Governance in Sociotechnical Systems: Addressing Stakeholder Concerns In-Band

Munindar P. Singh[1], Emilia Farcas[2], Kartik Tadanki[1], Ingolf Krueger[2], Matthew Arrott[2], Michael Meisinger[2]

[1]North Carolina State University, Raleigh
[2]Calit2, University of California, San Diego

Abstract

We address the challenge of administering sociotechnical systems, which inherently involve a combination of software systems, people, and organizations. Such systems have a variety of stakeholders, each in essence autonomous and contributing distinct concerns. Traditional architectural approaches assume that stakeholder concerns are fixed in advance and addressed out-of-band with respect to the system. In contrast, sociotechnical systems of interest have long lifetimes with changing stakeholders and concerns. We propose addressing stakeholders' concerns in-band during the operation of the system, thus supporting flexibility despite change. Our approach is based on contracts among stakeholders; the contracts are streamlined through a formal notion of organizations. We demonstrate our approach on a large sociotechnical system we are building as part of the Ocean Observatories Initiative.

Keywords

Governance; system administration; policies; model-driven architecture; requirements engineering

Introduction

We define *governance* as the administration of collaborations among autonomous and heterogeneous peers by *themselves*. Because each participant is independently implemented and operated, governance must be captured in terms of high-level normative relationships that characterize the expectations that each participant may place on the others.

Further, our interest lies in sociotechnical systems, which arise in a variety of domains such as scientific investigation, healthcare and public safety, defense and national security, global business and finance. Sociotechnical systems are systems-of-systems (SoS) and their value and complexity arise from the combination of capabilities provided by their (heterogeneous) constituent systems.

An excellent example of such a system is the NSF-funded Ocean Observatories Initiative (OOI), a thirty-year $400 million project [1]. OOI provides novel capabilities for data acquisition, distribution, modeling, planning and control of oceanographic experiments, with the main goal of supporting long-term oceanographic and climate research. The OOI stakeholders include ocean scientists, resource providers, technicians, operators, policy makers, application developers, and the general public.

The OOI presents system requirements that involve supporting thousands of stakeholders, tens of thousands of physical resources such as autonomous underwater vehicles (AUVs), and potentially millions of virtual resources such as datasets. The resources are independently owned and operated. Moreover, OOI facilitates virtual collaborations created on demand to share access to ocean observatory resources, including instruments, networks, computing, storage, databases, and workflows.

Stakeholder concerns in this setting are not simply generic ones such as performance, but how they can benefit from their (and others') resources, monitor their health, control their functioning, and administer their usage. Additional concerns include entering into scientific collaborations, managing resource conflicts, achieving and enforcing accountability of colleagues and staff. Importantly, the specifics can differ for each stakeholder individual or organization, and are influenced by whom the stakeholder interacts with. Such concerns are not readily enumerated during design, especially when dealing with long-lived sociotechnical systems. Not treating them would waste opportunities for improving social and scientific value of oceanographic research. Indeed, this is the current situation and its weakness has motivated the creation of the OOI.

Claims and Contributions

Addressing stakeholder concerns motivates software architecture. In contrast with existing approaches, we consolidate the above interaction-oriented stakeholder concerns into a single metaconcern, *governance*, or how stakeholders administer their collaborations.

Existing IT or SOA approaches treat governance primarily as a slow, ponderous out-of-band activity, whereby stakeholders and negotiate their concerns only during the design of a system, not during its operation. Such approaches are ill-suited for specific concerns arising during collaboration. In contrast, automation is essential to improve the quality (such as the precision, timeliness, productivity, and comprehensibility) and scale of governance. For this reason, we approach governance as a central endeavor carried out *in-band* in a sociotechnical system.

We propose a novel approach that gives first-class status to stakeholders as principals of the resulting system and to their concerns expressed via contracts and policies. Our approach is compatible with traditional approaches, and thus helps leverage existing tools and experience where appropriate.

Architectural Treatment of Governance

User Stories

We describe important OOI use cases for governance, which highlight the autonomy of the participants and the business relationships among them.

