policy is bought. Further, CAR OWNER and INSURER need to buy policy and provide coverage, respectively.

Subteams: If a team τ has a need p, we can assign p_i to subteams τ_i; this is appropriate only because we assume that τ_i is a subteam of τ.

$$\langle S, A, N \cup \{\mathsf{R}(\tau, p)\}\rangle \hookrightarrow$$
$$\langle S, A \cup \{p \hookrightarrow \bigwedge_i p_i\} \cup \bigcup_i \{\tau_i \sqsubseteq \tau\},$$
$$N \setminus \{\mathsf{R}(\tau, p)\} \cup \bigcup_i \{\mathsf{R}(\tau_i, p_i)\}\rangle$$

Example 11. Row 7 in Table I is obtained by applying Subteams. The payment need $\mathsf{R}(\mathrm{I}, \mathit{paid})$ is refined into approval and transaction, namely, $\mathsf{R}(\mathrm{I}, \mathit{app})$ and $\mathsf{R}(\mathrm{I}, \mathit{transfer})$. The manager and the finance department, which are the insurer's subteams, adopt these needs, that is, $\mathsf{R}(\mathrm{G}, \mathit{app})$ and $\mathsf{R}(\mathrm{F}, \mathit{paid})$.

Onus: A team takes on the onus for some need locally; that is, it decides not to delegate that need to another team.

$$\langle S, A, N \cup \{\mathsf{R}(\tau, p)\}\rangle \hookrightarrow$$
$$\langle S, A \cup \{\mathsf{O}(\tau, p)\}, N \setminus \{\mathsf{R}(\tau, p)\}\rangle$$

Example 12. Row 8 is obtained by applying Onus: all stakeholders take on the onus for their remaining needs.

Composition: Refinements compose in that, if part of a configuration is refined through a particular operation, so is an entire configuration through the same operation.

$$\frac{\langle S', A', N'\rangle \hookrightarrow \langle S'', A'', N''\rangle \qquad (S \cup S' \cup A \cup A') \not\vdash \mathsf{false}}{\langle S \cup S', A \cup A', N \cup N'\rangle \hookrightarrow \langle S \cup S'', A \cup A'', N \setminus N' \cup N''\rangle}$$

E. Logic of Design Elements

For simplicity, we assume that a logic is available for reasoning about the various elements of a design configuration. In particular, this logic provides an inference relation, notated $\vdash$. The various elements of a design configuration are closed under $\vdash$, as specified by Table II.

F. Example of Design Paths

Figure 2 illustrates the above definitions. The root is the requirement. Immediately below the root is the initial design configuration. Each of the leaves is a final configuration and satisfies the requirements. Each configuration is consistent. Each path is a well-formed design path and is sound for the requirements. The edge labels are informal descriptions.

TABLE II: The underlying logic is propositional logic augmented with the following axioms pertaining to the symbols introduced in Protos.

Conflict means we cannot satisfy both	$\dfrac{p \quad p \oplus q}{\neg q}$
A subteam's stronger need satisfies a team's need: AND	$\dfrac{\mathsf{R}(\tau', p \wedge q) \quad \tau' \sqsubseteq \tau}{\mathsf{R}(\tau, p)}$
A subteam's stronger need satisfies a team's need: OR	$\dfrac{\mathsf{R}(\tau', p) \quad \tau' \sqsubseteq \tau}{\mathsf{R}(\tau, p \vee q)}$
Subteams together cover needs that a team's need refines to	$\dfrac{\bigwedge_i \mathsf{R}(\tau^i, p_i) \quad p \hookrightarrow \bigwedge_i p_i \quad \bigwedge_i \tau^i \sqsubseteq \tau}{\mathsf{R}(\tau, p)}$
If a team takes on an onus, the corresponding need is covered	$\dfrac{\mathsf{O}(\tau, p)}{\mathsf{R}(\tau, p)}$
A conditional commitment along with its antecedent cover its consequent	$\dfrac{\mathsf{R}(\tau, p) \quad \mathsf{C}(\tau', \tau, p, q)}{\mathsf{R}(\tau, q)}$
An unconditional commitment covers its consequent	$\dfrac{\mathsf{C}(\tau', \tau, \mathsf{true}, q)}{\mathsf{R}(\tau, q)}$

