STATUS DIGEST # 3: CADCAD REPRESENTATION

BlockScience

December 18, 2020

OVERVIEW

In order to express the salient features of the STATUS network within the **cadCAD** simulation, testing and decision support framework, it is first represented as a generalized dynamical system (GDS). This representation defines the state space and policy space of the system, and provides an explicit prescription of the state variables and policy mechanisms to be implemented. The formal mapping of the STATUS network to the **cadCAD** implementation is then provided as a series of visualizations of the overall **cadCAD** specification.

Contents

1 Generalized Dynamical Systems (GDS): Introduction and Structure

cadCAD (https://cadcad.org) is a simulation, testing and decision support framework for a wide class of generalized dynamical systems (GDS).[1] GDS encompasses nearly all systems that include a *controller* or *policy* that impacts the transition of a system from one state to another (a state *update*). The **cadCAD** framework can faithfully reproduce the evolution of a GDS as a *digital twin* of the represented system, while providing an online, near-real-time "living laboratory" for:

- testing innovative control process,
- assessing the effects of system modifications and extensions, and
- informing system stakeholders of both suggested control parameter values and of potentially new control levers that fulfill one or more system objectives.

cadCAD is inherently *event-driven*, allowing both a flexible arrival flow of information and an asynchronous decision-making environment to be modeled.

1.1 A Brief Introduction to GDS

The **cadCAD** framework requires an understanding of the *system goals* or objectives of the GDS to be modeled, from the point of view of its designers, stakeholders and users. To that end, the framework is distinguished by having two interrelated processes—the state update process and the control policy process—that act *as if* they are engaged in a dynamic game. That is, in response to an existing state a policy process dictates which control to select, while in response to an existing control an update process indicates which state, among the various feasible alternatives, is the next to occur. Because **cadCAD** is event-driven, it is not necessary that the system dynamics be partitioned into a fixed, equally-spaced set of time intervals for controls to be decided, or for states to be updated. Rather, states and controls can evolve along different time scales.

For example, it may be that the underlying state update process emulates a continuous time system, using e.g. a system of differential equations f defined over a state space X and an admissible input space U:

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}_t, \boldsymbol{u}_t),$$

where $\boldsymbol{x}_t \in X$ is the current system state, $\boldsymbol{u}_t \in U$ is an existing input, and $\dot{\boldsymbol{x}}$ is the derivative of the state with respect to time, so that over a sufficiently small time interval $dt > 0$,

$$\boldsymbol{x}_{t+dt} = \boldsymbol{x}_t + \dot{\boldsymbol{x}}dt + o(dt) = \boldsymbol{x}_t + f(\boldsymbol{x}_t, \boldsymbol{u}_t)dt + o(dt).$$

[1] In what follows context determines whether the singular or plural is intended when using the GDS abbreviation in place of a single system, or to describe multiple examples or a class of such systems.

Note that the system continues to evolve regardless of whether or not $\boldsymbol{u}_t$ is adjusted over the time period dt. Indeed, it might be that the control is updated using an input map g only "every so often", i.e. at a set of $m < \infty$ discrete points in time $(t_0, t_1, \ldots, t_m)$, so that for a given time t,

$$\boldsymbol{u}_j = g(\boldsymbol{x}_t, \boldsymbol{u}_{j-1}),$$

where $t_{j-1} \leq t \leq t_j$ and (cf. equation 1.1) we write $\boldsymbol{u}_t := \boldsymbol{u}_{j-1}$ whenever these arrival time inequalities are true.[2]

Although it is not the case that *every* dynamical system is readily represented as a GDS, it is generically possible to represent most *control* systems in this fashion.[3] This relies upon a 'translation' from the system of interest into the representation framework given above. Once such a translation is available, a mapping between the components of **cadCAD** and the representation framework can be performed (see Section 2, p. 9).

1.2 The STATUS network as a GDS

The STATUS network may be represented as a GDS by identifying the following components:

1. the *state space* X;
2. the *policy space* or *admissible input space* U;
3. the *law of motion* or *state update map* f; and
4. the *policy* or *input map* g.