Collaboration. The stakeholders of OOI include research scientists or investigators as well as educators from middle and high schools. Consider a situation where a teacher in a school near Chesapeake Bay would like to present some information about the students' local environment. This data could be as simple as acidity levels in the Bay. Clearly, the teacher would need to access data that a researcher with the appropriate sensors would have gathered. The researcher may have entirely different interests from the teacher; for example, she may be interested in multiyear trends. To this end, the researcher would participate in a resource-sharing community where she would have shared the data streams being generated by her sensors. The teacher would also authenticate with OOI, discover the appropriate community, and enroll in it. Therein the teacher would discover the desirable data stream and extract the information he needs.

Affiliation. The stakeholders of OOI include not only investigators but also research institutions and laboratories. Two institutions may decide to share their resources on a reciprocal basis, and thus enter into a suitable contract. A researcher at one of those institutions would be able to discover with which institutions her institution is affiliated. She would then be able to access an affiliate institution and further discover a research laboratory based at the second institution. Lastly, she would be able to take advantage of resources belonging to the laboratory. Either institution may decide to end the affiliation but even its exit could be subject to the existing contract, e.g., that ongoing experiments not be aborted.

Basic Concepts

Our conceptual model is centered on the concept of *principal*. Principals include users, resources, and organizations (termed *Orgs* in our model). Each principal possesses a unique identity within OOI. Governance is achieved through interactions among principals: realized through their local policies and constrained by their contracts with each other. Each principal may adopt roles in one or more Orgs. In essence, each role corresponds to a contract between a principal who adopts it and the Org (also a principal). This contract constrains the further interactions between two principals present in the same Org. In general, a contract may arise as the result of a successful negotiation or may be implicitly imposed due to the parties adopting complementary roles in the same Org. Each contract references an Org that serves as its context.

Each principal is represented in the computational system by an agent. Each principal's agent supports bookkeeping regarding the contracts in which the principal participates. The agent helps determine if its principal is complying with its contracts and if others with whom it deals are complying as well. The agent continually tracks the state of each contract by updating the state for each observable action, such as sending or receiving a message (including making an observation of the environment).

Orgs serve multiple purposes in our architecture, specifically providing a backstop for contracts, a locus for identity, and a venue to share resources. Each Org defines the rules for adopting each of its roles. Joining an Org means adopting at least one role in that Org. Adopting a role means accepting the rules of the Org for that role. Thus, we understand enrollment in an Org as involving the creation of a contract and treat the subsequent interactions of the participants as arising within the scope of the given Org. An example of enrollment is someone joining eBay; an example of additional contracts is when two eBay members carry out a transaction. The members are subject to eBay's rules such as accepting the price announced by eBay at the end of an auction.

The above interactions, including enrollment, inherently involve the creation and manipulation of contracts and can potentially be operationalized in multiple ways. For example, for enrollment, (1) the prospective enrollee may request membership; (2) the prospective enroller may invite the enrollee; (3) a third party may introduce the enrollee and enroller; or (4) a third party may require the enrollee and enroller to carry out the enrollment. Such flexibility facilitates separating stakeholder concerns from each other and from the implementation, thereby improving how stakeholders comprehend the architecture and enhancing the confidence they can place in it.

Each principal applies its own policies to determine what actions to take. Thus, a principal can decide whether to adopt a role in an Org and, conversely, the Org can decide whether to admit a principal to a role. Each principal's decisions are subject to constraints such as the requirements imposed by the roles that it has adopted.