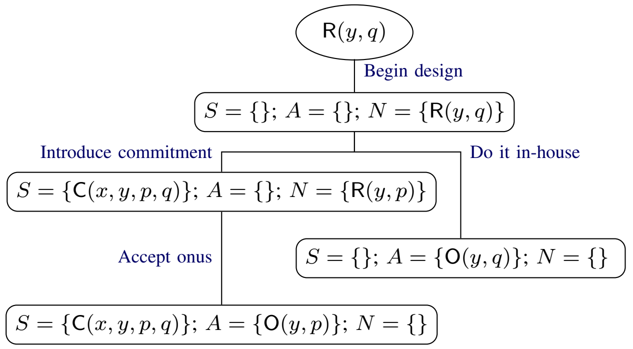
$\mathsf{R}(y, q)$
Begin design
$S = \{\}$; $A = \{\}$; $N = \{\mathsf{R}(y, q)\}$
Introduce commitment
Do it in-house
$S = \{\mathsf{C}(x, y, p, q)\}$; $A = \{\}$; $N = \{\mathsf{R}(y, p)\}$
Accept onus
$S = \{\}$; $A = \{\mathsf{O}(y, q)\}$; $N = \{\}$
$S = \{\mathsf{C}(x, y, p, q)\}$; $A = \{\mathsf{O}(y, p)\}$; $N = \{\}$

Fig. 2: Paths through the design space.

G. Soundness

We now establish the soundness of our formal model.

Theorem 1 (Soundness)**.** *Let $P = \langle S_0, A_0, N_0\rangle \ldots \langle S_n, A_n, N_n\rangle$ be a design path for requirements R. Then P is sound.*

PROOF SKETCH. *Establish the invariant that $A \cup S \cup N \vdash R$ by structural induction: it holds for an initial configuration by construction and for each subsequent configuration by inspection of the reductions. In a final configuration, $N = \varnothing$: hence we have the result.*

The above reductions do not consider conflict. The following reduction ensures that if τ_1 and τ_2 (could be the same

TABLE III: A portion of the design process during the modeling session on the London Ambulance System

Step	Specification	Assumptions	Needs	Refinement
1	$\mathsf{C}(\mathrm{CT}, \mathrm{SC}, \mathit{address} \wedge \mathit{status}, \mathit{incidentTaken})$	$A_1 = \{\mathit{incidentReported} \hookrightarrow \mathit{address} \wedge \mathit{status}, O(\mathrm{SC}, \mathit{address} \wedge \mathit{status}), O(\mathrm{CT}, \mathit{incidentTaken})\}$	$\mathsf{R}(\mathrm{SC}, \mathit{ambReceived})$, $\mathsf{R}(\mathrm{AC}, \mathit{mobilityInfoSent})$, $\mathsf{R}(\mathrm{RO}, \mathit{resourceAllocated})$, $\mathsf{R}(\mathrm{RA}, \mathit{infoReported})$	Onus
2	$\mathsf{C}(\mathrm{CT}, \mathrm{SC}, \mathit{address} \wedge \mathit{status}, \mathit{incidentTaken})$ $\mathsf{C}(\mathrm{CT}, \mathrm{RA}, \mathit{incidentTaken}, \mathit{infoReported})$	A_1	$\mathsf{R}(\mathrm{SC}, \mathit{ambReceived})$, $\mathsf{R}(\mathrm{AC}, \mathit{mobilityInfoSent})$, $\mathsf{R}(\mathrm{RO}, \mathit{resourceAllocated})$, $\mathsf{R}(\mathrm{CT}, \mathit{infoReported})$	Commitment introduction *ii*
3	$\mathsf{C}(\mathrm{CT}, \mathrm{SC}, \mathit{address} \wedge \mathit{status}, \mathit{incidentTaken})$ $\mathsf{C}(\mathrm{CT}, \mathrm{RA}, \mathit{incidentTaken}, \mathit{infoReported})$	$A_2 = A_1 \cup \{O(\mathrm{CT}, \mathit{infoReported})\}$	$\mathsf{R}(\mathrm{SC}, \mathit{ambReceived})$, $\mathsf{R}(\mathrm{AC}, \mathit{mobilityInfoSent})$, $\mathsf{R}(\mathrm{RO}, \mathit{resourceAllocated})$	Onus
4	$\mathsf{C}(\mathrm{CT}, \mathrm{SC}, \mathit{address} \wedge \mathit{status}, \mathit{incidentTaken})$ $\mathsf{C}(\mathrm{CT}, \mathrm{RA}, \mathit{incidentTaken}, \mathit{infoReported})$ $\mathsf{C}(\mathrm{RA}, \mathrm{RO}, \mathit{infoReported}, \mathit{resourceAllocated})$ $\mathsf{C}(\mathrm{RO}, \mathrm{AC}, \mathit{resourceAllocated}, \mathit{mobilityInfoSent})$ $\mathsf{C}(\mathrm{AC}, \mathrm{SC}, \mathit{mobilityInfoSent}, \mathit{ambReceived})$	A_2	$\mathsf{R}(\mathrm{AC}, \mathit{ambReceived})$, $\mathsf{R}(\mathrm{RO}, \mathit{mobilityInfoSent})$, $\mathsf{R}(\mathrm{RA}, \mathit{resourceAllocated})$	Commitment introduction *ii*
5	$\mathsf{C}(\mathrm{CT}, \mathrm{SC}, \mathit{address} \wedge \mathit{status}, \mathit{incidentTaken})$ $\mathsf{C}(\mathrm{CT}, \mathrm{RA}, \mathit{incidentTaken}, \mathit{infoReported})$ $\mathsf{C}(\mathrm{RA}, \mathrm{RO}, \mathit{infoReported}, \mathit{resourceAllocated})$ $\mathsf{C}(\mathrm{RO}, \mathrm{AC}, \mathit{resourceAllocated}, \mathit{mobilityInfoSent})$ $\mathsf{C}(\mathrm{AC}, \mathrm{SC}, \mathit{mobilityInfoSent}, \mathit{ambReceived})$	$A_3 = A_2 \cup \{O(\mathrm{AC}, \mathit{ambReceived})$, $O(\mathrm{RO}, \mathit{mobilityInfoSent})$, $O(\mathrm{RA}, \mathit{resourceAllocated})\}$	$\varnothing$	Onus