The multiple terminologies for e.g. U, f and g are due to the intersection between traditional control-theoretic terminology, listed first, and their equivalent terminology in the GDS representation.[4]

In what follows the articulation of the state space and the policy space is performed in detail, while Section 2 discusses the implementation of the law of motion (state update map) within the context of the **cadCAD** model implementation– this leverages the `partial_state_update_block`, or PSUB, component of **cadCAD** to describe how, in response to policy decisions, the state of the system is 'mutated' to a new state.

The final component, the policy or input map, summarizes the *behavioral* mechanisms that cause a decision-making actor to select one input action over other possibly feasible actions. The **Client** and **Contract** component *incentive* input maps are discussed at length in STATUS Digest #2, while the *architectural* (or mechanical) input maps are outlined in STATUS Digest #1 and are here subsumed into the discussion of the policy space for brevity.

[2] It must also be that case that $t + dt < t_j$ whenever the arrival inequalities are satisfied—as dt is treated as an infinitesimal this poses no formal problems when the state is modeled as a continuous time system of differential equations, but care must be taken when f represents instead a discrete system of difference equations, or a discretization of a continuous time model for computation.

[3] Cf. Zargham and Shorish, *Generalized Dynamical Systems* (2021, in progress), for a presentation of the GDS representation for dynamical systems.

[4] Ibid.

1.2.1 The State Space

The STATUS network possesses both *global* and *local* States. A network Client acts as part of the peer-to-peer (p2p) network that facilitates the communication and computation tasks of the network—it possesses local States, i.e. States that are in the information set of the Client alone, as well as one global State, its global identifier. As Clients are the backbone of the network, we commence with definitions of the Client states (cf. STATUS Digest #1 for a description of the notation used in the following) and follow with succeeding State definitions for the App and Contract components.

State Variable 1: Client Identity

The Identity is a global state: a network address h_i of Client $i \in V$.

State Variable 2: Client Neighborhood

The Neighborhood is a local state: the Client neighborhood N_i of a Client i.

State Variable 3: Client Neighbor Accessibility

Neighbor Accessibility $p_i(h_j)$ is a local state, determined by the outcome of a ping from the Client i to a neighbor j with address h_j, or by a ping or Message received from a neighbor j to the Client.

State Variable 4: Client Message Buffer

The Message buffer M_i is a local state: a list of Messages received by Client i and locally stored.

State Variable 5: Client File List

The File list F_i is a local state: a list of Files that Client i is currently storing.

State Variable 6: Client Trust Ledger

The Trust Ledger T_i is a local state: a list of Client i's neighbors and their associated trust scores, in a tuple (j, t_j) for $j \in N_i$.

The App component acts as the gateway for Entities to engage in the network. As such they possess a unique global state (their identifier) and also one or more Accounts that they use to facilitate value transfer in response to Message events.

State Variable 7: App Identity

The Identity is a global state: a network address h_j of **App** $j \in K$.

State Variable 8: App Account

The Account is a global state: a list of **Account** addresses a_j associated with **App** j.

The **Contract** component acts as part of the incentive mechanism of the p2p network (cf. STATUS Digest #2), and enables trust in the mechanism by having both its smart contract and its state variables available globally throughout the network. Because the **Contract** component resides on-chain, its smart contract **Account** address is also its identity (we abstract away here from the realistic situation in which more than one address represents the 'complete' **Contract** component, but this may be introduced into the below specification without difficulty by specifying e.g. vectors of **Account** addresses representing different smart contract or bookkeeping locations).

State Variable 9: Contract Identity

The Identity is a global state: an **Account** address a_c of the **Contract** component.

State Variable 10: Contract Ledger

The Ledger is a global state: a list L of submitted route claims from **Message** recipient or proving **Clients**. For each claim submitted, a (cryptographic hash of a) tuple $\{\mathbf{a}, \mathbf{h}, t_R, a_p\}$ is stored in L, where $\mathbf{a} := (a_{h_1}, a_{h_2}, \ldots, a_{h_J})$ is the vector of **Account** addresses of **Client** identities $\mathbf{h} := (h_1, \ldots h_J)$ participating in the route, t_R is the time taken to fulfill the Request, and a_p is the address of a **Client** p submitting the claim.

