Designing Sociotechnical Systems with Protos

Fatma Başak Aydemir, Paolo Giorgini, John Mylopoulos

University of Trento, Italy
email: {aydemir, pg, jm}@disi.unitn.it

Abstract. Traditional requirements engineering approaches give little emphasis on the engineering of interactions. Increasingly, the software systems of today are developed independently and integrated with each other, as in sociotechnical systems, where several technical systems support the interaction between the social systems. We propose the Protos methodology as a set of refinement rules which, once applied to the stakeholder requirements, produce multiple specifications as a solution to the requirements problem of sociotechnical systems. We use the London Ambulance System as an example domain.

1 Introduction and Objectives

A sociotechnical system (STS) is a complex interplay of humans, organizations, and technical systems [1]. Examples of STSs include e-tourism domain (an interplay of travelers, hotels, travel agencies, web sites, and other information systems), disaster management systems (police and fire departments, technical infrastructure, civilians, and so on), and health care systems (hospitals, laboratories, doctors, patients, medical equipment, information systems, and others). STSs are designed to satisfy requirements of multiple stakeholders and the satisfaction of the requirements depends not only on the independent performance of the individual subsystems but also on the success of the interaction among the subsystems.

i^* [2] models capture actors and their respective requirements and are proven to be useful for modeling requirements of STSs. However, i^* dependencies do not capture the reciprocal nature of social interactions within STSs. Agent–oriented software development framework Tropos [3] fails to model the mutual interests of agents in social interactions for it is based on i^* [4].

We propose Protos that is inspired by i^* and Tropos for designing sociotechnical systems. Protos aims to provide a requirements modeling language and a set of reduction rules to STS stakeholders so that the stakeholders produce an interaction protocol by applying reduction rules to their requirements as a specification for their requirements problem. A protocol consists of commitments, which are contractual relationships from one subsystems to another. Protos reduction rules refine the STS design problem at hand until the design is finalized by the stakeholders. The contributions of this research are as follows:

- ◇ We provide a requirements modeling language that also captures the essence of interactions in STSs.

- ◇ We specify interactions within an STS as a protocol that consists of commitments.
- ◇ We adapt the classical idea of refinement to provide a set of design reduction rules that systematically produces an interaction protocol and individual subsystem specifications of an STS.

This paper is organized as follows. Section 2 presents the research baseline. Section 3 includes the scientific contributions and Section 4 provides the conclusions, ongoing and the future work.

2 Research Baseline

Requirements and specifications. According to Jureta et al. [5], the requirements problem takes as input a set of requirements R and produces a specification S and a set of domain assumptions A such that (a) $A, S \vDash R$, i.e., if the domain assumptions and the specification hold, then requirements are fulfilled; (b) A and S are consistent, and so is R. This formulation is a variant of the initial formulation of the requirements problem proposed in [6].

Specifications as commitments. Chopra et al. [7] add to this formulation of the requirements problem by treating specifications as commitments for multi-party systems, such as STSs. A commitment is a 4-tuple $C(debtor, creditor, antecedent, consequent)$ where debtor and creditor are agents (including systems/subsystems), antecedent and consequent are propositions and the commitment represented is that the debtor will make the consequent true provided the antecedent holds [8]. A commitment is unconditional if its antecedent is "true".

3 Scientific Contributions

3.1 Protos Requirements Modeling Language

Elements of the Protos Modeling language are as follows. London Ambulance System [9], a real–life case study from the healthcare domain is used to exemplify each concept.

Stakeholder: an autonomous entity that *participates in the design process* of the STS. LAS has nine stakeholders some of which are the service consumers, call takers, ambulance crew, and the radio operator.

Team: a non-empty set of human, organizational, or technical entities that are *designed to be part of the STS*. A stakeholder may appear as a team in the specification, or play multiple teams during the design process. In LAS, the resource allocator and dispatcher may form a team where they closely work together as a sub-unit of the STS.

Proposition: a state of the world in the domain of the STS that is being designed. In LAS domain, 'call received', 'call response time is less than five

seconds', and 'the ambulance is equipped' exemplifies propositions related to the domain. A proposition could be a state of the world that a stakeholder wants to achieve, similar to a goal in i^*. A proposition could also be an assumption about the domain, that is believed to be true by stakeholders.

Requirement: a representation of a state of the world which is expected to be achieved by a team. A requirement is represented as $R(\tau, \pi)$, where τ is a team and π is the proposition that τ desires to achieve. For example, the service consumer's requirement of receiving an ambulance is represented as $R(SC, AmbReceived)$.

Conflict: an irreflexive, symmetric relationship over propositions. $p \oplus q$ means that the proposition p conflicts with the proposition q. $AmbAtStation$ conflicts $AmbSetOut$ for the to cannot be true at once.

Commitment: a contractual relationship between two teams where the debtor team promises the creditor team to bring out the condition when the antecedent holds. A commitment is represented as $C(\tau_1, \tau_2, \pi_1, \pi_2)$ where τ_1 is the debtor team, τ_2 is the creditor team, π_1, is the antecedent, and π_2 is the consequent. $C(CT, SC, incidentReported, incidentTaken)$ is the representation of the commitment between the service consumer (SC) and the call taker (CT), stating that the latter commits the former to take the incident when an incident is reported.