SC: service consumer, CT: call taker, RA: resource allocator, RO: radio operator, AC: ambulance crew

team) have conflicting needs, at most one of those needs can be pursued.

Conflict introduction: Introduce an assumption of a conflict into a design configuration.

$$\langle S, A, N \cup \{\mathsf{R}(\tau_1, p), \mathsf{R}(\tau_2, q)\}\rangle \hookrightarrow$$
$$\langle S, A \cup \{p \oplus q\}, N \setminus \{\mathsf{R}(\tau_2, q)\} \cup \{\mathsf{R}(\tau_1, p)\}\rangle$$

If we include the above reduction, Theorem 1 fails: although consistency is preserved, we can no longer establish that the original requirements are satisfied, because some may be dropped along the way. Incorporating conflict would lead us to formalize *satisficing* [9]. Then we may seek to establish that a design process satisfices the stated requirements.

Let us see how Protos would support design in the presence of conflicts. Imagine in Table I that CAR OWNER has an additional requirement concerning data privacy, that is, $\mathsf{R}(\mathrm{C}, \mathit{privacy})$. Conceptually, this requirement would conflict with the disclosure of personal data, that is, $\mathsf{R}(\mathrm{C}, \mathit{disclosure})$. In Protos, the stakeholder would introduce an assumption that $\mathit{disclosure} \oplus \mathit{privacy}$ via Conflict Introduction. Then, by Conflict from Table II, we are guaranteed that one of the two propositions, i.e., one of the two requirements, cannot be met. In essnce, the design process fails at this point.

V. Preliminary Evaluation

We conducted a preliminary evaluation via a modeling session involving modelers other than the authors. To this end, we recruited eight modelers, all doctoral students in computing who are familiar with goal modeling, to participate in our modeling session. At the beginning of the session, we instructed the participants in Protos concepts, reductions, and design process.

We instructed the modelers to apply Protos to jointly create a specification of Kramer and Wolf's [10] case study of the London Ambulance System (LAS), which we described for the modelers. Kramer and Wolf's LAS scenario includes nine stakeholders, namely, the service consumer, call taker, call reviewer, LAS management, ambulance crew, call reviewer, radio operator, operator at ambulance station, dispatcher, and resource allocator. The requirement of the service consumer is to receive an ambulance when there is an incident. The call taker needs incident details to report the incident and the resource allocator requires the incident report. The radio has the requirement of the resource allocation information and the ambulance crew needs mobility instructions. For our modeling session, we merged the call taker and call reviewer stakeholders to ensure a one-to-one correspondence between the study participants and the stakeholders.

