The Internet of Things and Multiagent Systems

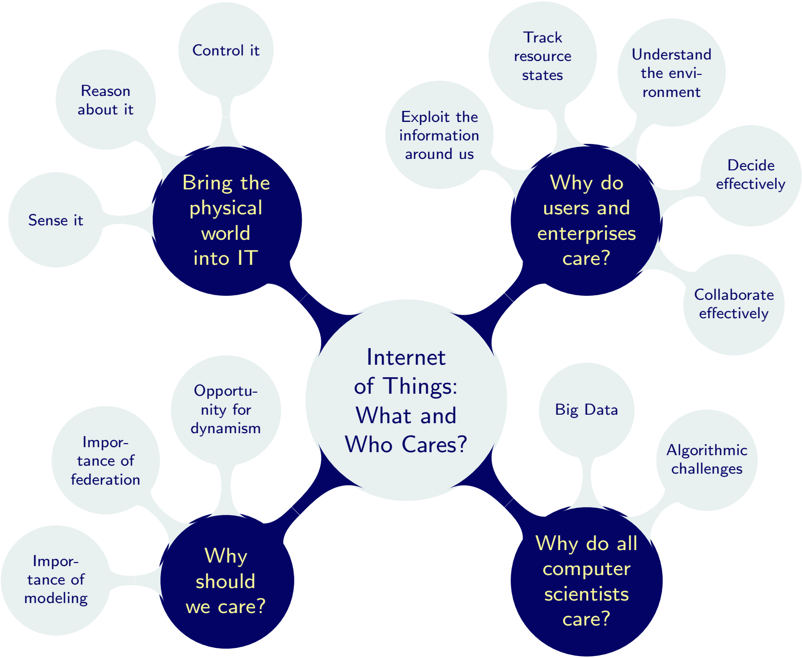
Tutorial

Munindar P. Singh and Amit K. Chopra

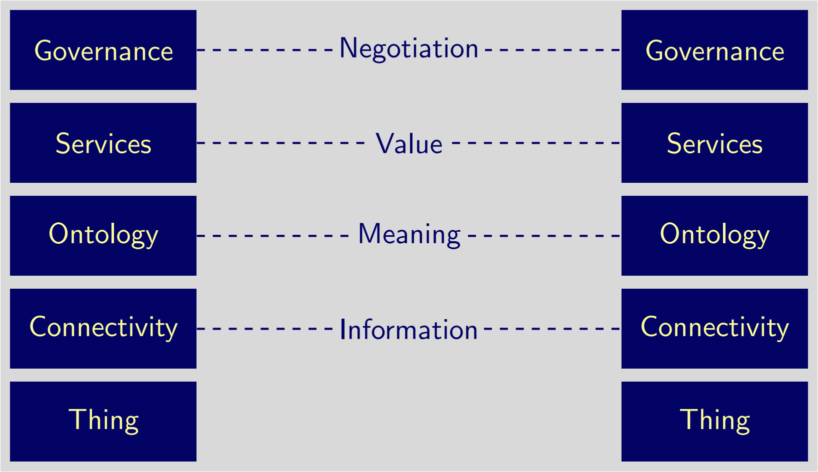
North Carolina State University
Lancaster University

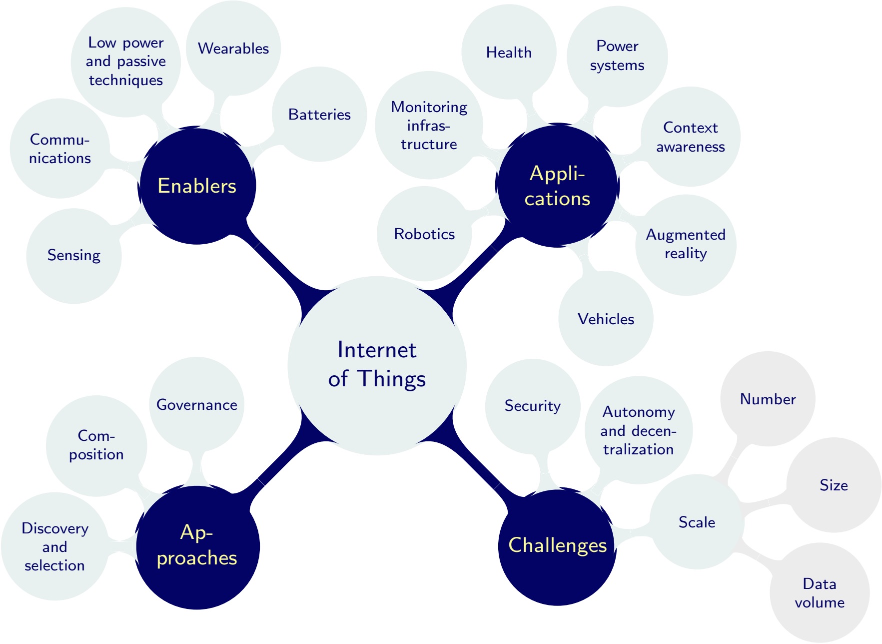
June 2015

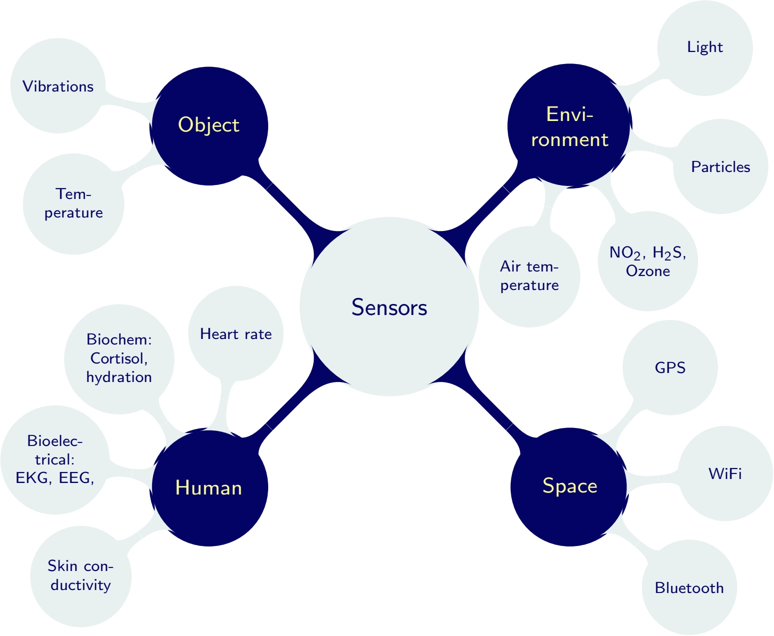

Scope and motivation of this tutorial
An introduction to a new area
From a sociotechnical perspective
Not a programming guide
Focus on research challenges
With a hope to help build a research community


Internet of Things: What and Who Cares?
Bring the physical world into IT
Sense it
Reason about it
Control it
Why do users and enterprises care?
Exploit the information around us
Track resource states
Understand the environment
Decide effectively
Collaborate effectively
Why should we care?
Opportunity for dynamism
Importance of federation
Importance of modeling
Why do all computer scientists care?
Big Data
Algorithmic challenges

IoT Federation Levels




Internet of Things
Enablers
Low power and passive techniques
Wearables
Batteries
Commu-nications
Sensing
Appli-cations
Monitoring infras-tructure
Health
Power systems
Context awareness
Robotics
Augmented reality
Vehicles
Ap-proaches
Governance
Com-position
Discovery and selection
Challenges
Security
Autonomy and decen-tralization
Scale
Number
Size
Data volume


Sensors
Object
Vibrations
Tem-
perature
Envi-
ronment
Light
Particles
NO_2, H_2S,
Ozone
Air tem-
perature
Human
Heart rate
Biochem:
Cortisol,
hydration
Bioelec-
trical:
EKG, EEG,
Skin con-
ductivity
Space
GPS
WiFi
Bluetooth

Core: Connectivity

Combination of low and high-bandwidth

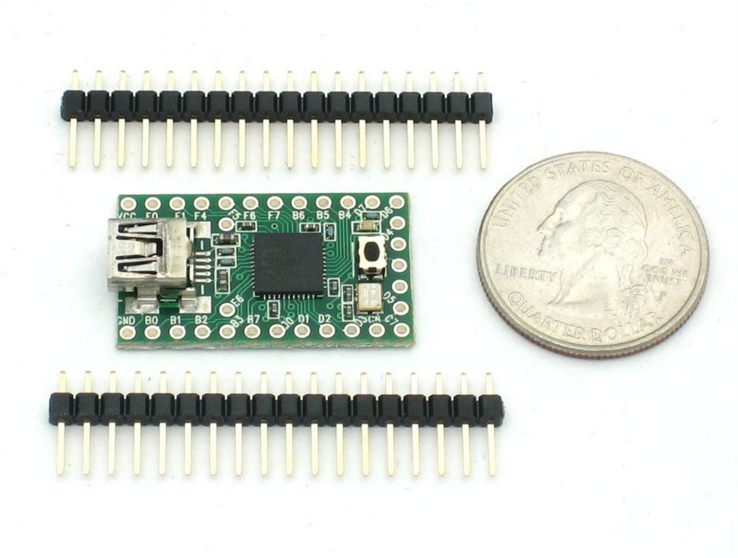
- Low-bandwidth connectivity
  - Between RFID tags and readers
  - Between sensors and base stations
- High-bandwidth
  - Wireless, with aggregator nodes such as smartphones and Arduino devices
  - Broadband, from aggregator nodes to store data in the cloud

Core: Passive and Low Power Technologies

Many dimensions

- Passive (batteryless)
  - No power
  - Require a proximal (within ∼10 cm) reader
- Active, battery-based
  - Can send information
  - Can last years but need battery replacement
- Active, self-powered
  - Harvest power from the
    - Environment, e.g., solar
    - Human body, temperature differential
    - Human body, movement
  - Enough power to transmit to a local, e.g., on-body, hub
    - Low-power radio: $10^{-7}W$ (versus Bluetooth: $10^{-3}W$)
    - Human body, temperature differential
    - Human body, movement
  - Can potentially last "forever"

Microcontrollers for Programmably Controlling Actuators



© Adafruit, New York

Constrained Node Networks

Data and communication

Class	RAM (data)	Flash (code)	Access
C0	$\ll$ 10 KiB	$\ll$ 100 KiB	Via others
C1	10 KiB	100 KiB	Constrained protocols, e.g., CoAP
C2	50 KiB	250 KiB	Capable, but prefer lightweight

Credit: IETF RFC 7228, May 2014, ⟨http://tools.ietf.org/html/rfc7228⟩

Constrained Node Networks

Power

Name	Energy limitation	Example power source
E0	Event	Event-based harvesting
E1	Period	Battery: recharge or replace
E2	Lifetime	Nonreplaceable battery
E9	None	Mains power

Credit: IETF RFC 7228, May 2014, ⟨http://tools.ietf.org/html/rfc7228⟩

Constrained Node Networks

Communication strategy

Name	Energy limitation	Strategy
P0	Normally off	Reattach on demand
P1	Low power	Appears connected: high latency possible
P9	Always on	Always connected

Credit: IETF RFC 7228, May 2014, ⟨http://tools.ietf.org/html/rfc7228⟩

Power Effectiveness: Processors

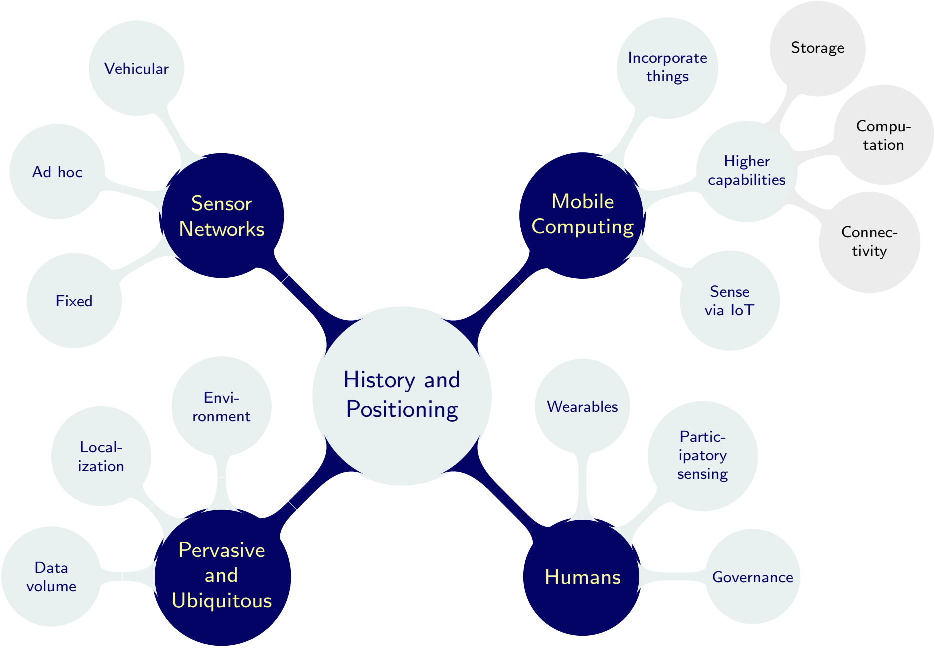
	Prototype	**Commercial 1**	**Commercial 2**
	NCSU ASSIST	EnOcean STM 31xC	Semtech SX1282
Voltage	0.5V	2.1V	1.0V to 1.6V
Processor	16b MSP430	Custom	8b CoolRISC
Power consumption	$< 1\mu W$@$200kHz$	5.1mA @ 3–5V	1.2μW @ 32kHz; 600μW typical
Power harvesting	RF, solar thermoelectric	Yes	No