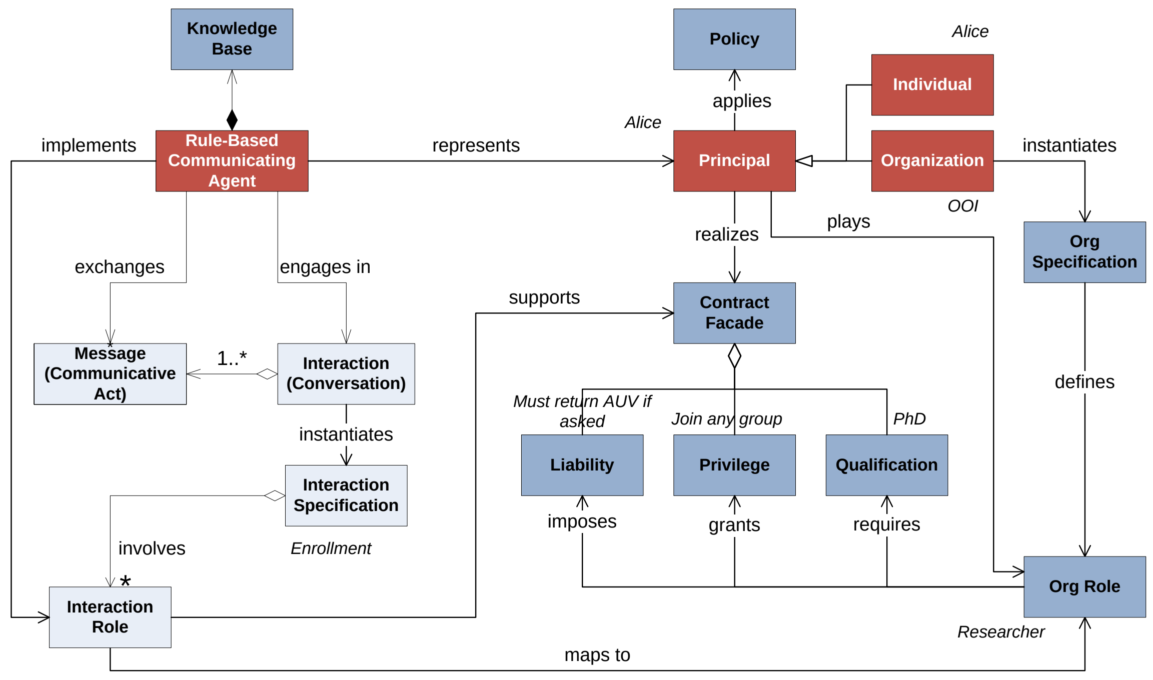

Figure 1 Overview of Governance Model

The model from Figure 1 relates an Org specification with a contract. Each clause of a contract involves two or more Org roles. In effect, each Org role partitions its view of the relevant parts of the contract. We model the role-relevant parts of each contract as consisting of three components, assembled into a *contract façade*:

- *Qualifications*, which specify eligibility requirements for a principal to take on a role. For example, a professor must have a university identity to join a PhD committee.
- *Privileges*, which specify what authorizations and powers a principal gains in adopting the role. A professor as committee member is authorized to review the student's lab notebook and empowered to determine if the student passes.
- *Liabilities*, which specify what a principals becomes subject to in adopting the role. A committee member must attend a PhD defense.

Each principal applies its policies, to determine whether to enroll, potentially to take advantage of its privileges, and ideally to satisfy its liabilities. In general, we cannot guarantee compliance, but we address compliance in two main ways:

- Conservatively, ensure that the actions taken by a principal are compliant. This can work where the principal is not autonomous and heterogeneous. We can subject the principal to a guard that allows only the policy-compliant (attempted) actions of the principal to proceed.

- Optimistically, recognize that a principal may proceed as it would, but detect and handle noncompliant behavior. We can accomplish detection either by introducing a monitor in the architecture or through the principals monitoring each other. We can respond to detected violations by escalating them to the nearest scoping Org.

Contract Conceptual Model

Our contract conceptual model has roots in recent research into software engineering and agents [7]. We model a contract recursively as a set of contracts with the recursion bottoming out as a set of clauses (see Figure 2). Our taxonomy of clauses is based on the study of real-life contracts:

- The *Main* Clauses deal with the main "business" reason for having a contract in the first place. Of particular interest are the following types:
    - A Service clause states the kind of service a party of the contract shall provide.
    - A Quality of Service clause states additional requirements. Example: will provide a 128kbps flow rate.
- The *Scoping* Clauses specify the purpose and scope of a contract. These are crucial in typical business contracts because of their potential effect on legal rights of the parties involved. We expect these might be rather straightforward in most OOI governance settings, although the main OOI membership would have a description of the scoping requirements for when users sign up for an OOI account.
- The *Visibility* Clauses deal with how much access the parties have to the internal implementations of each other. Computer scientists would naively treat the parties to a contract as black boxes. However, this is usually not the case for business contracts of any importance or complexity. In general, each party would rely upon visibility clauses to make sure that the work product is of an adequate quality, that the effort is robust, and does not violate any laws or regulations to which the parties might be subject.
    - An *Implementation* clause imposes restrictions on how a service is to be implemented, typically in a manner that would not be apparent from the service or quality of service clauses. Example: the data must be archived in at least three separate physical stores.
    - A *Monitoring* clause provides privileges to monitor the progress of a service being delivered, verify its quality, audit the books for usage, or examine the implementation.