The study participants then designed the LAS STS starting from their respective requirements. Table III provides a partial design process with a few representative stakeholders where the call taker and the service consumer established a commitment and refined it prior to Step 1 of Table III. Both stakeholders take the respective onuses which are added into the assumptions set in Step 1. In Step 2, the call taker and resource allocator apply commitment introduction *ii* and the call taker commits to the resource allocator to report information about the incident upon receiving such information. In Step 3 the call taker takes the onus for this commitment. Step 4 and 5 are compressed representations of the iterative design process that is similar to Step 2 and 3, where at each step commitment introduction *ii* is applied and then the oOnus is taken by the respective stakeholder.

Observations. The commitment network created by the participants following Protos superficially resembles the *i** strategic dependency diagram for LAS, as provided in [11]. However, the Protos modeling of the LAS offers distinct advantages over the *i** modeling of the LAS. One, the participants were able to collectively refine the commitments to make the interaction explicit and concrete. Two, the stakeholders were able to model conditional interactions via commitments, whereas an *i** dependency is not conditional. Three, the design reductions of Protos helped focus the design process; *i** lacks such a formalization. Four, removing the needs as the respective stakeholders took on the corresponding onuses helped the participants track the status of the design. Further, an empty needs set served as a clear stopping criterion for the design process; such a criterion does not exist in *i**.

Threats to validity. The reference document used for the LAS domains [10] includes a small number of stakeholders relative to other examples such as smart cities or a national health-care system. This somewhat narrow starting point and the participants' limited familiarity with the domain prevented them exploring designs that are not described in the reference document. Despite this, however, the participants elaborated details of interaction that are not present in the reference document by negotiating and using argumentation. The participants were collaborative and understanding to others' demands. In a real-life scenario one may expect more aggressive behavior from the stakeholders due to their conflict of interests that may even result in failure in the design process.

VI. Relevant Literature

Our conception of an STS is in terms of a protocol specified as a set of commitments. In our conception, the STS itself, unlike Baxter and Sommerville's conception [12], does not have goals. Individuals principals, on the other hand, may have goals and may represent them in their agents.

Tropos [3] models STSs in terms of dependencies among *actors*, which can be stakeholders. Tropos dependencies differ from commitments in that they refer to the goals of the actors [13]. Tropos does not specify a protocol; instead it specifies software that implements functionality to achieve stakeholder goals. This is effectively a regimented approach. Still the idea of formal requirements analysis is one we adopt from Tropos, though we approach it quite differently: the name Protos is an anagram of Tropos. Telang and Singh [14] propose an extension of Tropos with commitments that enables specifying cross-organizational business interactions. We go further by providing the formal foundations for RE for sociotechnical systems. Dalpiaz et al. [15] illustrate how principals providing services can use commitments to and from others in reasoning about the satisfaction of their own goals. However, such work does not explain how the commitments are obtained starting from the stakeholders needs. Liu et al. [16] formalize commitments in a different sense—as a relation between a principal and a service, not between principals, as in done in Protos. Dalpiaz et al. [17] model adaptive STSs using Tropos; the approach could benefit from the adoption of Protos requirements analysis in order to obtain a well-founded and accurate specification of the STS.

Protos includes STS stakeholders explicitly in the design process and represents them explicitly in its output in contrast to RE methodologies such as KAOS [18] and problem frames [19]. Notably, neither approach supports deriving protocol specifications—like Tropos, their focus is on software specification. Mahfouz et al. [20] apply goal modeling toward deriving choreography specifications. Choreographies specify interaction among principals but at a lower level of abstraction (via control flow constructs) than commitments.

Work on viewpoints in software engineering [21], [22] emphasizes the need to look at a system from different perspectives. In Protos, a stakeholder represents a *viewpoint*. Commitments represent a way of relating viewpoints formally (both at design and runtime). Castro et al. [23] note the complexity of Tropos models, especially the lack of modularity. Since commitments decouple principals (and their agents), they significantly alleviate the modularity problem.

Sutcliffe [24] argues for using multiple kinds of description for effective design—scenarios, design rationale, models, and so on. In the present paper, we mainly focus on deriving formal models of systems. In earlier work on commitments [25], we applied argumentation as a way of recording the design rationale of stakeholders. Extending Protos with argumentation is a direction of future work.