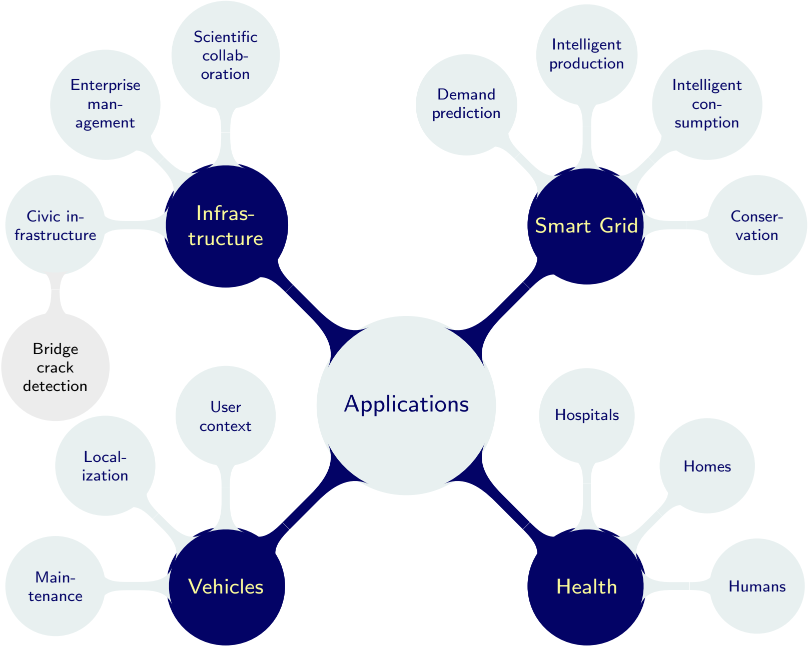
Credit: ASSIST Center, NCSU, ⟨http://assist.ncsu.edu/⟩

Power Effectiveness: Radios

	Prototype	Commercial 1	Commercial 2
	NCSU ASSIST	EnOcean STM 31xC	Semtech SX1282
Voltage	0.5–1.0 V	2.1V	1.0V to 1.6V
Transmission power	$6\mu W$ @ 200kbps	$30mW$ @125 kbps	$\sim 40mW$
Reception power	$200\mu W$ WBAN $120nW$ @ 12.5 kbps WU	$40mW$	$\sim 12mW$

Credit: ASSIST Center, NCSU, ⟨http://assist.ncsu.edu/⟩


History and Positioning
Sensor Networks
Vehicular
Ad hoc
Fixed
Mobile Computing
Incorporate things
Higher capabilities
Storage
Computation
Connectivity
Sense via IoT
Pervasive and Ubiquitous
Environment
Localization
Data volume
Humans
Wearables
Participatory sensing
Governance


Applications
Infras-tructure
Scientific collab-oration
Enterprise man-agement
Civic in-frastructure
Bridge crack detection
Smart Grid
Demand prediction
Intelligent production
Intelligent con-sumption
Conser-vation
Vehicles
User context
Local-ization
Main-tenance
Health
Hospitals
Homes
Humans

Vehicle Sensors

© Munindar P. Singh

Vehicle Actuators

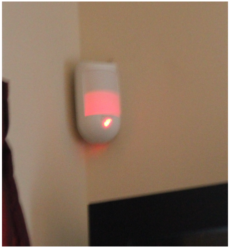
Over the air modification of Tesla chassis elevation

US Government Work
https://www.flickr.com/photos/departmentofenergy/9522268517

Internet of Homes and Hotels?

© Munindar P. Singh

© Nest.com



© Munindar P. Singh

Internet of Trails

Crabtree Lake Trail near Raleigh, North Carolina

Internet of Oceans: Global Hybrid Profile Mooring Launch

© Tom Kleindinst, WHOI

Internet of Oceans: Glider Being Launched

Smart Grid

© Munindar P. Singh
EU project Scanergy's demo setup at AAMAS 2015
http://scanergy-project.eu

Internet of Lakes

Crabtree Lake near Raleigh, North Carolina

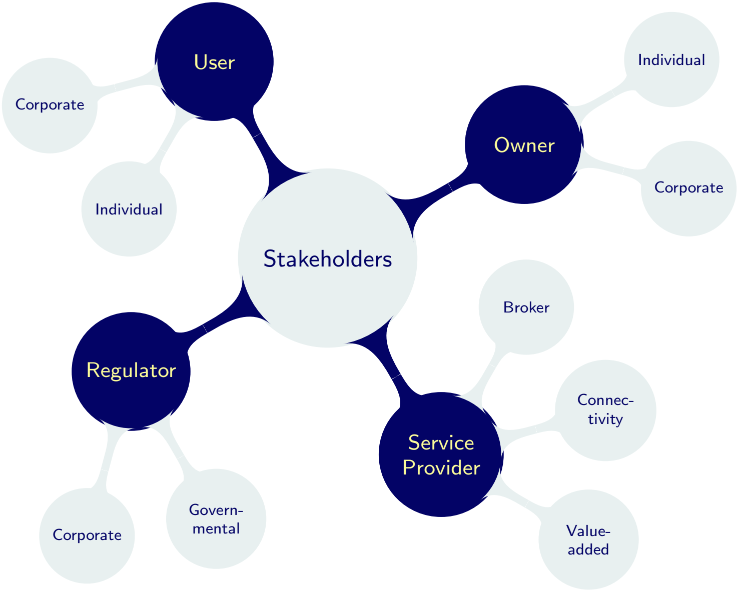
Internet of Dogs

Tagged stray dog in Istanbul

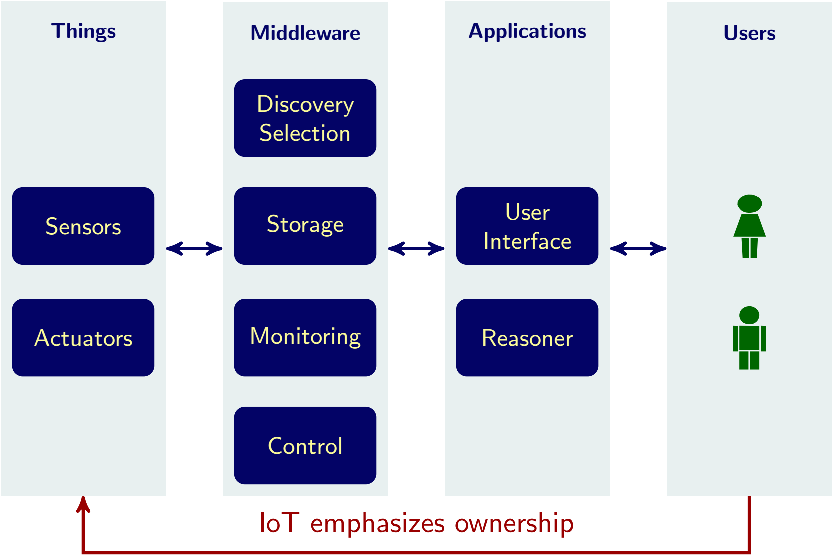
© Munindar P. Singh

Internet of Monuments?

From the Acropolis; appears to be a visual reader, not an IoT sensor

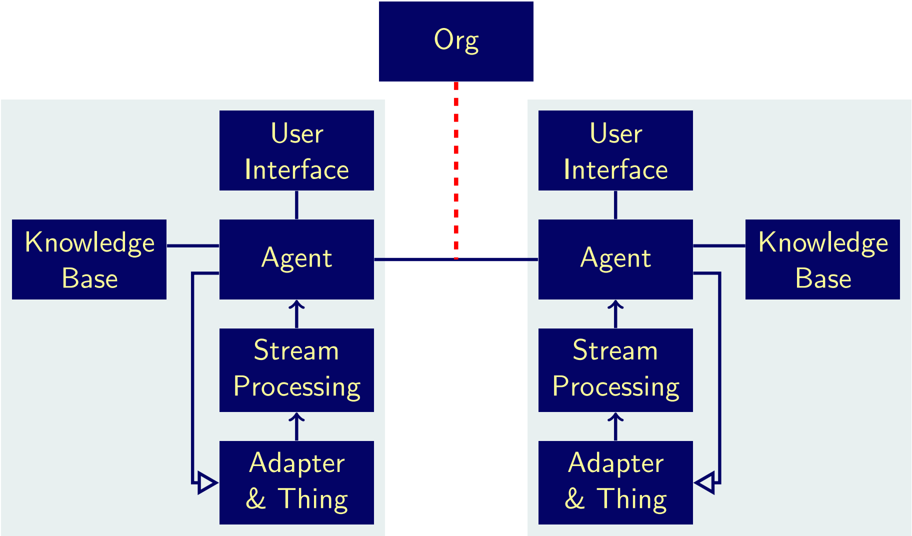

Stakeholders
User
Corporate
Individual
Owner
Individual
Corporate
Regulator
Corporate
Govern-
mental
Service
Provider
Broker
Connec-
tivity
Value-
added

Main Architectural Elements

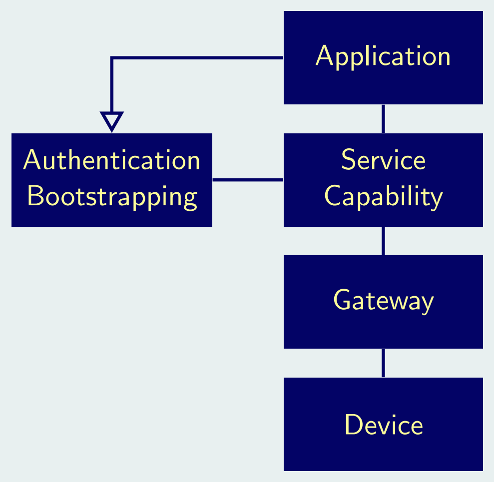


Decentralization Calls for a Multiagent Architecture

Each agent represents an autonomous party, contains a reasoner; interact within Org



Emerging Standards Support Decentralization in Principle



Based on ETSI draft TS 102 690, May 2014, ⟨http://docbox.etsi.org/smartM2M/Open/Latest_Drafts/⟩

Comparing Traditional Web and IoT Protocols

From ReST to Constrained RESTful Environments (CoRE)

HTTP	Application layer	MQTT CoAP
TCP	Transport layer	UDP
IPv4 IPv6	Network layer	6LoWPAN Zigbee
Ethernet WiFi	Data link layer	IEEE 802.15.4
Information Resource		Thing

Protocol: MQTT

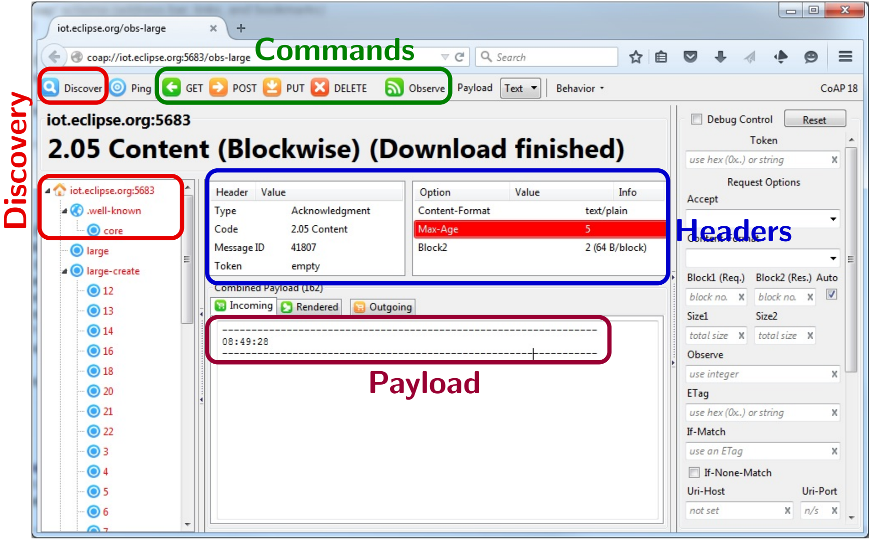
Née Message Queue Telemetry Transport

- OASIS standard as of November 2014, v3.1.1
- Focused on machine-to-machine communications
- Asynchronous
- Data parsimonious
- Small set of primitives, including
  - Publish
  - Subscribe
  - Create and cancel subscriptions
  - Configure “last will and testament” (LWT) notifications
- Detect disconnections without a DISCONNECT message
  - Send configured LWT notifications
- Three QoS levels (separately specified by sender and receiver)
  0. At most once; message: PUBLISH; no message ID needed
  1. At least once; resend with DUP bit until PUBACK received
  2. Exactly once; server stores, then forwards to receivers

CoAP: Constrained Application Protocol

- IETF RFC 7252, June 2014
- Asynchronous
- Data parsimonious
- Supports URIs
- Supports resource discovery from server
- HTTP-like verbs: GET, PUT, POST, DELETE
- Small set of request and response types
- Communication patterns
  - Caching
  - Block transfer of large content
- QoS support (message types)
  - Confirmable CON: require acknowledgment
  - Nonconfirmable NON: do not require acknowledgment
  - Duplicates to be ignored by receiver

CoAP Tooling: Copper Plugin for Firefox

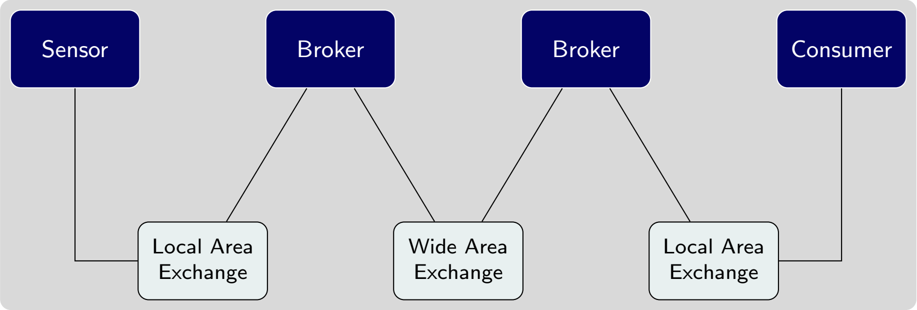


Credit: Matthias Kovatsch, ⟨http://people.inf.ethz.ch/~mkovatsc/copper.php⟩

AMQP: Advanced Message Queuing Protocol

- OASIS standard as of October 2012, v1.0
- Protocol—distinguish from an API such as JMS
- Decouples communications from destination address
- Long-lived conversations
- Variety of communication patterns
  - Intercept
  - Delegate
  - Multiplex and demultiplex
- Upcoming improvements
  - Traffic flow and QoS
  - Decentralized deployment and governance
  - Multiple underlying protocols

AMQP Exchange Space Abstraction



Based on work with Ocean Observatories Initiative, UCSD Scripps

XMPP: Extensible Messaging and Presence Protocol

Originally meant for collaboration and content sharing

- Adapted and enhanced for IoT via XMPP Extension Protocols
- XEP-0323: Internet of Things – Sensor Data, v0.4, March 2015
- Describes protocols and data formats for variety of needs
  - Request sensor reading and responses thereto
  - Requests with multiple responses
  - Requests to multiple things
  - Discovery: what *features* (including services) a thing supports
    - Different from discovery of a thing
- Specification of quality of data value (termed QoS), e.g.,
  - Missing, estimated, manually read, delayed, invoiced, . . .