- The *Normative* Clauses deal with matters that are important to the regulations and policies that apply on the interactions among the parties to the contract. Thus, the normative clauses are of special importance to our proposed use of contracts for governance.
  - A *Prohibition* clause imposes restrictions upon the services that each party may perform for another. Example: the information being provided by the data stream may only be used for noncommercial purposes.
- The *Resolution* Clauses deal with how to respond to failures in a contract, including the possibility of sanctions (of violators) and compensations (by violators). The most likely forms of sanctioning will be through the somewhat amorphous means of reputation and via escalation of complaints to the Org that provides the scope for a contract. An Org may sanction a principal that it judges to be malfeasant by ejecting the principal and possibly escalating a complaint further. At the top level, OOI may eject and disbar a malfeasant principal.

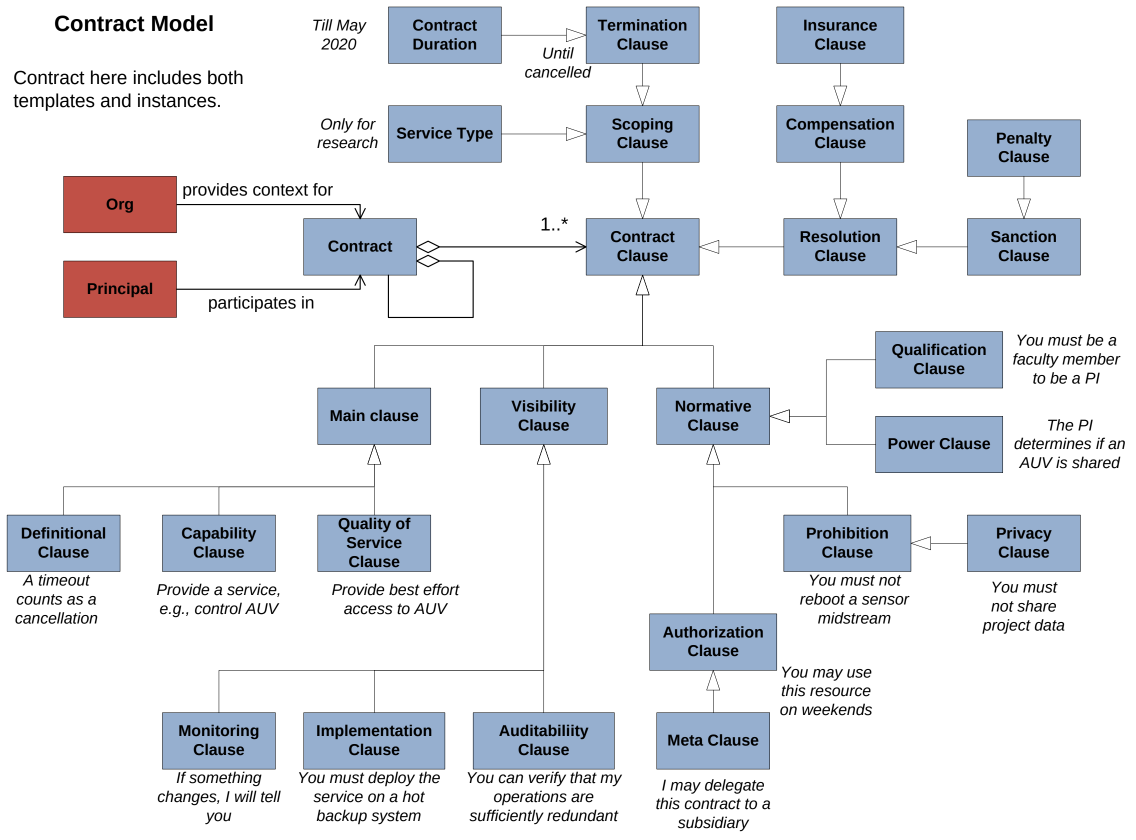
Contract Model
Contract here includes both templates and instances.
Till May 2020
Contract Duration
Until cancelled
Termination Clause
Insurance Clause
Only for research
Service Type
Scoping Clause
Compensation Clause
Penalty Clause
Org
provides context for
Contract
1..*
Contract Clause
Resolution Clause
Sanction Clause
Principal
participates in
Qualification Clause
You must be a faculty member to be a PI
Main clause
Visibility Clause
Normative Clause
Power Clause
The PI determines if an AUV is shared
Definitional Clause
A timeout counts as a cancellation
Capability Clause
Provide a service, e.g., control AUV
Quality of Service Clause
Provide best effort access to AUV
Prohibition Clause
You must not reboot a sensor midstream
Privacy Clause
You must not share project data
Authorization Clause
You may use this resource on weekends
Monitoring Clause
If something changes, I will tell you
Implementation Clause
You must deploy the service on a hot backup system
Auditabiliity Clause
You can verify that my operations are sufficiently redundant
Meta Clause
I may delegate this contract to a subsidiary