In mainstream software engineering, especially RE, there is increasing recognition of the importance of representing the social aspects of systems [26]. Newer approaches such as Sommerville et al.'s [27] apply *responsibility* modeling to represent contingency plans in organizations. However, reducing responsibilities to plans makes the approach a regimented one. With Protos, social and organizational factors make their appearance not only as requirements and assumptions, but in the specifications themselves as social expectations. Notice that commitments imply responsibility: a commitment for something means the debtor is responsible to the creditor for bringing about that thing. Dardenne et al.'s [18] notion of responsibility is not a relation between principals. Strens and Dobson [28] use a directed notion of responsibility as a structure that specifies not only what the responsibility-holder is responsible for but also his or her obligations. Assuming that their responsibilities are like commitments, specifying obligations in addition is unnecessary.

The need to involve end-users in the design process has been noted and addressed in the literature on Participatory Design in the fields of Human Computer Interaction (HCI) and Computer Supported Collaborative Work (CSCW) for more than twenty years (e.g., [29], [30]). Questions answered in such research range from "Who participates?", "By what means?" and "During which part of the design process?". Results from such research include novel design processes, such as semi-structured workshops and collaborative prototyping, new media for design, such as graphical facilitation and storyboarding, and card games, as well as cooperative prototyping and cooperative evaluation. We share many of the principles of this research. There are, however, several key differences in objectives and research methodology between our proposal and this body of research. Firstly, our focus is on design of STSs, whereas much of this literature focuses on interfaces and atomic software systems. Secondly, our focus is on the requirements problem for STSs and how to solve it through a design process, as opposed to addressing abstract user concerns. Thirdly, we are offering a formal framework for a design process that explores a formally defined design space, amenable to formal analysis and tool support. Instead, HCI and CSCW work has emphasized the pragmatics of the design process that are outside the scope of our current research.

VII. Conclusions and Directions

Protos is a novel RE process for sociotechnical systems that enables refining stakeholder requirements into commitment-based system specifications. Whereas other approaches are conceptually founded upon the notion of refining requirements into machines, Protos refines them into protocols. This brings modularity to the problem space: the problem of designing principals' software (agents) is related but separate from the problem of specifying protocols. Further, it demonstrates a generalization of Zave and Jackson's foundational characterization of RE.

We emphasize the point about the divergence of the requirements of the STS stakeholders from the requirements of the principals. Supporting this divergence is the key to innovation. Simply put, whatever the goals of the stakeholders might be we simply cannot install such goals in the decision models for the individual principals.

Protos opens up some interesting and important directions for further research. First, the emphasis on multiple stakeholders suggests a deeper study of conflict at design and run time, incorporating the notion of *satisficing* requirements, possibly in relation to notions such as social welfare of the stakeholders and bringing to bear techniques such as argumentation [25] for determining which of the conflictin requirements to satisfy and which to ignore. Second, the sociotechnical setting opens up challenges of vagueness, inconsistency, and defeasibility of requirements [31]. Third, we would need to explore modeling concepts geared for requirements in diverse STS settings, e.g., with respect to the normative relationships [32], and for which we can establish results such as completeness. Fourth, it would be crucial to develop a methodology and tools that support the Protos design process.

Acknowledgments

Thanks to Anup Kalia for comments on a previous version of this paper. John Mylopoulos, Paolo Giorgini, and Fatma Başak Aydemir were supported in part by European Research Council advanced grant 267856. Amit Chopra was partially supported by a Marie Curie Trentino Cofund grant. Munindar Singh was partially supported by U.S. Department of Defense under the Science of Security Lablet grant.

References

[1] P. Zave and M. Jackson, "Four dark corners of requirements engineering," *ACM TOSEM*, vol. 6, no. 1, pp. 1–30, 1997.

[2] A. van Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 2009.

[3] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An agent-oriented software development methodology," *JAAMAS*, vol. 8, no. 3, pp. 203–236, 2004.

[4] M. P. Singh, "An ontology for commitments in multiagent systems: Toward a unification of normative concepts," *AI & Law*, vol. 7, pp. 97–113, 1999.

[5] S. Browne and M. Kellett, "Insurance (Motor Damage Claims) Scenario," *Doc. Identifier D1.a*, 1999.

[6] M. P. Singh, "Semantical considerations on dialectical and practical commitments," in *Proc. AAAI*, Jul. 2008, pp. 176–181.

[7] R. J. Back and J. Wright, *Refinement Calculus: A Systematic Introduction*. Springer-Verlag, 1998.

[8] A. J. I. Jones and M. J. Sergot, "On the characterisation of law and computer systems: the normative systems perspectives," in *Proc. DEON*, 1993.

[9] H. Simon, *The Sciences of the Artificial*, 3rd ed. Cambridge, MA: MIT Press, 1996.