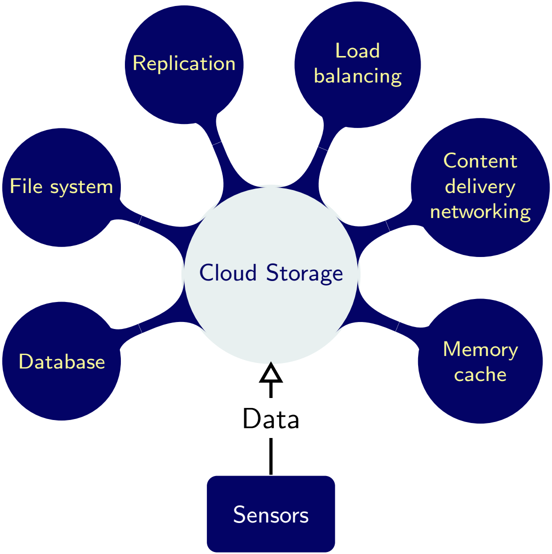
XMPP: Extensible Messaging and Presence Protocol

- XEP-0347: Internet of Things – Discovery, v0.3, November 2014
- Uses JID or *Jabber ID* or an XMPP address for various resources
  - Registries
  - Provisioning servers
  - Things
- Specifies protocols and data formats for
  - Finding a registry
  - Registering a thing: a thing registers itself
    - Specifies secret data for registration
    - May provide the same data via a sticker (QR-code) on a physical thing
    - Public things remain searchable after being claimed
  - Owner claiming a thing (by supplying the secrets)
    - Only claimed, currently owned, things can be searched
  - Removing a thing from a registry
  - *Disowning* or relinquishing a thing

Cloud Storage

Eliminate need to set up complex storage services from scratch

- On demand
- Variety of data services
- NoSQL (and NewSQL) databases
  - Alternative to ACID databases
  - Tradeoff strong consistency for low latency



Cloud Services

Querying and transforming data

- Structured Query Language (SQL)
- Rule engines
- Complex event processing (CEP)
- Streaming SQL
  - *Continuous* queries
  - SQL-interface for event processing
- MapReduce programming
- Hosting and executing custom programs

ETSI's Service Capability Layer

On top of connectivity: holdall for functional, security, and governance aspects

- Communication
  - Semisynchronous: quick ack followed by full answer
  - Asynchronous
  - Works on top of another protocol, e.g., HTTP and CoAP
- Service capabilities
  - Registration
  - Access control
  - Authentication
  - Data transfer
  - Subscribe and notify
  - Handling groups

Fog Computing

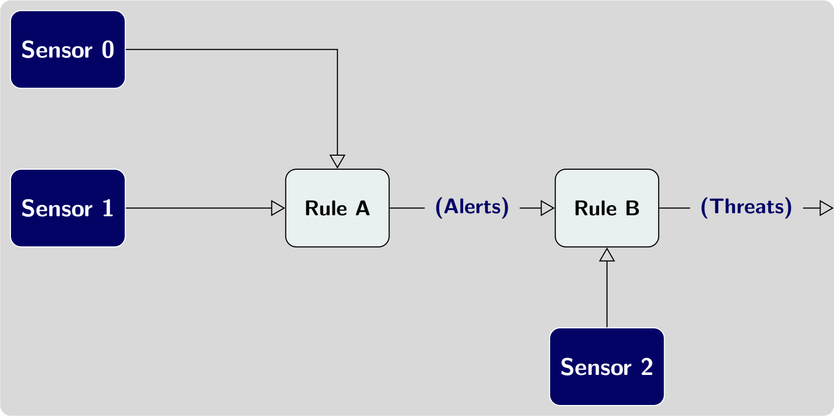
Cisco's term for decentralized or "ground-level" variant of cloud computing

- Intermediary between things and the cloud (data centers)
- Place computing, storage, services near the edge
- Process data closer to where it originates
  - Many devices talk to one another
  - Improve latency and throughput
- Improves treatment of autonomy with gains in
  - Local governance
  - Security

Event-Driven Architecture

Consume and produce event streams

- Event attributes: ID, timestamp, and content values



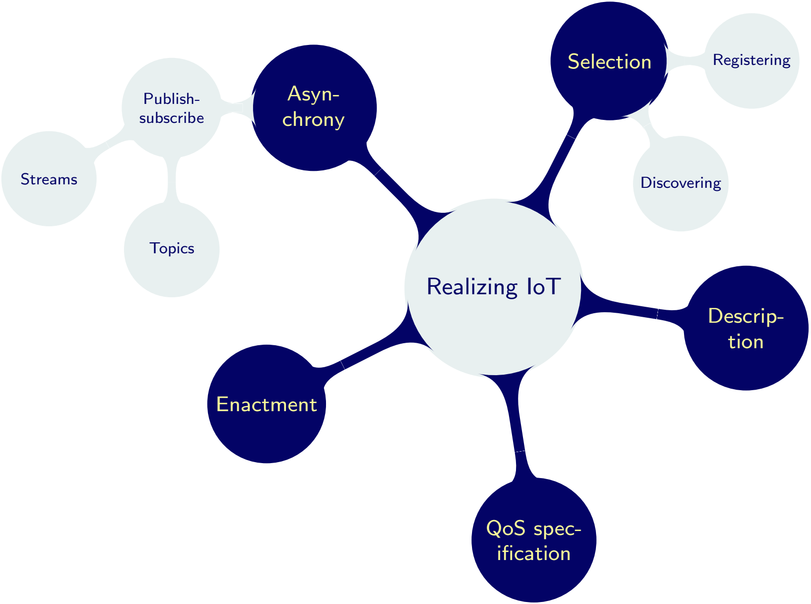
Linked Data Highlights

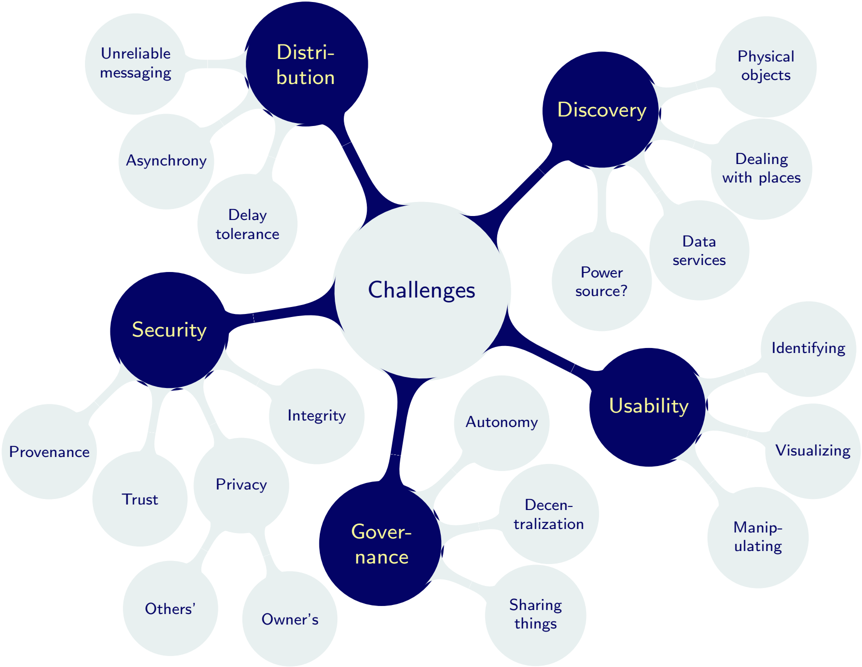
- Web-based identity and network
  - URIs identify resources
  - Deferencing a URI produces a resource description
  - Resources link to other resources
- Web-based semantics
  - Encoded in a Semantic Web language, such as the Resource Description Framework, RDF
- Exploit underlying architecture, e.g., Domain Name System and Web servers, to locate resources

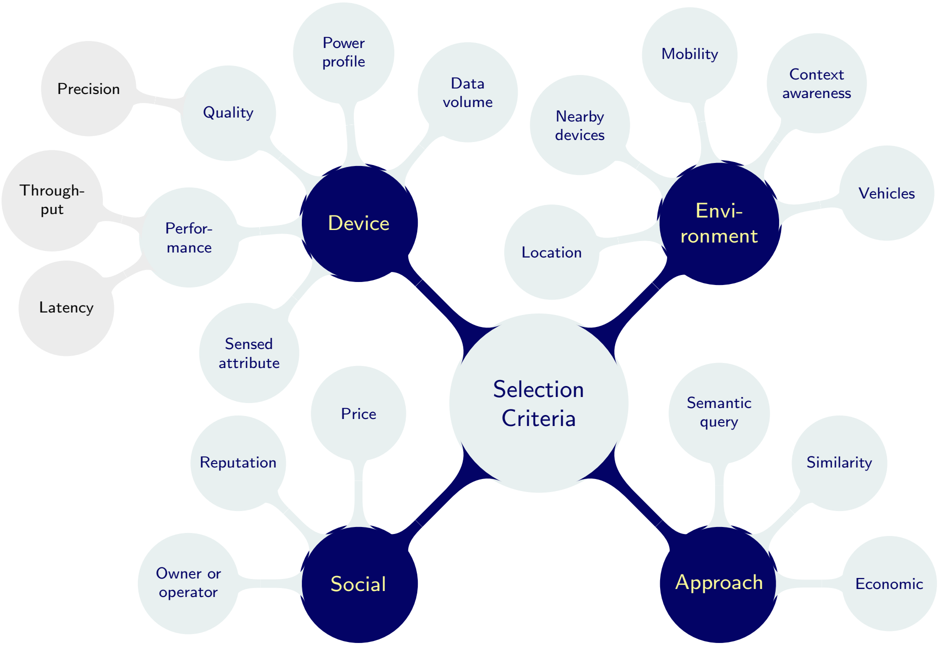
Linked Data for IoT

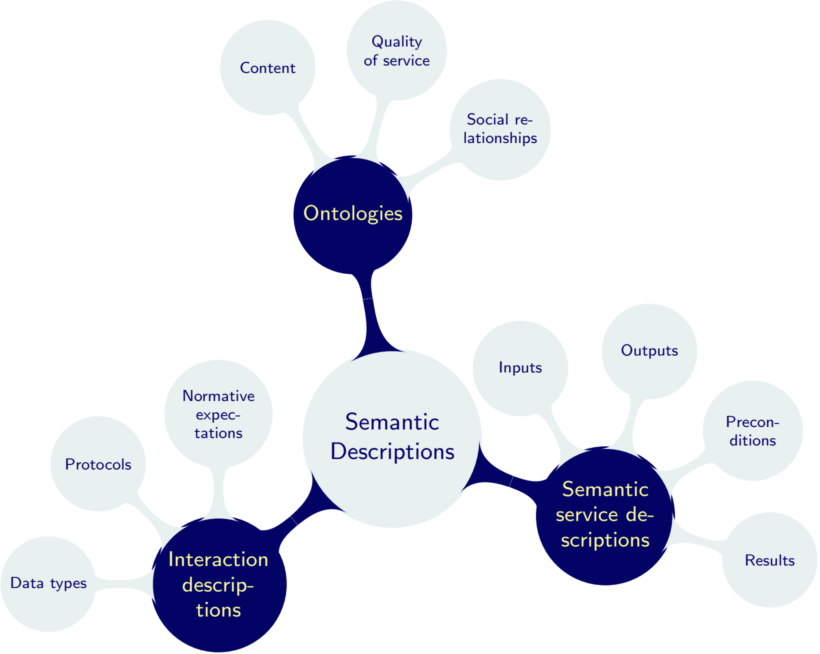
Sensor data is worthless unless linked to something else!