Onus: means that a team takes on the onus of ensuring a proposition. When the service consumer takes the final responsibility of reporting an incident, $O(SC, incidentReported)$ explicitly states its intention and prevents the stakeholders taking further design actions with respect to that proposition.

3.2 Requirements Specification for STSs

The traditional requirements specification problem cannot capture STSs for *(i)* STSs have a heterogeneous structure consisting of social, technical and organizational subsystems, *(ii)* satisfaction of stakeholder requirements heavily rely on the success of the interaction within STSs rather than a execution of a single machine. We reformulate the problem as $A, P, \{O_1, \ldots, O_n\} \vDash R$ where A is the assumptions on the domain, P is the interaction protocol consisting of commitments, and $\{O_1, \ldots, O_n\}$ are specifications for teams $\{\tau_1, \ldots, \tau_n\}$.

3.3 Protos Design Process

Refinement of a design problem p is an incremental improvement of p to p' where a solution for p' is also a solution for p. Protos design process is a continuous refinement of the design space until a solution is created for all requirements, that is, onuses are taken for all requirements. Stakeholder requirements constitute the finite requirements set $\mathcal{R}$. Commitments created during the design process are added into the protocol set $\mathcal{P}$, which is empty at the beginning of the design. Onuses taken by teams are stored in the onus sets, possibly with subscripts for the respective team $\mathcal{O}_n$, for simplicity we use a single onus set $\mathcal{O}$. We use the term *need* for the requirements that are yet to be addressed during the

design episode, and keep all *needs* in the needs set $\mathcal{N}$. Initially, $\mathcal{N}$ includes all requirements. Domain assumptions are kept in the set $\mathcal{A}$.

Need refinement is based on the proposition refinement. For example, $\mathcal{N}$ includes $R(CT, incidentReported)$, that is the call taker has a requirement for incident reports. During the design process, the call taker further specifies this requirement by stating that she requires the address of the incident and the conditions of the people involved. Such refinement is represented as $incidentReported \hookrightarrow address \wedge status$. The refinement clause is added to the assumption set $\mathcal{A}$, initial need is removed from the need set $\mathcal{N}$, and two new needs $R(CT, address)$ and $R(CT, status)$ are added to $\mathcal{N}$ instead.

Commitment introduction: 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 its need for p is satisfied; a commitment is created and added to $\mathcal{P}$. In LAS domain, the service consumer has a requirement for incidents to be entered to the system. Knowing that she cannot satisfy her requirement at runtime, she gets a commitment from another team. $C(CT, SC, incidentReported, incidentTaken)$ states that the call taker commits to enter the incidents to the system if an incident is reported. As a result of the commitment, $incidentTaken$ becomes the need of the call taker for he promises to achieve it. On the other hand, the service consumer adopts *incidentReported* as her need, for it is the antecedent that detaches the commitment and obliges the call taker to satisfy the consequent. The need set $\mathcal{N}$ is updated accordingly. The commitment created is added to the set $\mathcal{P}$. A second type of commitments could be introduced to the design when the antecedent of the commitment is a condition that is assumed to be true, rather than a requirement that is satisfied by another team. When the ambulance crew engages in a commitment to reach a destination in less than 30 mins if there is no traffic, the creditor (service consumer) cannot make sure that the condition 'no traffic' is satisfied. Similar to the previous rule, the creditor adopts the consequent as her need and the created commitment $C(AC, SC, noTraffic, responseTime < 30mins)$ is added to the protocol set $\mathcal{P}$. However, the assumption $noTraffic$ is added to the assumptions set $\mathcal{A}$ instead of being adopted by the creditor. A third way of introducing commitments is cyclic commitments, given there are at least two teams $\tau_1 \dots \tau_n$ where $n \geq 2$, these teams can create a cyclic commitment where τ_i commits $\tau_{i+1 \ mod \ n}$ to bring out its need $p_{i+1 \ mod \ n}$ provided that its need p_i holds. A reciprocal commitment is a special case where $n = 2$. For example, the service consumer and the private ambulance service create reciprocal commitments when they commit to each other to pay the fee when provided the ambulance service and provide the ambulance service when the fee is paid as in $C(SC, PAS, ambulanceService, feePaid)$ and $C(PAS, SC, feePaid, ambulanceService)$.

Subteams creation: A team can create subteams and delegate some of its current needs. For example, when the call taker has needs to send the incident details to the call reviewer, save the incident details to the database, and provide support to the service consumer, she can create a subteam 'call center information system' and delegate the first two need two her subteam. This de-

sign reduction changes the set N, the first two needs of the call taker is removed from the set and re-added as the needs of the subteam 'call center information system'.