Figure 2 Contract Model

Contract Lifecycle

The lifecycle of contracts includes the following key phases depending on the actions of two or more participants [7]:

- *Negotiation* or when the contract comes into being. The contract is created through a series of communicative acts. A negotiation is initiated when one party proposes to another party. The parties may make zero or more counterproposals to each other. The negotiation ends when one of the parties rejects the last proposal or all the parties accept. It is presumed that the proposer accepts its proposal, so in two-party settings, only one party (the recipient of the last propose or counterpropose) needs to accept.
- *Execution* or when a contract is activated. Contracts often include standing commitments and activated only when there is demand for a service. Example: A researcher agrees to provide a dataset when another one requests it. But there is nothing for anyone to do until the first request comes in. The service requests may require actions by more than one party in order to be fulfilled and, thus, cannot be accurately modeled in client-server terms.
- *Monitoring* or determining progress during execution. In almost all cases, the parties to the contract have to agree that the desired service was performed. Thus, they must assess the outcomes if only to declare success. However, we think of more elaborate forms of monitoring of the contract execution, which would be specified by the monitoring clauses in the contract (see the contract model).
- *Resolution* or addressing the violation of any contract clauses during contract execution. This involved applying the resolution clauses from the contract specification—imposing penalties or offering compensations to the aggrieved parties, including potentially renegotiating the terms or otherwise injecting additional contract clauses. Upon successful resolution, the contract resumes execution; otherwise it terminates in failure.
- *Termination* or bringing a contract to a closure, after either fulfillment or failure. A terminated contract may be archived or analyzed. If a contract ends in failure, one or more of the parties may escalate matters: by complaining to a designated authority such as a higher Org.

Case Study

OOI enables its primary stakeholders (scientists) the opportunity to seamlessly collaborate with other scientists across institutions, projects, and disciplines and to access and compose resources as needed.

To address complexity, mitigate risks, and accommodate requirement changes, OOI uses a spiral development process, a variant of the Incremental Commitment Model (ICM) [3].

ICM includes iterative development cycles focusing on incremental refinement of system definition and stakeholder commitment and satisfaction. We have adopted selected architectural views from the Department of Defense Architecture Framework (DoDAF) [5] to document the OOI architecture.

OOI resources are distributed both physically and virtually among different organizations, each with their own policies for resource access and data delivery or consumption. We model OOI itself as an Org that is the highest scope for all OOI users and their interactions. The OOI Org serves as the root Org for the identities for all OOI principals and helps monitor and enforce contracts among them.

Figure 3 illustrates the use case where two research organizations (each an Org) form an affiliation relationship with each other. Both Org A and Org B are what we term *resource-sharing Orgs*, and define two main roles: *owner* (of a resource) and *user* (of a resource). Each principal who adopts *owner* can contribute its resources to the Org, so those resources can be discovered by any principal who adopts the role *user*. In addition, to form affiliations, each Org supports additional roles capturing the affiliation relationship. These roles are *affiliateOrg* to capture the clauses for the affiliated community, and *affiliateMember* to capture the clauses for the members of the affiliated community, which could have weaker rights than its own members.

The affiliation contract between Orgs propagates to their respective members. As a result, a member of Org A can discover services offered by members of Org B. Once it has discovered such services, it may negotiate with and engage them as appropriate.

Our notation is similar to message sequence charts in terms of having a swim lane for each principal. However, instead of messages, we use horizontal lines to show joint (governance) actions that create or modify relationships among the (two or more) parties whose lifelines they connect. Any temporal order requirements are captured via the dashed arrows that connect some pairs of the horizontal lines. In general, the parties would realize a governance interaction such as enrollment by exchanging multiple messages, e.g., propose, counterpropose, accept, or reject.