[10] J. Kramer and A. L. Wolf, "Proceedings of the 8th International Workshop on Software Specification and Design," *ACM SIGSOFT*, vol. 21, no. 5, pp. 21–35, 1996.

[11] V. E. S. Souza, "An Experiment on the Development of an Adaptive System based on the LAS-CAD—Incomplete Version1," DISI, University of Trento, Tech. Rep., 2012.

[12] G. Baxter and I. Sommerville, "Socio-technical systems: From design methods to systems engineering," *Interact. Comput.*, vol. 23, no. 1, pp. 4–17, 2011.

[13] A. K. Chopra and P. Giorgini, "Requirements engineering for social applications," in *Proc. i* Work.*, 2011.

[14] P. R. Telang and M. P. Singh, "Enhancing Tropos with commitments," in *Conceptual Modeling*, Springer, 2009.

[15] F. Dalpiaz, A. K. Chopra, P. Giorgini, and J. Mylopoulos, "Adaptation in open systems: Giving interaction its rightful place," in *Proc. ER*, 2010, pp. 31–45.

[16] L. Liu, Q. Liu, C.-H. Chi, Z. Jin, and E. Yu, "Towards a service requirements modelling ontology based on agent knowledge and intentions," *Int. J. Agent-Oriented Softw. Eng.*, vol. 2, no. 3, pp. 324–349, 2008.

[17] F. Dalpiaz, P. Giorgini, and J. Mylopoulos, "Adaptive socio-technical systems: a requirements-based approach," in *Requirements Engineering*, vol. 18, no. 1, pp. 1–24, 2013.

[18] A. Dardenne, A. Van Lamsweerde, and S. Fickas, "Goal-directed requirements acquisition," *Sci. Comput. Program.*, vol. 20, no. 1, pp. 3–50, 1993.

[19] M. Jackson, *Problem Frames: Analyzing and Structuring Software Development Problems*. Addison-Wesley, 2000.

[20] A. Mahfouz, L. Barroca, R. Laney, and B. Nuseibeh, "Requirements-driven collaborative choreography customization," in *Proc. ICSOC*, 2009, pp. 144–158.

[21] J. C. S. do Prado Leite and P. A. Freeman, "Requirements validation through viewpoint resolution," *IEEE Trans. Softw. Eng.*, vol. 17, no. 12, pp. 1253–1269, Dec. 1991.

[22] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke, "Viewpoints: A framework for integrating multiple perspectives in system development," *Intl. J. Softw. Eng. Know.*, vol. 2, no. 1, pp. 31–57, 1992.

[23] J. Castro, M. Kolp, L. Liu, and A. Perini, "Dealing with Complexity Using Conceptual Models Based on Tropos," in *Conceptual Modeling Foundations and Applications*, Springer, 2009, pp.335–362.

[24] A. Sutcliffe, "Integrating Design Representations for Creativity," Springer, 2013, no. 20, pp. 85–104.

[25] A. K. Chopra and M. P. Singh, "Colaba: Collaborative design of cross-organizational business processes," in *Proc. Wkshp. Req. Eng. Syst., Services, & Syst. of Syst/.* IEEE, 2011, pp. 36–43.

[26] E. Yu, P. Giorgini, N. Maiden, and J. Mylopoulos, Eds., *Social Modeling for Requirements Engineering*. MIT Press, 2011.

[27] I. Sommerville, R. Lock, T. Storer, and J. Dobson, "Deriving information requirements from responsibility models," in *Proc. CAISE*, 2009, pp. 515–529.

[28] R. Strens and J. Dobson, "How responsibility modelling leads to security requirements," in *Proc. NSPW*, 1993, pp. 143–149.

[29] M. Muller, D. Wildman, and E. White, "Taxonomy of Participatory Design Practices," in *CACM*, vol. 36, no. 4, pp. 26-28, 1993.

[30] L. Suchman, "Located accountabilities in technology production," in *Scand. J. Info. Sys.*, vol. 14, no. 2, pp. 91–105, 2002.

[31] I. J. Jureta, A. Borgida, N. A. Ernst, and J. Mylopoulos, "Techne: Towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling," in *Proc. RE*, 2010, pp. 115–124.

[32] M. P. Singh, "Norms as a basis for governing sociotechnical systems," in *ACM Trans. Intell. Sys. & Tech.*, vol. 5, no. 1, pp. 21:1–21:23, 2013.