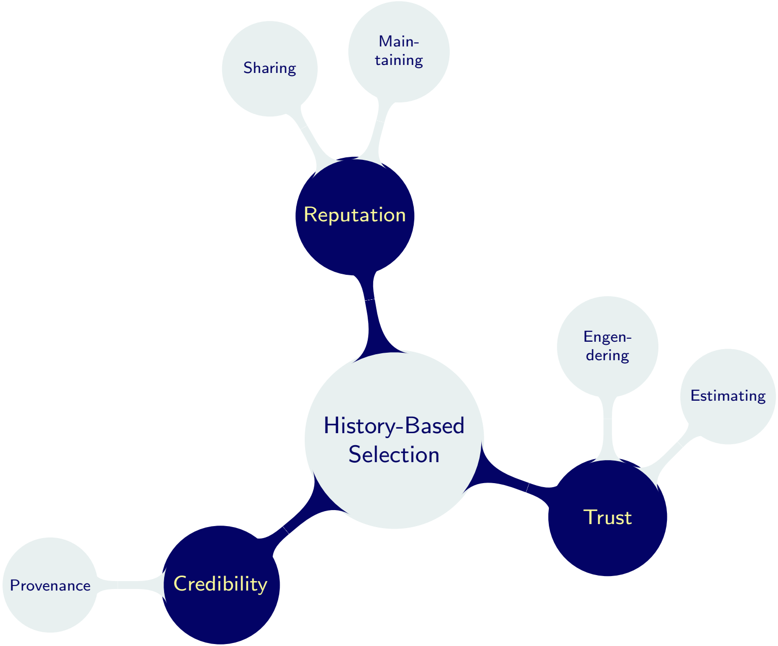
- Describe things
  - Link thing descriptions to one another
  - Enable thing discovery
- Describe the information produced by things
- Describe how to configure things
- Enable things to exchange semantic information
  - Facilitate reasoning
  - Promote interoperability
- Supported by standards that incorporate URIs
  - CoAP, ETSI's Service Capability Layer

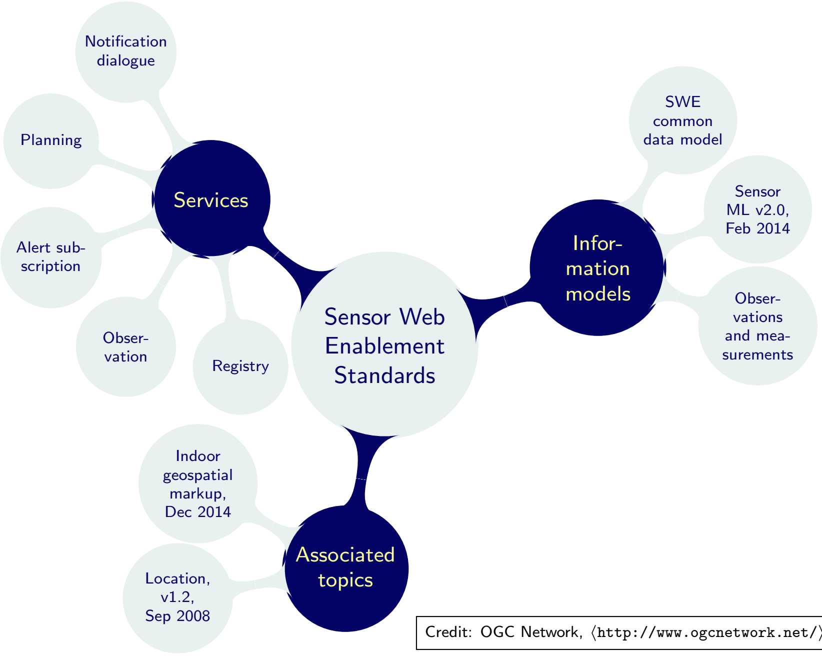

Realizing IoT
Asyn-chrony
Publish-subscribe
Streams
Topics
Selection
Registering
Discovering
Descrip-tion
QoS spec-ification
Enactment


Challenges
Distri-
bution
Unreliable messaging
Asynchrony
Delay tolerance
Discovery
Physical objects
Dealing with places
Data services
Power source?
Security
Integrity
Provenance
Trust
Privacy
Others'
Owner's
Gover-
nance
Autonomy
Decen-
tralization
Sharing things
Usability
Identifying
Visualizing
Manip-
ulating


Selection Criteria
Device
Quality
Precision
Power profile
Data volume
Perfor- mance
Through- put
Latency
Sensed attribute
Envi- ronment
Nearby devices
Mobility
Context awareness
Vehicles
Location
Social
Price
Reputation
Owner or operator
Approach
Semantic query
Similarity
Economic


Semantic
Descriptions
Ontologies
Content
Quality
of service
Social re-
lationships
Semantic
service de-
scriptions
Inputs
Outputs
Precon-
ditions
Results
Interaction
descrip-
tions
Normative
expec-
tations
Protocols
Data types


History-Based Selection
Reputation
Sharing
Main-taining
Trust
Engen-dering
Estimating
Credibility
Provenance



Credit: OGC Network, ⟨http://www.ogcnetwork.net/⟩

Sharing, Fusing, and Revising Information

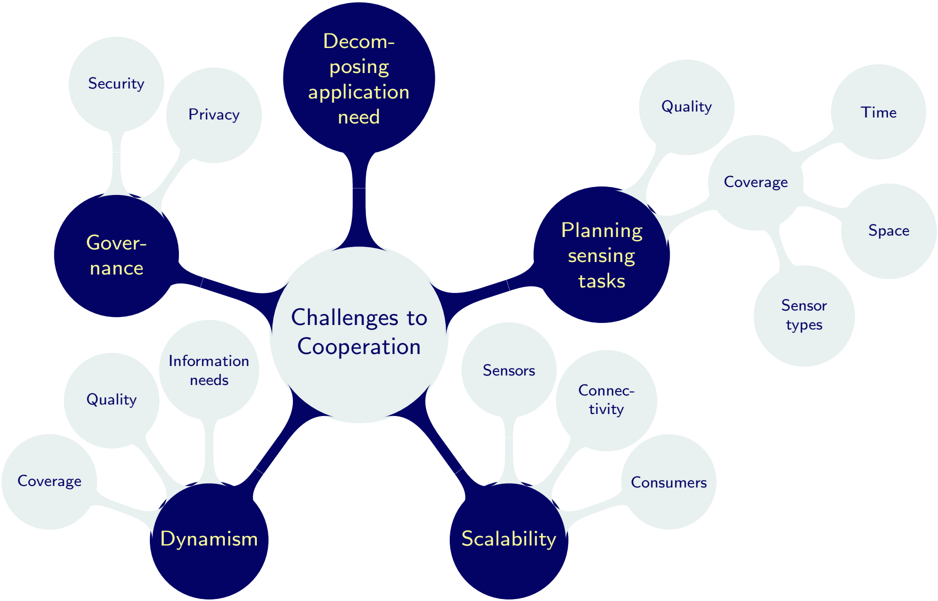
Inspired by social computing

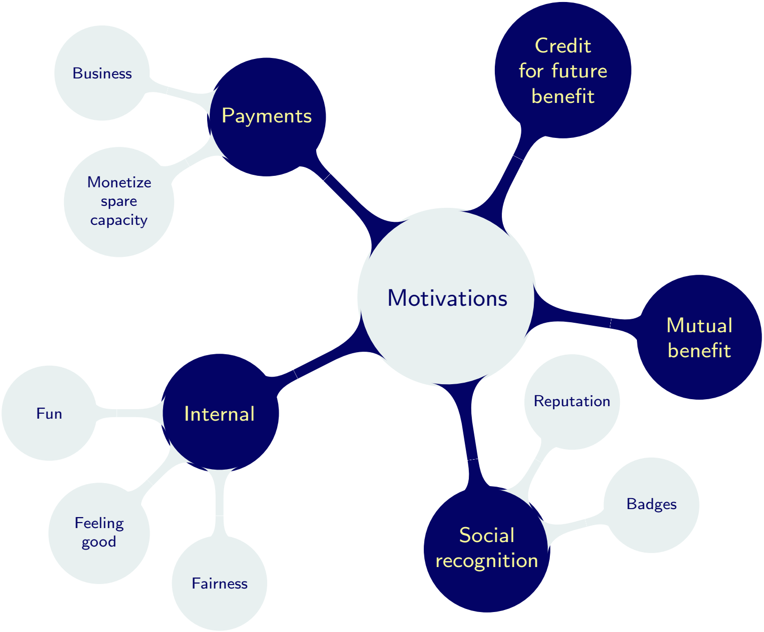
- Folk wisdom about aggregating knowledge
- Galton's Vox Populi
  - Median of many estimates represents the voice of the people
  - Underlying intuition of crowdsourcing
- Condorcet Jury Theorem
  - If each information source is better than 50% accurate (binary case)
  - Then their majority is even more accurate (depending on how many)
- Participatory or community sensing
  - Usually based on phone sensors rather than things as such
  - Includes explicit user actions, e.g., sending a picture

Assumptions in Participatory Sensing

Why would anyone participate honestly?

- Sensor owners bear a cost when the sensors provide data
  - Energy
  - Bandwidth
  - Installation and maintenance
- Sensor owners may
  - Refuse to participate
  - Configure their sensors to provide low quality results
  - Instruct their sensors to falsify results
- Majority or central estimates work when
  - The user (consumer) owns and controls all sensors
  - Incentives promote cooperation and verification
  - Other aspects induce *prosocial* behavior

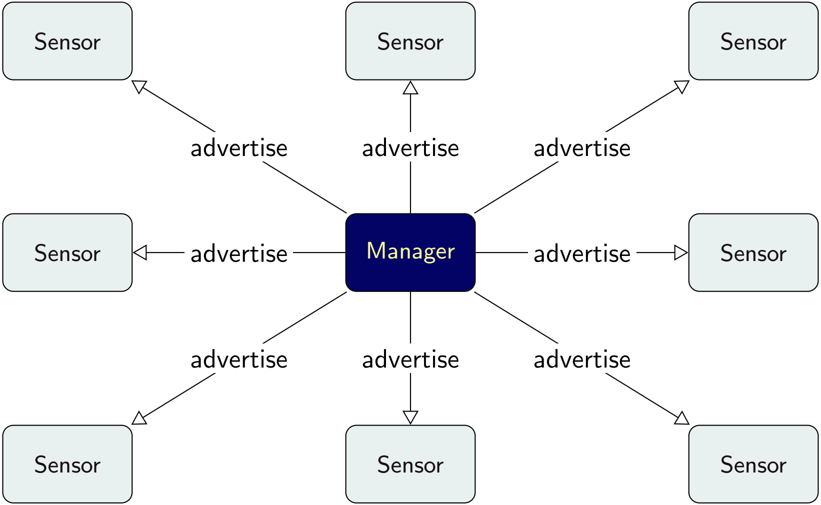

Challenges to Cooperation
Decom-posing application need
Gover-nance
Security
Privacy
Planning sensing tasks
Quality
Coverage
Time
Space
Sensor types
Dynamism
Information needs
Quality
Coverage
Scalability
Sensors
Connec-tivity
Consumers


Motivations
Payments
Business
Monetize spare capacity
Credit for future benefit
Mutual benefit
Internal
Fun
Feeling good
Fairness
Social recognition
Reputation
Badges


Incentive Mechanisms
Fixed reward
May lead to cheating
Spot checking
May not be viable
Mutual checking
May induce subop-timality
Assumes unique perspective
Dynamic pricing
Balance demand and supply
Assumes quality estimation
Tension
Avoid exploitation
Project fairness