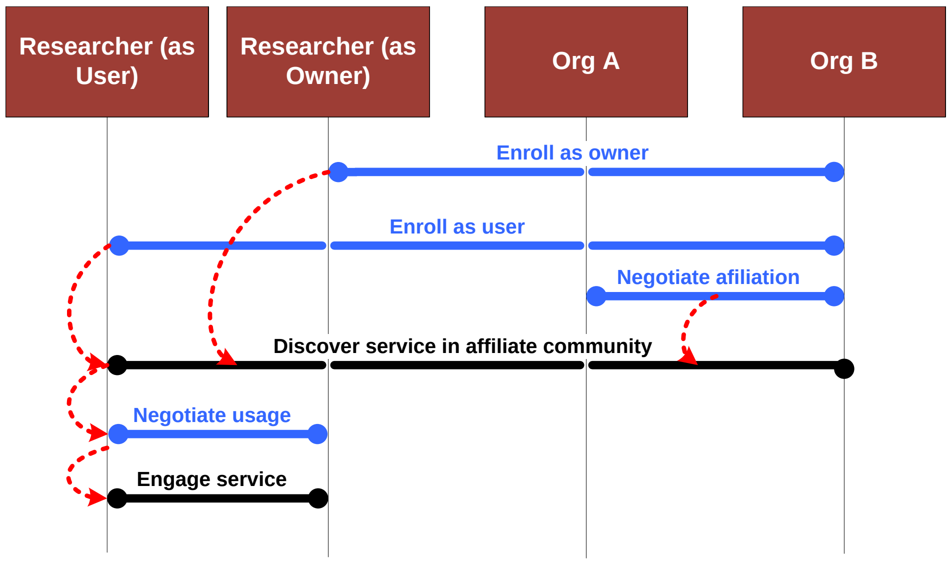

Figure 3 Governance of resource sharing across affiliated Orgs

Realizing Governance

Rich Services. We apply the Rich Services architecture [6], a type of SOA that provides decoupling between concerns and allows for hierarchical service composition. Rich Services is a logical architecture that can be mapped to possible deployments such as Enterprise Service Buses or multiagent systems.

For the affiliation use case in OOI, each Org and the User itself are modeled as a Rich Service within the root OOI Rich Service. Infrastructure Services include identity and policy management, logging of all conversations and actions, as well as repositories for the community specification and the contracts already established with other parties. Each Rich Service has its local policies and a local representation of the contracts it participates in.

The Rich Services architecture provides a clear separation between the business logic and its external constraints, supporting their composition at the infrastructure level through specialized interceptors. When requirements change during the lifetime of the system, they often change regarding to policies and not to core services; therefore, the decoupling between them allows to update Infrastructure Services without modifying the services that are composed.

Agents and multiagent systems. A specific implementation of governance is a rule-based communicating agent, which maintains the applicable rules and information about the state of the world and any ongoing interactions in a knowledge base. An agent represents a principal in an Org as a locus of autonomy and identity. We have prototyped such an agent using an agent platform (specifically, Java Agent Development Framework (JADE)) and a rule engine (specifically, Java Expert System Shell (JESS)). An agent platform provides a container for the execution of agents, communication infrastructure to enable agent communication, and directory services. A rule engine maintains and applies the facts and rules for an agent and, thus, enables reasoning and reaction.

Rules led to a simple implementation where an agent loads the rules corresponding to each role that it adopts. The rules are generated from the contract specifications for which we developed a domain Specification Language; its constructs are based on properties and predicates.

Evaluating Claims and Contributions

We attribute the power of our architectural treatment of governance to the following main principles that it respects.

- Centrality of organizations in modeling communities; modeling the OOI itself as an entity; specifying rules of encounter; monitoring contracts; sanctioning violators.
- Autonomy of stakeholders; representing stakeholders as runtime entities (agents) that apply autonomous policies and are subject only to applicable organizational rules of encounter.
- Emphasizing normative relationships and modeling them explicitly to make them easy to inspect, share, and manipulate; accommodating openness of the system by recognizing that autonomous parties may violate rules of encounter and, thus, may need enforcement ex post facto, such as via sanctions.