Superteams creation: Two or more teams that have needs for p_is where p_is jointly refine p, the teams may create a super team that has a need for p. For example, if there are multiple technical subsystems of the LAS that deal with the smaller pieces of a bigger task, such as communication within the teams, a superteam may be created to deal with the communication within LAS as a whole. As a result of the superteam creation, the needs of the initial teams w.r.t. communication is replaced with the need of the superteam for communication in $\mathcal{A}$.

Commitment refinement: A commitment could be refined in two ways. The first type of the refinement includes the refinement of the proposition part of a commitment, that is either the antecedent, consequent, or both. The service consumer and the call taker may agree to refine $C(CT, SC, incidentReported, incidentTaken)$ to $C(CT, SC, address \wedge status, incidentTaken)$ together. The second type of commitment refinement is to refine the team part of the commitment (debtor, creditor, or both) through creating subteams or superteams. A commitment refinement requires the agreement of the teams that are engaged in the commitment.

Onus: When a team declares that it accepts the final responsibilty,the onus, for a certain need, that need $R(\tau, \pi)$ is removed from $\mathcal{N}$ and the onus $O(\tau_n, \pi)$ is added to $\mathcal{O}_n$. A design process is successfully terminated when $\mathcal{N}$ becomes empty, therefore a solution is specified for all requirements.

The Protos design process is outlined as follows.

1. At the beginning of the process, $\mathcal{N} = \mathcal{R}, \mathcal{A} = \varnothing, \mathcal{P} = \varnothing, \mathcal{O} = \varnothing$
2. For a need n in $\mathcal{N}$ apply one of the design reduction rules described above and update the design configuration $\mathcal{N}$,$\mathcal{A}$,$\mathcal{P}$,$\mathcal{O}$.
3. Apply Step 2. until $\mathcal{N}$ is empty
4. If none of the rules apply at the current design configuration and $\mathcal{N}$ is not empty, the design process fails. Start over with the initial requirements set $\mathcal{R}$.

Due to the social nature of STSs, many of the design reduction rules may include sub-steps such as negotiation or argumentation, especially for the rules that require the agreement of the teams. In a realistic scenario, each stakeholder either represents itself or is represent by an analyst, so the design process is distributed. Initial stakeholders may not take any onuses for their respective requirements at the end, new teams may be added to the design through commitment introductions, subteam, and superteam creations. Additional teams may add their needs to $\mathcal{N}$ when they are introduced, but the initial requirements set do not change during the design. The teams may prefer to go deeper in the design process and specify the technical details, or they may stop the process at a higher level of abstraction, do not discuss the technical details with other teams, and deal with them individually. For the later phase, traditional approaches such as i^* and Tropos can be used.

4 Conclusions, Ongoing and Future Work

In this paper we propose a requirements modeling language and outline a design process for designing STS. We have explained the concepts used in the modeling language, and the design configuration. We provide a set of design reductions and examples of how they can be applied to STS design.

We run a case study with eight participants who are familiar with goal–oriented requirements engineering to model LAS using Protos modeling language and the design process. The participants successfully finalized the designed process where each technical or social subsystems (team) accepted the responsibility for a requirement or for its refinements.

Ongoing work includes refining the modeling language, development of a visual syntax and providing tool support for the designers. We outline future work expanding the framework by adding reasoning for analysis, conflict detection, and conflict handling as well as various types of refinements that enrich the language and the design process.

Acknowledgements. This work was supported in part by European Research Council advanced grant 267856, titled "Lucretius: Foundations for Software Evolution", http://www.lucretius.eu. The authors thank Amit Chopra, Fabiano Dalpiaz, and Munindar P. Singh for their major contributions to the Protos project.

References

1. Dalpiaz, F., Giorgini, P., Mylopoulos, J.: Adaptive socio-technical systems: a requirements-based approach. Requirements engineering **18**(1) (2013) 1–24
2. Yu, E.S.K.: Towards modelling and reasoning support for early-phase requirements engineering. In: Proceedings of 3rd IEEE International Symposium on Requirements Engineering. (1997) 226–235
3. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An agent-oriented software development methodology. Auton. Agents Multi-Agent Sytems **8**(3) (May 2004) 203–236
4. Telang, P.R., Singh, M.P.: Enhancing tropos with commitments. In: Conceptual Modeling: Foundations and Applications. (2009) 417–435
5. Jureta, I.J., Mylopoulos, J., Faulkner, S.: Revisiting the Core Ontology and Problem in Requirements Engineering. RE 2008 (September 2008) 71–80
6. Jackson, M., Zave, P.: Deriving specifications from requirements: an example. Proc. 17th Int. Conf. Softw. Eng. (1995) 15–24
7. Chopra, A.K., Dalpiaz, F., Giorgini, P., Mylopoulos, J.: Reasoning about Agents and Protocols via Goals and Commitments. AAMAS 2010 (2010) 457–464
8. Singh, M.P.: An ontology for commitments in multiagent systems. Artificial Intelligence and Law **7**(1) (1999) 97–113
9. Kramer, J., Wolf, A.L.: Succeedings of the 8th International Workshop on Software Specification and Design. ACM SIGSOFT Softw. Eng. Notes **21**(5) (1996) 21–35