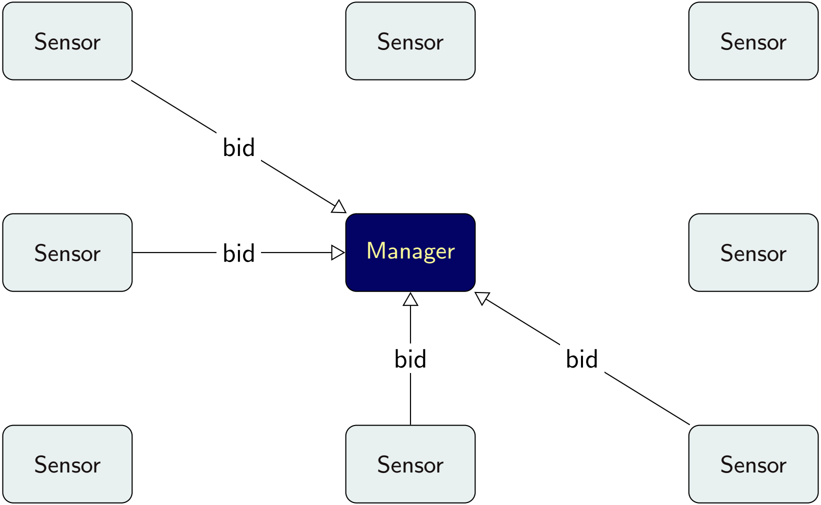
Contract Net Protocol: 1

Announce



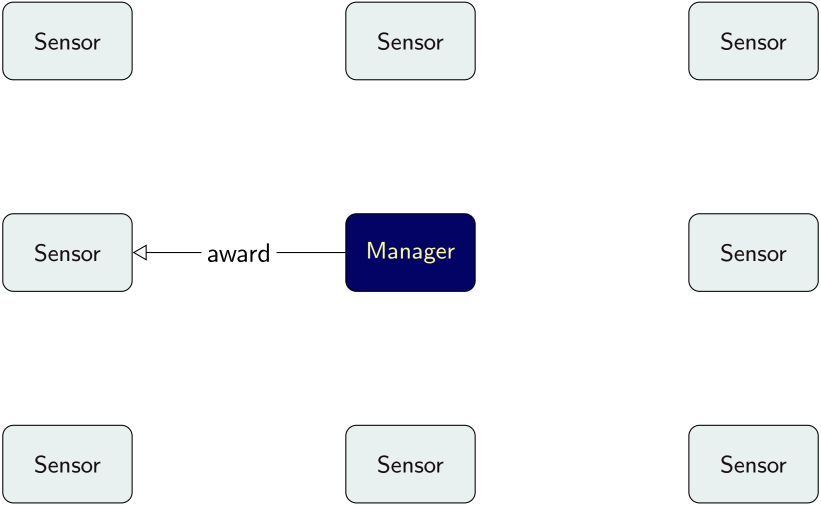
Contract Net Protocol: 2

Bid



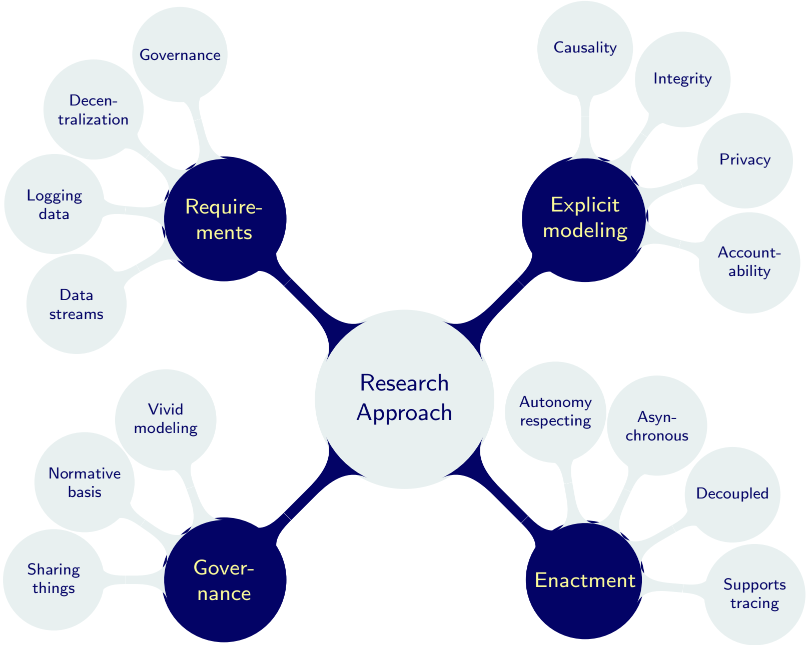
Contract Net Protocol: 3

Award



Contract Net Protocol Evaluated

- Generic protocol
  - Can apply recursively
- Supports important properties
  - Robustness against sensor failure
  - Robustness against connectivity failure
  - Incorporating economic features
- Not well specified
  - Confuses interaction and internal reasoning
  - Limited to two-party interactions

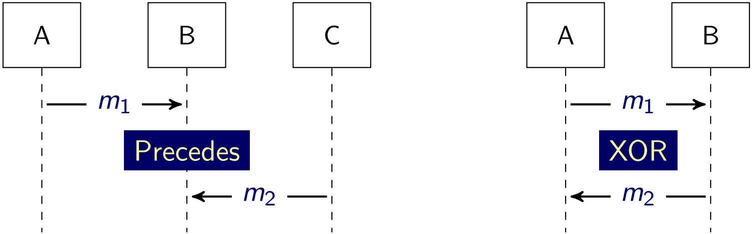

Research Approach
Require- ments
Governance
Decen- tralization
Logging data
Data streams
Explicit modeling
Causality
Integrity
Privacy
Account- ability
Gover- nance
Vivid modeling
Normative basis
Sharing things
Enactment
Autonomy respecting
Asyn- chronous
Decoupled
Supports tracing

Motivating Declarative Interaction Orientation

- Challenges
  - Operational challenges
    - Asynchronous communication
    - Unordered channels
    - Unreliable channels
  - Social challenges
    - Autonomy and heterogeneity of participants
    - Need for explicit information semantics, norms, and policies
- Current techniques
  - Address operational challenges procedurally
    - Client-server with limited support for asynchrony
    - Two-party interactions
    - Multiparty approaches: over-constrained
  - Address social challenges declaratively
    - Impedance mismatch with operational models
    - Not conducive to data representation
- Information-based approach for interactions
  - Operationalization at low level
  - Socially relevant representation at high level

Traditional Specifications: Procedural

Low-level, over-specified protocols, easily wrong



- Traditional approaches
  - Emphasize arbitrary ordering and occurrence constraints
  - Then work hard to deal with those constraints
- Our philosophy: The Zen of Distributed Computing
  - Necessary ordering constraints fall out from *causality*
  - Necessary occurrence constraints fall out from *integrity*
  - Unnecessary constraints: simply *ignore* such

Properties of Participants

- Autonomy
- Myopia
  - All choices must be local
  - Correctness must not rely on future interactions
- Heterogeneity: local $\neq$ internal
  - Local state (projection of global state, which is stored nowhere)
    - Public or observable
    - Typically, must be revealed for correctness
  - Internal state
    - Private
    - Must never be revealed: to avoid false coupling
- Shared nothing representation of local state
  - Enact via messaging

BSPL, the Blindingly Simple Protocol Language

Main ideas

- Only *two* syntactic notions
  - Declare a message schema: as an atomic protocol
  - Declare a composite protocol: as a bag of references to protocols
- Parameters are central
  - Provide a basis for expressing meaning in terms of bindings in protocol instances
  - Yield unambiguous specification of compositions through public parameters
  - Capture progression of a role's knowledge
  - Capture the completeness of a protocol enactment
  - Capture uniqueness of enactments through keys
- Separate structure (parameters) from meaning (bindings)
  - Capture many important constraints purely structurally

Key Parameters in BSPL

Marked as ⌜key⌝

- All the key parameters *together* form the key
- Each protocol must define at least one key parameter
- Each message or protocol reference must have at least one key parameter in common with the protocol in whose declaration it occurs
- The key of a protocol provides a basis for the uniqueness of its enactments

Parameter Adornments in BSPL

Capture the essential causal structure of a protocol (for simplicity, assume all parameters are strings)

- ⌜in⌝: Information that must be provided to instantiate a protocol
  - Bindings must exist locally in order to proceed
  - Bindings must be produced through some other protocol
- ⌜out⌝: Information that is generated by the protocol instances
  - Bindings can be fed into other protocols through their ⌜in⌝ parameters, thereby accomplishing composition
  - A standalone protocol must adorn all its public parameters ⌜out⌝
- ⌜nil⌝: Information that is absent from the protocol instance
  - Bindings must not exist

The *Hello* Protocol

```
Hello {
 role Self, Other
 parameter out greeting key

 Self ↦ Other: hi[out greeting key]
}
```

- At most one instance of *Hello* for each greeting
- At most one *hi* message for each greeting
- Enactable standalone: no parameter is ⌜in⌝
- The key of *hi* is explicit; often left implicit on messages

The *Data Transfer* Protocol

```
Data Transfer {
 role Sender, Recipient
 parameter in ID key, in data

 Sender ↦ Recipient: dataM[in ID, in data]
}
```

- At most one *dataM* for each ID
- Not enactable standalone: **why?**
- The key of *dataM* is implicit (for brevity)

The *Request Response* Protocol

```
Request Response {
 role Requester, Responder
 parameter in ID key, out query, out answer

 Requester ↦ Responder: request[in ID, out query]
 Responder ↦ Requester: response[in ID, out query, out answer]
}
```

- The key ID uniquifies instances of *Request Response*, *request*, and *response*
- Not enactable standalone: at least one parameter is ⌜in⌝
- An instance of *request* must precede any instance of *response* with the same ID: **why?**
- No message need occur: **why?**
- *response* must occur for *Request Response* to complete: **why?**

The *Subscription Setup* Protocol

```
Subscription Setup {
 role Publisher, Subscriber
 parameter out ID key, out metadata, out rID

 Subscriber ↦ Publisher: request[out ID, out metadata]

 Publisher ↦ Subscriber: accept[in ID, in metadata, out rID]
 Publisher ↦ Subscriber: reject[in ID, in metadata, out rID]
}
```

- The key ID uniquifies instances of *Subscription Setup* and each of its messages
- Enactable standalone
- A *request* must precede an *accept* with the same ID
- A *request* must precede a *reject* with the same ID
- An *accept* and a *reject* with the same ID *cannot* both occur: **why?**

Sensor Subscription

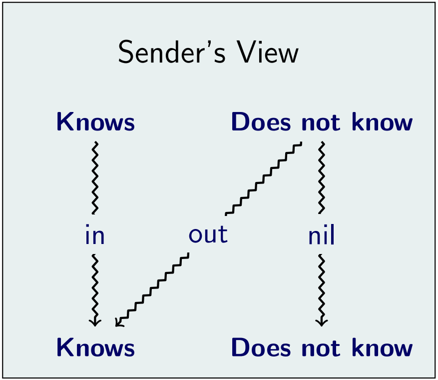
Potentially to be composed with *Subscription Setup*

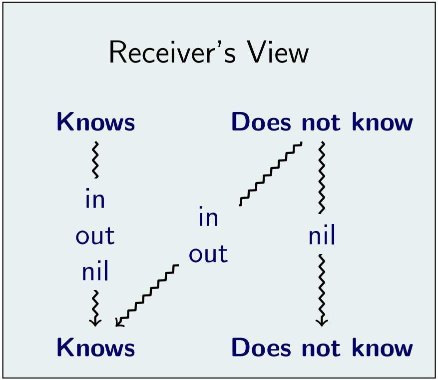
```
Sensor Subscription {
 role Sensor, Subscriber
 parameter in contractNO key, out request key, out dataResponse key
 private end

  Subscriber ↦ Sensor: start[in contractNO, out query]
  Subscriber ↦ Sensor: stop[in contractNO, in query, out end]

  Sensor ↦ Subscriber: dataResponse[in contractNO, nil end, in
      query, out item]
}
```

- Notice the composite key
- One contract maps to many queries
- One query maps to many data responses

Knowledge and Viability

When is a message viable? What effect does it have on a role's local knowledge?





- Knowledge increases monotonically at each role
- An ⌜out⌝ parameter **creates** and transmits knowledge
- An ⌜in⌝ parameter transmits knowledge
- Repetitions through multiple paths are harmless and superfluous

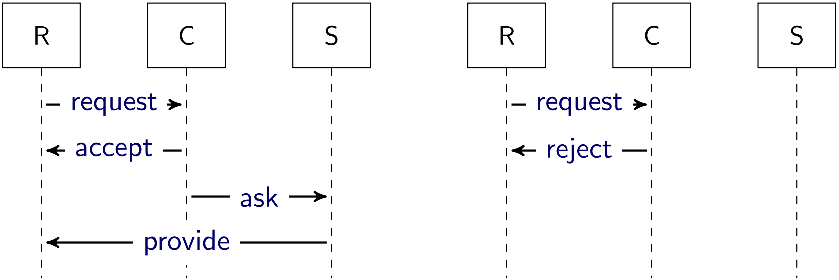
The *Approved Access* Protocol

```
Approved Access {
 role Requester, Controller, Source
 parameter out ID key, out item, out outcome
 private token, responded, value

  Requester ↦ Controller: request[out ID, out item]
  Controller ↦ Requester: accept[in ID, in item, out token, out
      responded]
  Controller ↦ Requester: reject[in ID, in item, out outcome,
      out responded]
  Controller ↦ Source: ask[in ID, in item, in token]
  Source ↦ Requester: provide[in ID, in item, in token, out
      value, out outcome]
}
```

- Demonstrates delegation instead of request-response patterns
- Enactable standalone: no parameters are ⌜in⌝
- *reject* conflicts with *accept* on responded (a *private* parameter)
- *reject* or *provide* must occur for completion (to bind outcome)

Possible Enactments as Sets of Local Histories

Each participant's local history: sequence of messages sent and received



Safety: *Approved Access Unsafe*

Removed local conflict between *accept* and *reject*

```
Approved Access Unsafe {
 role Requester, Controller, Source
 parameter out ID key, out item, out outcome
 private token, value //REMOVED: responded

  Requester ↦ Controller: request[out ID, out item]
  Controller ↦ Requester: accept[in ID, in
    item, out token] //, out responded
  Controller ↦ Requester: reject[in ID, in item, out outcome]
     //, out responded
  Controller ↦ Source: ask[in ID, in item, in token]
  Source ↦ Requester: provide[in ID, in item, in token, out
     value, out outcome]
}
```

- CONTROLLER can send both *accept* and *reject*
- Thus outcome can be bound twice in the same enactment

Liveness: *Approved Access Minus Ask*

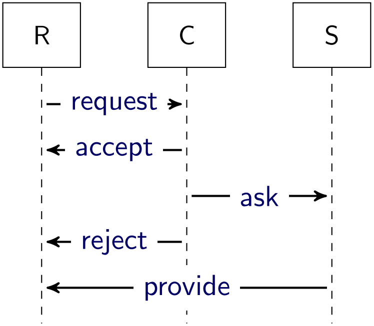
Omitted the *ask* message

```
Approved Access Minus Ask {
 role Requester, Controller, Source
 parameter out ID key, out item, out outcome
 private token, responded, value

  Requester ↦ Controller: request[out ID, out item]
  Controller ↦ Requester: accept[in ID, in item, out token, out
      responded]
  Controller ↦ Requester: reject[in ID, in item, out outcome,
     out responded]
  // Controller ↦ Source: ask[in ID, in item, in token]
  Source ↦ Requester: provide[in ID, in item, in token, out
      value, out outcome]
}
```