In the OOI, policies specified in Org contracts govern the circumstances under which resources can be discovered, accessed, and utilized. In the example, we considered two classes of stakeholder roles: user and owner of a resource. The user is concerned with accessing a resource, without facing any hidden obligations. The owner is concerned with providing resources (with spare capacity) to expand impact of the resources on others and to treat the resources as a basis for negotiating value in exchange.

Our governance approach addresses stakeholder (user and owner) concerns as follows:

- The resource sharing community provides access to needed resources and clarifies what restrictions are imposed on the user as a result; guarantees that the user will not subject to the whims of the resource owner once the user begins an allowed interaction with a resource.
- The affiliation community expands resource sharing to external organizations and provides access to remote resources on a reciprocal basis.

- The user and owner can negotiate detailed contractual terms beyond the clauses imposed by being members of a community
- The user and owner can accommodate changing needs, renegotiate the contract, or may decline to renew a contract
- In deployment, policies are separated from the business functionality, allowing them to be changed easily over time according to stakeholder needs.

Our work builds upon methodologies such as Model-Driven Architecture (MDA) and goal-oriented requirements engineering, and goes beyond them by providing a systematic treatment of governance from the modeling level to implementation.

Addressing the inherent complexities of sociotechnical systems involves going beyond traditional Service-Oriented Architecture (SOA), specifically in accommodating multiple ownership domains [4]. Following [8], we view services as analogous to real-life services, not computational objects. We identify principals as the participants in service engagements described in terms of the contractual relationships, and define patterns on the creation, propagation, and manipulation of such contracts.

Our approach coheres with recent advances in goal-oriented methodologies, specifically Tropos [9]. Tropos includes concepts of actors with goals and capabilities, which agrees with our conceptualization. Tropos emphasizes the goals of the actors whereas we emphasize their contracts and would capture their goals both in what contracts they enter and how they choose to perform them.

Recently, ultra-large-scale systems (ULSSIS) have garnered attention [2]. ULSSIS inherently involve multiple stakeholders who not only use the system, but may also contribute resources, form virtual communities, and determine the rules that govern their interactions. We understand sociotechnical systems to be ULSSIS. Our approach applies naturally to ULSSIS because it dynamically captures stakeholder concerns by (1) defining patterns of interaction based on Orgs; (2) enabling stakeholders to select roles in desirable Orgs; and (3) supporting the specification and application of policies potentially customized to each stakeholder.

References

1. Ocean Observatories Initiative – CyberInfrastructure (OOI-CI) http://www.oceanobservatories.org/
2. Northrop, L., Feiler, P., Gabriel, R.P., Goodenough, J., Linger, R., Longstaff, T., Kazman, R., Klein, M., Schmidt, D., Sullivan, K. and Wallnau, K. Ultra-Large-Scale Systems: The Software Challenge of the Future. Software Engineering Institute, 2006.
3. Boehm, B., Lane, J.: Using the Incremental Commitment Model to Integrate System Acquisition, Systems Engineering, and Software Engineering. CrossTalk, 19(10):4–9, 2007.
4. MacKenzie, C., Laskey, K., McCabe, F., Brown, P., Metz, R.: OASIS Reference Model for Service Oriented Architecture 1.0, (2006), http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf

5. DoD Architecture Framework v1.5, Department of Defense, 2007. http://www.defenselink.mil/cio-nii/docs/DoDAF_Volume_I.pdf
6. Arrott, M., Demchak, B., Ermagan, V., Farcas, C., Farcas, E., Krueger, I.H., Menarini, M.: Rich Services: The integration piece of the SOA puzzle. *Proc. IEEE International Conference on Web Services (ICWS)*, pp. 176–183, 2007.
7. Nir Oren, Sofia Panagiotidi, Javier Vazquez-Salceda, Sanjay Modgil, Michael Luck, Simon Miles, Towards a Formalisation of Electronic Contracting Environments. *Proc. Workshop on Coordination, Organizations, Institutions and Norms*, LNCS 5428, pages 156–171, Springer-Verlag, 2009.
8. M. Singh, A. Chopra, N. Desai, Commitment-Based Service-Oriented Architecture. *IEEE Computer*, 42(11): 72–79, 2009.
9. P. Bresciani, A, Perini, P. Giorgini, F. Giunchiglia, J. Mylopoulos, Tropos: An Agent-Oriented Software Development Methodology, *Journal of Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.