- If CONTROLLER sends *reject*, the enactment completes
- If CONTROLLER sends *accept*, the enactment deadlocks

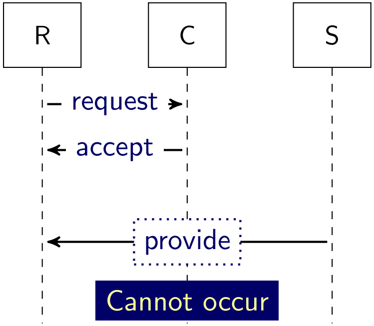
Safety and Liveness Violations

Approved Access Unsafe



Safety Violation

Approved Access Minus Ask



Liveness Violation

Encode Causal Structure as Temporal Constraints

- *Reception*. If a message is received, it was previously sent.
- *Information transmission* (sender's view)
  - Any ⌜in⌝ parameter occurs prior to the message
  - Any ⌜out⌝ parameter occurs simultaneously with the message
- *Information reception* (receiver's view)
  - Any ⌜out⌝ or ⌜in⌝ parameter occurs before or simultaneously with the message
- *Information minimality*. If a role observes a parameter, it must be simultaneously with *some* message sent or received
- *Ordering*. If a role sends any two messages, it observes them in some order

Verifying Safety

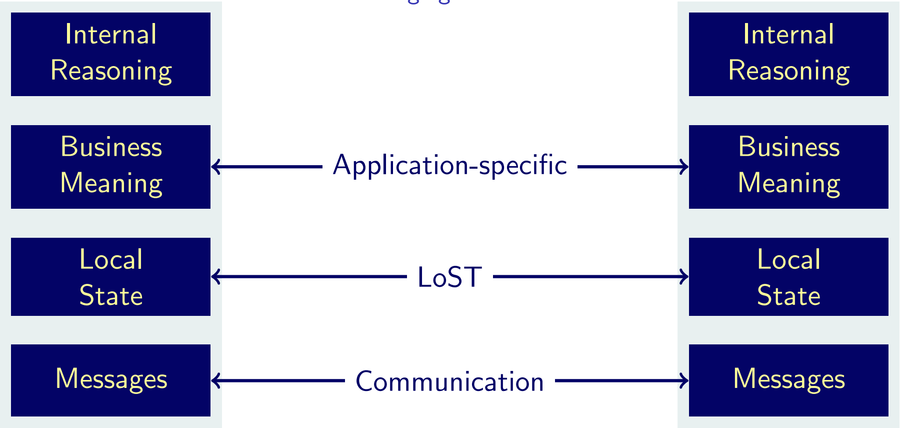
- Competing messages: those that have the same parameter as out
- *Conflict*. At least two competing messages occur
- *Safety* iff the causal structure $\wedge$ conflict is unsatisfiable

Verifying Liveness

- *Maximality*. If a role is enabled to send a message, it sends at least one such message
- *Reliability*. Any message that is sent is received
- *Incompleteness*. Some public parameter fails to be bound
- *Live* iff the causal structure $\wedge$ the occurrence is unsatisfiable

Realizing BSPL via LoST (Local State Transfer)

Does *not* assume FIFO or reliable messaging



- Unique messages
- Integrity checks on incoming messages
- Consistency of local choices on outgoing messages

Implementing LoST

Think of the message logs you want

- For each role
  - For each message that it sends or receives
    - Maintain a *local* relation of the same schema as the message
- Receive and store any message provided
  - It is not a duplicate
  - Its integrity checks with respect to parameter bindings
  - Garbage collect expired sessions: requires additional annotations
- Send any unique message provided
  - Parameter bindings agree with previous bindings for the same keys for ⌜in⌝ parameters
  - No bindings for ⌜out⌝ and ⌜nil⌝ parameters exist

Comparing LoST and ReST

	ReST	*LoST*
Modality	Two-party; client-server; synchronous	Multiparty interactions; peer-to-peer; asynchronous
Computation	Server computes definitive resource state	Each party computes its definitive local state and the parties collaboratively and (potentially implicitly) compute the definitive interaction state
State	Server maintains no client state	Each party maintains its local state and, implicitly, the relevant components of the states of other parties from which there is a chain of messages to this party

Comparing LoST and ReST

	ReST	*LoST*
Transfer	State of a resource, suitably represented	Local state of an interaction via parameter bindings, suitably represented
Idempotent	For some verbs, especially GET	Always; repetitions are guaranteed harmless
Caching	Programmer can specify if cacheable	Always cacheable
Uniform interface	GET, POST, ...	⌜in⌝, ⌜out⌝, ⌜nil⌝
Naming	Of resources via URIs	Of interactions via (composite) keys, whose bindings could be URIs

Remark on Control versus Information Flow

- Control flow
  - Natural within a single computational thread
  - Exemplified by conditional branching
  - Presumes master-slave relationship across threads
  - Impossible between mutually autonomous parties because neither controls the other
  - May sound appropriate, but only because of long habit
- Information flow
  - Natural across computational threads
  - Explicitly tied to causality

Bliss Conceptual Model: Functions of Parameters

- Key
  - For interaction instantiation and uniqueness
- Payload
  - For interaction meaning
- Completion
  - To help determine when the interaction is over
- Integrity
  - For interaction integrity
- Control
  - To force certain preferred orders of enactment

BSPL Highlights

Taking a declarative, information-centric view of interaction to the limit

- Specification
  - Protocol: name and a set of parameters (including keys)
  - Each protocol has *inputs* and *outputs*
  - A message is an atomic protocol
  - A composite protocol is a set of references to protocols
- Representation
  - A protocol corresponds to a relation (table)
  - Integrity constraints apply on the relations
- Enactment via LoST: Local State Transfer
  - Information represented: local $\neq$ internal
  - Purely decentralized at each role
  - Materialize the relations *only* for messages

Information Centrism

Characterize each interaction purely in terms of information

- Explicit causality
  - Flow of information coincides with flow of causality
  - No hidden control flows
  - No backchannel for coordination
- Keys
  - Uniqueness
  - Basis for completion
- Integrity
  - Must have bindings for some parameters
  - Analogous to NOT NULL constraints
- Immutability
  - Durability
  - Robustness: insensitivity to
    - Reordering by infrastructure
    - Retransmission: one delivery is all it needs

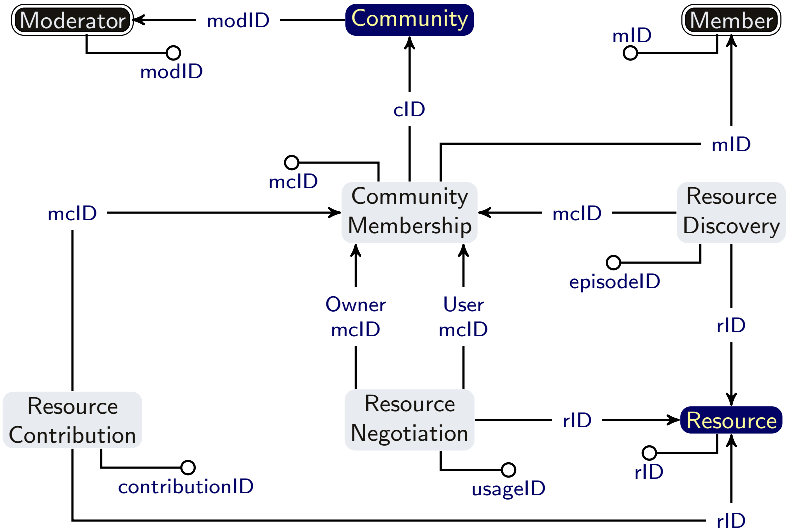
Bliss Methodology

Iterate over the following steps

1. Identify the roles needed in a protocol
2. Identify the conceptual social object computed
3. Identify the messages (or, recursively, subprotocols) to compute the social object
4. Identify each message as a component of the social object and any additional constraints
5. Introduce polymorphism of messages to support flexible sourcing of parameter bindings

Schema for IoT Resource Sharing

Maps to four protocols, naturally composed



The *Community Membership* Protocol

```
Community Membership {
 role Mod, Mem // Moderator, Member
 parameter in cID key, in mID key, opt mcID, out outcome
 private request

 Mem ↦ Mod: requestAdmission[in cID, in mID, out request]
 Mod ↦ Mem: admit[in cID, in mID, in request, out mcID, out
     outcome]
 Mod ↦ Mem: deny[in cID, in mID, in request, nil mcID, out
     outcome]
}
```

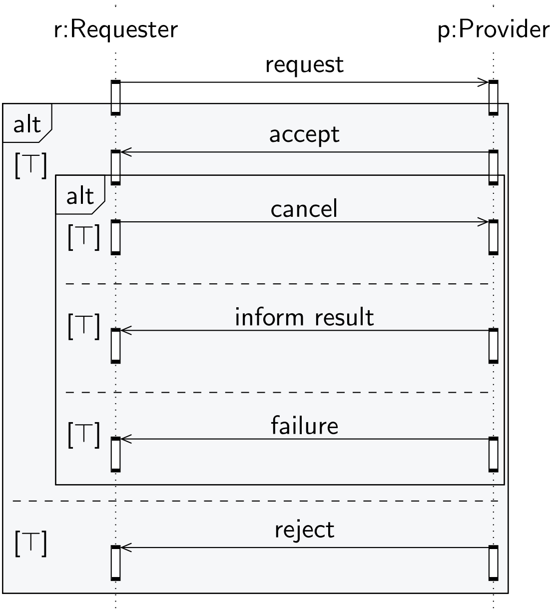
The *Resource Contribution* Protocol

```
Resource Contribution {
 role Mod, Mem // Moderator, Member
 parameter in mcID key, in rID key, out contributionID key, out
     outcome
 private rlocation, rType

 Mem ↦ Mod: contribute[in mcID, in rID, out rLocation, out
     rType, out contributionID]
}
```

The *Resource Discovery* Protocol

```
Resource Discovery {
 role Mod, Mem // Moderator, Member
 parameter out episodeID key, out rID key, out rOwnerID
 private rlocation, rType

 Mem ↦ Mod: search[out episodeID, out rLocation, out rType]
 Mod ↦ Mem: response[in episodeID, in rLocation, in rType, out
     rOwnerID, out rID]
}
```

Service Request
Protocol
(Erroneous: Unsafe)

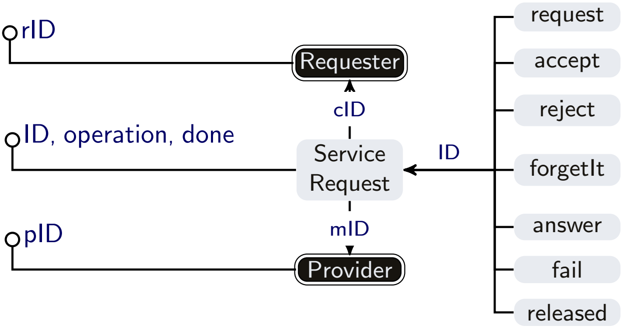


BSPL Reconstruction of Unsafe *Service Request*

Combining some parameters to reduce clutter

```
protocol OOI Service Request Unsafe {
 role R, P
 parameter out ID key, out operation, out result
 private confirmation

 R ↦ P: request[out ID, out operation]
 P ↦ R: accept[in ID, out confirmation]
 P ↦ R: reject[in ID, out confirmation, out result]
 R ↦ P: cancel[in ID, out result]
 P ↦ R: fail[in ID, out result]
 P ↦ R: answer[in ID, out result]
}
```

A Conceptual Schema for *Service Request*



The *Service Request* Protocol Via Bliss, Now Corrected

```
protocol OOI Service Request Corrected {
 role R, P
 parameter out ID key, out operation, out result
 private confirmation, releaseToken

 R ↦ P: request[out ID, out operation]
 P ↦ R: accept[in ID, in operation, out confirmation]
 P ↦ R: reject[in ID, in operation, out confirmation, out
     result]
 R ↦ P: forgetIt[in ID, in operation, in confirmation, out
     releaseToken]
 P ↦ R: answer[in ID, in operation, in confirmation, nil
     releaseToken, out result]
 P ↦ R: fail[in ID, in operation, in confirmation, nil
     releaseToken, out result]
 P ↦ R: released[in ID, in operation, in releaseToken, out
     result]
}
```

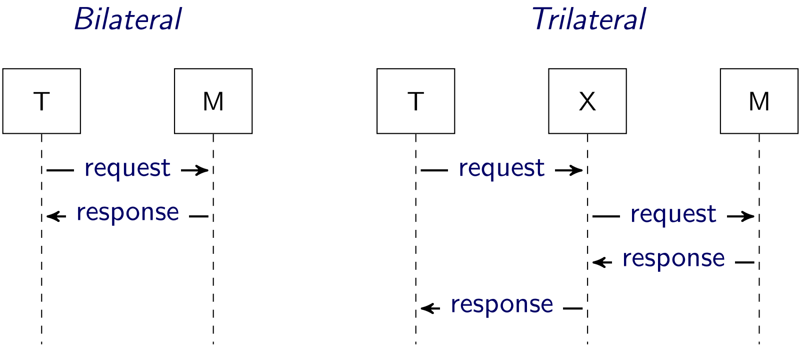
in-out Polymorphism

price could be ⌜in⌝ or ⌜out⌝

```
Flexible−Offer {
 role B, S
 parameter in ID key, out item, price, out qID

 B ↦ S: rfq[ID, out item, nil price]
 B ↦ S: rfq[ID, out item, in price]

 S ↦ B: quote[ID, in item, out price, out qID]
 S ↦ B: quote[ID, in item, in price, out qID]
}
```

- The price can be adorned ⌜in⌝ or ⌜out⌝ in a reference to this protocol

Composing Protocols



- How can we compose *Trilateral* from two copies of *Bilateral*?
  - As a protocol: without imposing private constraints on a party

The *Bilateral Discovery* Protocol

Inspired from foreign currency transaction standards

```
Bilateral {
 role Taker, Maker
 parameter out ID key, out query, out result

  Taker ↦ Maker: request[out ID, out query]
  Maker ↦ Taker: response[in ID, in query, out result]
}
```

- Need another variant to realize the desired composition

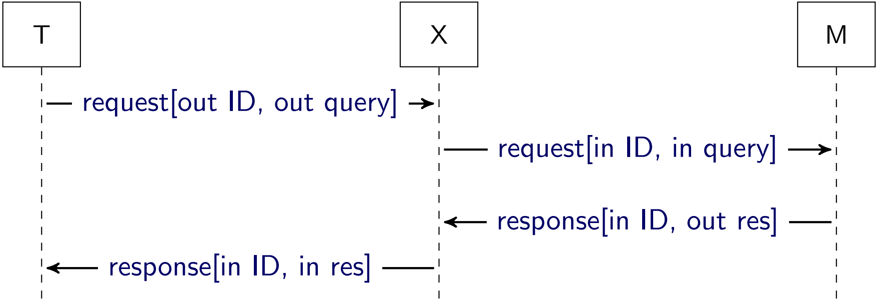
The *General Bilateral Discovery* Protocol

Combining the variants into one via polymorphism

```
GeneralBilateral {
 role Taker, Maker
 parameter ID key, query, res

  Taker ↦ Maker: request[out ID, out query]
  Taker ↦ Maker: request[in ID, in query]

  Maker ↦ Taker: response[in ID, in query, out res]
  Maker ↦ Taker: response[in ID, in query, in res]
}
```

The *Trilateral Discovery* Protocol

```
Trilateral {
 role Taker, Exchange, Maker
 parameter out ID key, out query, out res

 GeneralBilateral(Taker, Exchange, out ID, out query, in res)
 GeneralBilateral(Exchange, Maker, in ID, in query, out res)
}
```

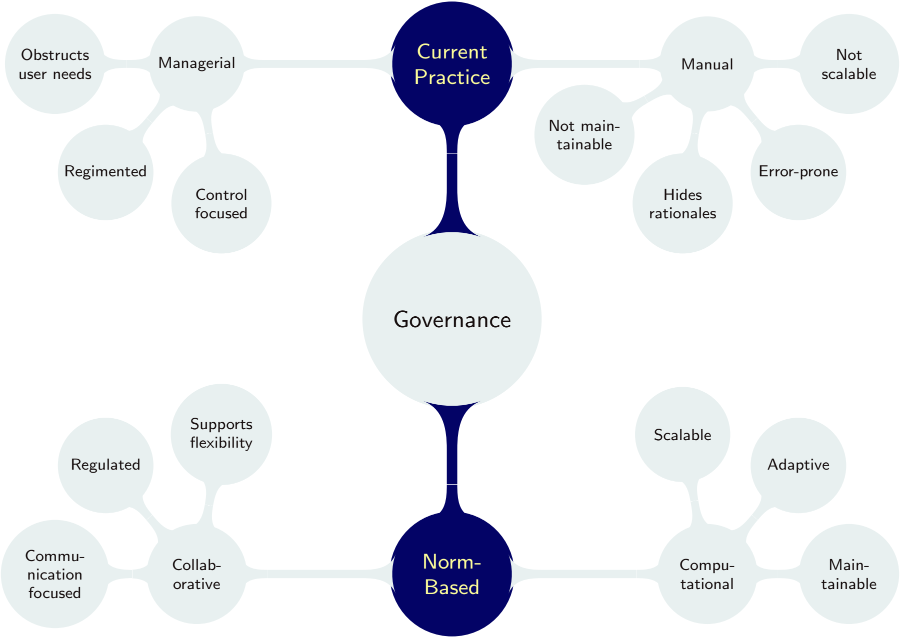


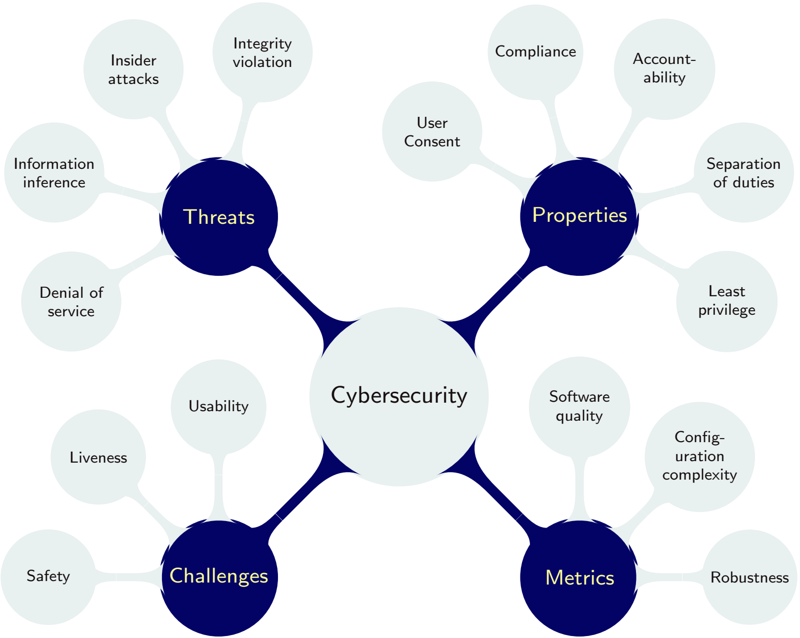
The *Contract Net* Protocol

Demonstrating enhancements: uni and $\sqsupseteq$

```
Contract Net {
 role Manager uni, Bidder ⊒ Winner uni
 parameter out ID key, out request, out response, out decision

 Manager ↦ Bidder: CfB[out ID, out request]

 Bidder ↦ Manager: bid[in ID, in request, out response]

 Manager ↦ Winner: award[in ID, in request, in response, out
     decision]
}
```

- Could also write this without the enhancements


Governance
Current Practice
Managerial
Obstructs user needs
Regimented
Control focused
Manual
Not scalable
Not maintainable
Hides rationales
Error-prone
Norm-Based
Collaborative
Communication focused
Regulated
Supports flexibility
Computational
Scalable
Adaptive
Maintainable


Cybersecurity
Threats
Insider attacks
Integrity violation
Information inference
Denial of service
Properties
User Consent
Compliance
Account-ability
Separation of duties
Least privilege
Challenges
Usability
Liveness
Safety
Metrics
Software quality
Config-uration complexity
Robustness

Governance for Secure Collaboration

Broadly, administering sociotechnical systems to serve stakeholder needs

Existing and emerging standards, such as ETSI's, provide partial support for governance

- Resource management
  - Devices
  - Channels
- Group management
  - Membership creation, verification, maintenance...
  - Subscriptions, announcements
  - Permissions
- Credential management
  - Authentication
  - Authorization

Operational specifications but no formal representation or model

Credit: ETSI draft TS 102 690, May 2014, ⟨http://docbox.etsi.org/smartM2M/Open/Latest_Drafts/⟩

Governance: Critique

- Currently, automated support comes with managerial imposition: by superiors on subordinates
  - Control over managed resources
  - Necessary but not sufficient
  - Unsuited to many settings
    - When user needs aren't met, they subvert managerial diktats
    - Resulting in vulnerabilities
- Currently, governance is manual via out-of-band communications
  - Low productivity
  - Poor scalability to fine-grained, real-time governance decisions
  - Hidden, implicit considerations yield low confidence in correctness and poor maintainability
    - Lead to errors
    - Therefore, vulnerabilities

Governance Challenges in IoT

Accommodating autonomy, heterogeneity, and dynamism

- Support *configurational adaptation*
  - Resource sharing: Offer ocean instrument for sharing
  - Affiliation: Add new laboratories
  - Sanction: Allow external sharing of results to fulfill deliverables
- Support *operational adaptation*
  - Resource sharing: Preempt low-priority users in case of oil spill
  - Affiliation: Forbid unilateral publishing of results
  - Sanction: Absolve researcher who reveals results to prevent public endangerment (extenuating circumstances)
- Research challenges
  - Abstractions to capture rules of encounter
  - Methods to design and analyze such abstractions
  - Methods to implement such abstractions

Foundations of Secure Collaboration over IoT

Social perspective that complements technical (data, application, infrastructure) perspectives

- *Policy:* An implementation-independent specification of decision making
- *Normative basis:* Key relationships are reflected in norms, to be used a standard of correctness for interactions
- *Social context:* An Org (as a microsociety) recursively provides the context for the norms among and policies of its members
- *Interaction orientation:* How agents apply policies to enter into, monitor, and enact normative relationships

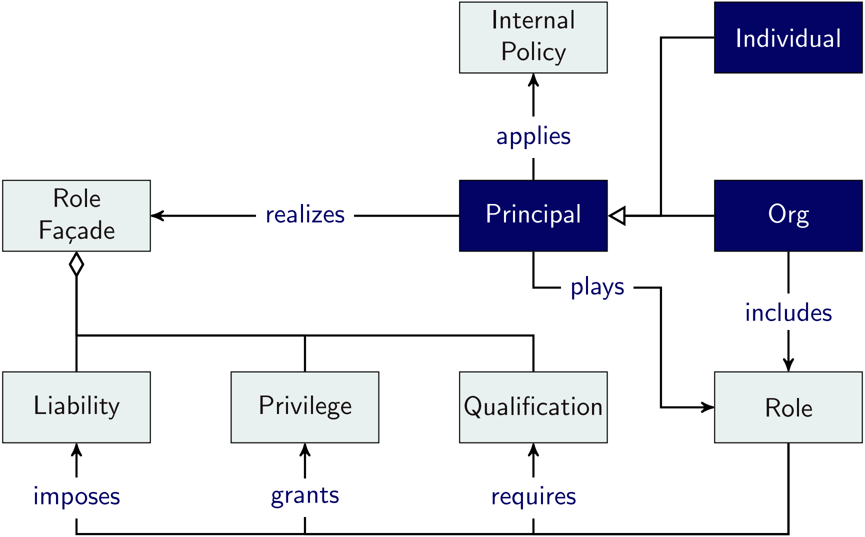
Principles of Governance: What Policies Give Us

Administration that is intelligent and intelligible

- Vividness of modeling
  - Grounded in applications; modeled entities are real
- Minimality of operational specifications
  - Leaving restrictions unstated except where essential to correctness
- Reification of representations
  - Explicit: hence, inspectable, sharable, and manipulable

Overview of Policy-Governed Secure Collaboration

Conceptual Model



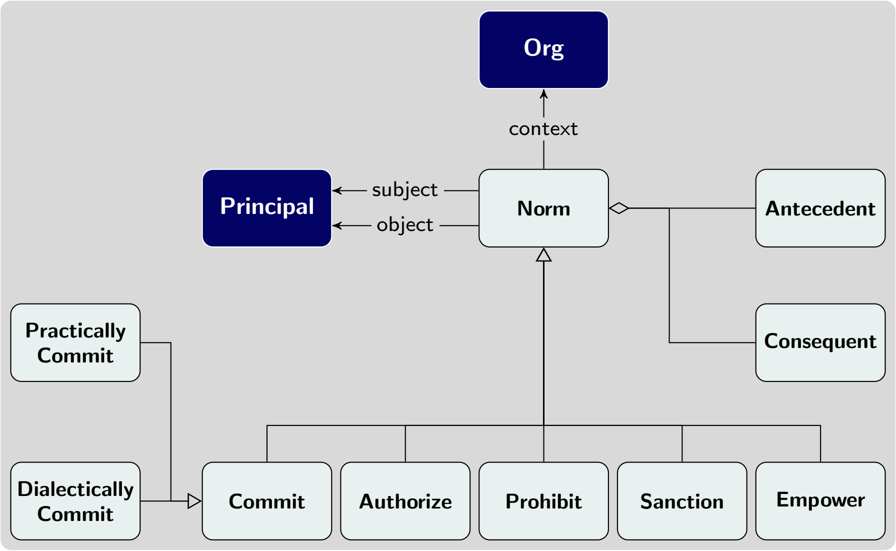
Achieving Governance: Principals and Orgs

Put collaboration in organizations center stage

- Principals are the stakeholders: people and organizations
  - Provide a locus for interaction
- Orgs are like *institutions:* have an identity and life time distinct from their members; also principals
  - Examples: NCSU, DoD, OOI, ...
  - Provide a locus for roles
  - Characterized via norms
  - Potentially enforce norms on members playing specific roles
    - An Org's main hold over its members is the threat of expulsion

Types of Directed Normative Relationships or Norms

Declarative; composable; manipulable

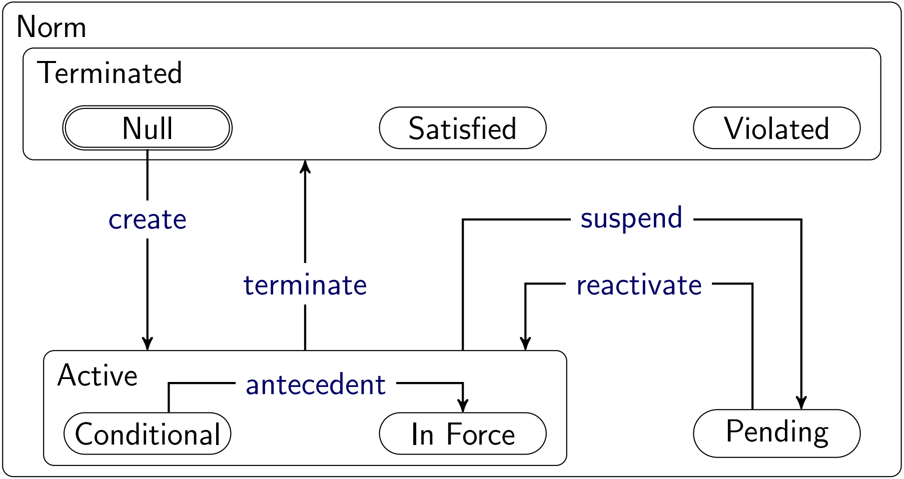


Norms as Façades

Norm	**Subject's Façade**	**Object's Façade**
Commitment	Liability	Privilege
Authorization	Privilege	Liability
Power	Privilege	Liability
Prohibition	Liability	Privilege
Sanction	Liability	Privilege

Life Cycle for Norms: 1

Using a variant of the UML state diagram notation



Life Cycle for Norms: 2

Substate of a terminated norm

If terminated in		**Then**				
ant	**con**	**Com**	**Aut**	**Pro**	**San**	**Pow**
false	false	null	null	null	null	null
false	true	sat	vio	null	null	null
true	false	vio	null	sat	null	vio
true	true	sat	sat	vio	sat	sat

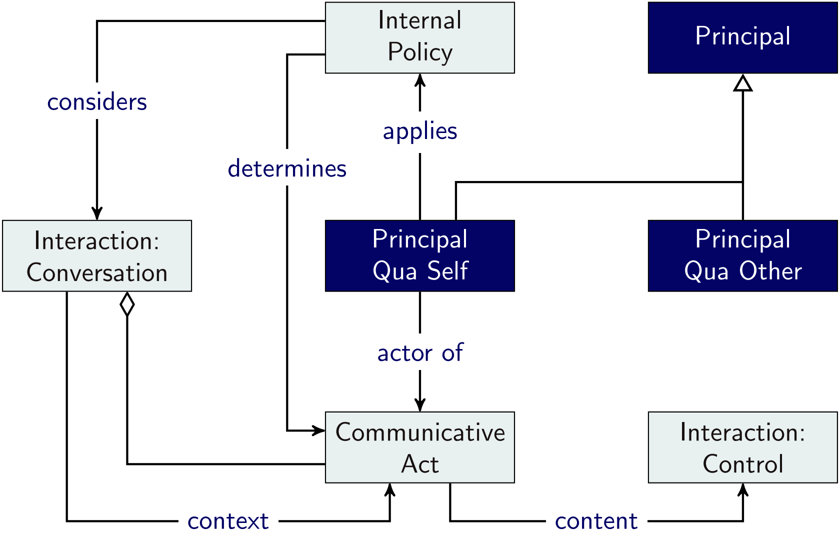
Unifying Norms and Policies for Governance

Promoting precision, verifiability, modularity, and reusability for secure collaboration

- Norms characterize interactions in terms of expectations and accountability
  - Provide the standards of correctness for governance
  - Packaged as role façades
  - Adopted by an agent to support its goals and concomitant policies
  - Help identify *policy points*: where policies apply
- An agent adopts policies that, given its role façades and goals,
  - Support discharging its liabilities
  - Potentially exploit its privileges
  - May not individually or collectively comply with norms
  - May thus violate some security expectations

Governance and Policies: Two Kinds of Interaction

Conversations with autonomous parties; control over resources



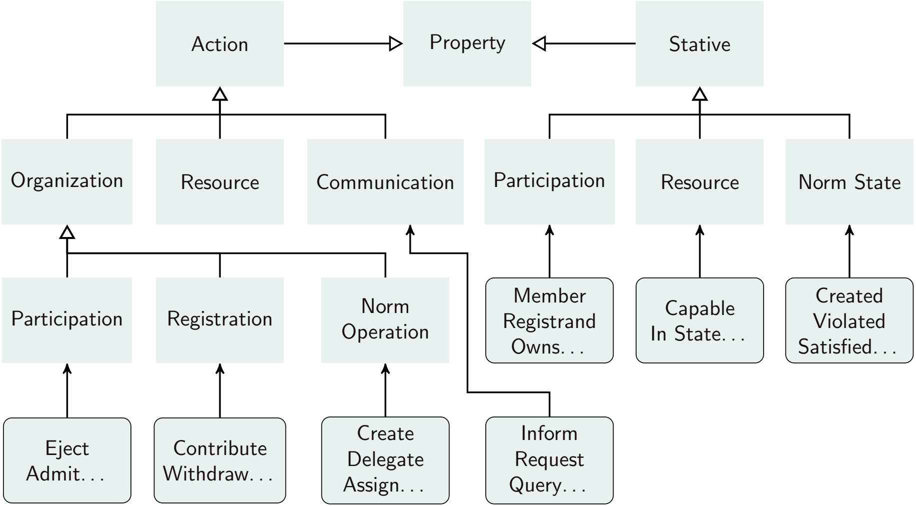
Governance and Policies: Information Model

Relevant information

- Attributes of the parties involved
  - Qualifications, affiliations
- Attributes of the capabilities involved
  - Interactions to be carried out upon resources
  - Collated as interaction types and resource types
- Attributes of the relationships among the parties involved
  - Participations in different Orgs
  - Arrangements among Orgs (captured as participations)
  - Ongoing interactions

Vocabulary for Governance and Policies

Norms and Orgs



Interactions and Policy Types

Going beyond access control

- Understand a policy in terms of its cause and its effect
- Cause
  - *Reactive:* triggered by a request from another stakeholder
  - *Proactive:* triggered by local observations
- Effect
  - *Authorization* of action to be taken on behalf of requester
  - *Enablement* of action, which would otherwise not be taken
  - *Expectation* of action, which would now be performed

An Information Model and Commitment Specification

```
TakeCharge(tcID, nuID, phID, patID, tcThreshold) key tcID
CardiacEvent(ceID, nuID, phID, patID, ceMagnitude) key ceID
CPR(cprID, nuID, phID, patID, cprDuration) key cprID
```

```
commitment CardioCare nuID to phID
create TakeCharge
detach CardiacEvent [, TakeCharge + 180]
  where ceMagnitude >= tcThreshold
discharge CPR [,CardiacEvent + 5]
```

A Cardio Care commitment from a nurse to a physician is

- *created* upon Take Charge;
- *detached* if a CardiacEvent for this patient happens above the specified threshold within 180 minutes
  - Else the commitment *expires*
- *discharged* if CPR on this patient happens within five minutes of the Cardiac Event (else *violated*)

Generate Log Schema

```
CREATE TABLE TakeCharge (
  tcID VARCHAR(10), nuID VARCHAR(10), phID VARCHAR(10),
     patID VARCHAR(10), tcThreshold VARCHAR(10),
  stamp DATETIME,
  PRIMARY KEY(tcID)
);

CREATE TABLE CardiacEvent (
  ceID VARCHAR(10), nuID VARCHAR(10), phID VARCHAR(10),
     patID VARCHAR(10), ceMagnitude VARCHAR(10),
  stamp DATETIME,
  PRIMARY KEY(ceID)
);

CREATE TABLE CPR (
  cprID VARCHAR(10), nuID VARCHAR(10), phID VARCHAR(10),
     patID VARCHAR(10), cprDuration VARCHAR(10),
  stamp DATETIME,
  PRIMARY KEY(cprID)
);
```

Generate Canonical Queries for Accountability Checking

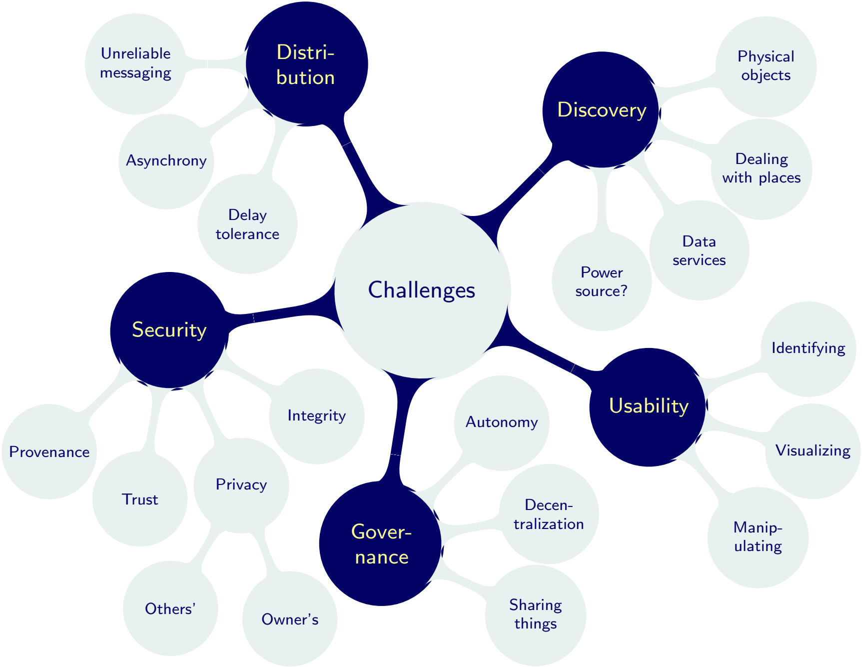
In relational algebra (Jun Yang's notation)

Query for which Cardio Care commitments are detached

```
((\select_{(stamp >= stamp38)} (
 (TakeCharge) \join
 (\rename_{ceID, nuID, phID, patID, ceMagnitude, stamp38}
  (\select_{ceMagnitude = tcThreshold} (CardiacEvent)))))
     \union
  (\select_{(stamp >= stamp37)}
    ((\select_{ceMagnitude = tcThreshold}
      (CardiacEvent)) \join
      (\rename_{tcID, nuID, phID, patID, tcThreshold, stamp37}
       (TakeCharge)))));
```

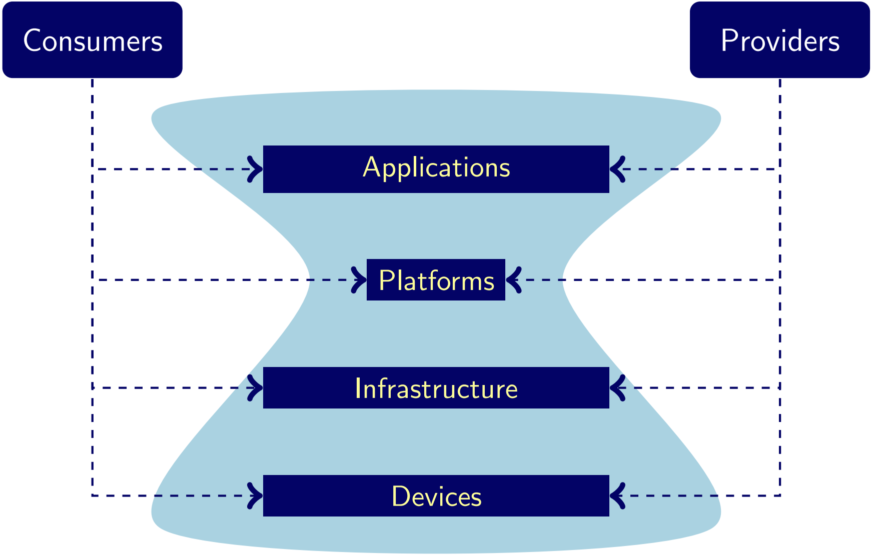
Challenges and Partial Recent Progress

- Storing and retrieving events to determine the state of a norm
  - Mapping commitments to relational algebra [AAAI 2015]
- Maintaining alignment of views despite decentralization
  - Communications to guarantee (eventual) alignment [AAMAS 2015]
  - TBD: maximizing partial or "quick" alignment
- Designing protocols and Org contexts for monitorability
  - Failure of compositionality of monitorability [IJCAI 2015]
  - Automatically close a context to ensure monitorability
- Designing protocols and Org for robustness and resilience
  - Typology of sanctions and sanctioning processes [Draft]
  - TBD: Formalization of normative robustness and resilience
  - TBD: Reasoning about sanctions for design of Orgs
- Design processes conducive to autonomy
  - Abstract formal model of a sociotechnical design process [RE 2014]
  - TBD: Methodologies


Challenges
Distri-
bution
Unreliable
messaging
Asynchrony
Delay
tolerance
Discovery
Physical
objects
Dealing
with places
Data
services
Power
source?
Security
Provenance
Trust
Privacy
Others'
Owner's
Integrity
Gover-
nance
Autonomy
Decen-
tralization
Sharing
things
Usability
Identifying
Visualizing
Manip-
ulating

IoT Hourglass: Few Platforms for Many Applications

Benefit from the upper layers; excitement from the lower layer



Summary and Directions

Exercise: Collective concept map

- What theme do you remember most from today?
- What additional high-level themes should we consider within
  - Artificial intelligence?
  - Distributed computing?
  - Information systems?
- What IoT research question would be worth pursuing?

Thanks and Plugs

- Acknowledgments
  - US Department of Defense
  - US National Science Foundation
- Read and publish in
  - ACM Transactions on Internet Technology
  - IEEE Internet Computing

http://www.csc.ncsu.edu/faculty/mpsingh/