Note that under this formulation is it *not* the case that the **Contract** component necessarily verifies the route claim is correct. As discussed in STATUS Digest #2, the **Contract** component is normally passive, and only randomly conducts an *audit* on submitted claims in order to incentivize **Clients** to engage in activity that supports the network's system goals.

To summarize, the **state space** X of the network is composed of both local and global state spaces for each component group:

$$X := H \times N \times P \times M \times F \times T \times A \times L,$$

where

1. H is the *identity address space* (usually a space of possible cryptographic hashes, e.g. $\mathbb{Z}_{2^{256}}$);

2. $N = V$ is the *neighborhood space*—each N_i is a subset of this space;
3. $P = V$ is the *accessibility space*—each P_i is a subset of this space;
4. M is the *message space*, e.g. the space of all messages of a maximum size—each M_i is a subset of this space;
5. F is the *filename space*, e.g. the space of all filenames of a maximum size—each F_i is a subset of this space;
6. $T = N \times [0, 1]$ is the *trust ledger space*—each T_i is a subset of this space;
7. A is the global **Account** *address space* (usually a space of possible cryptographic hashes, e.g. $\mathbb{Z}_{2^{256}}$);
8. L is the *ledger space*, consisting of all possible routes of maximum length $|V|$ between **Clients** in V.

1.2.2 The Policy Space

The policy space governs the implementation of decisions made at various levels of the network. The natural decision-making agent is an **Entity** that decides whether or not to send a **Message** via an **App**. This will depend upon factors such as the opportunity cost of using the network (i.e. the ability to send a **Message** by other means), the direct cost of sending a **Message** (both in terms of units of account and in terms of time taken), and, ultimately, the benefit derived from using the network. Although a full simulation framework can incorporate **Entity** decision-making, the initial **cadCAD** simulation abstracts away from this agent and concentrates solely upon the decisions of the **Client** and **Contract** components. Incorporating **Entities** into the network is discussed further in the fourth STATUS Digest.

Furthermore, the **App** component may also be modeled as a decision-making agent in its own right—for example, it may reject requests from **Clients** to transfer payment, in the case that insufficient funds exist in the associated **Account**. However, here we abstract away from the distinction between the **App** and the **Client** and treat the **Client** as the 'primal' decision-making agent, so that both the aforementioned accounting decisions and the more deliberate value transfer decisions, such as "I've decided not to pay for a service I've received", can be modeled with the same underlying p2p member. This is again an abstraction that can be refined in future versions of the implementation (cf. the fourth STATUS Digest).

Client component behaviors are categorized by their messaging (or file transfer) decisions, such as initiating a request by sending a **Message** or deciding whether or not to pass on incoming **Messages**, and their claim and payment decisions, such as whether to submit claims of fulfilled routes to the **Contract** component or whether to transfer promised funds in the event that a request was successfully routed.

Policy 1: Client Neighbor Identification

A **Client** may decide to send a ping to another **Client** in its neighborhood, to ascertain if the **Client** is accessible.

Policy 2: Client Message Demand

A **Client** may decide, on the basis of an **Entity** decision relayed through its **App**, to initiate a request by sending a **Message**.

Policy 3: Client Message Routing

A **Client** may decide, upon receiving a **Message** from a **Client**, to route the **Message** to its neighbors. This may depend upon the trust score of the transmitting **Client**, as described in STATUS Digest #2.

Policy 4: Client File Routing

A **Client** may decide, upon receiving a **Message** requesting a **File** that it stores, to deliver the **File** to the requester.

Policy 5: Client Claim Submission

A **Client** may decide to submit a claim that a particular route has fulfilled a request, or a batch of such claims, to the **Contract** component.

Policy 6: Client Payment Transfer

A **Client** may decide to initiate payment transfer from an **Account** held by the **App** it is associated with, in response to the fulfillment of a request it created.

Policy 7: Client Trust Update

A **Client** may decide to update the trust score of one or more of its neighbors in its Trust Ledger, in response to its own experience or in response to a global audit report from the **Contract** component.

The **Contract** component's decision-making is built into its smart contract representation. As such it does not contain discretionary powers, but its actions nevertheless constitute behaviors because they act to change the state of the system.

Policy 8: Contract Ledger Recording

The **Contract** component records routes submitted to it from **Clients** into the **Contract** Ledger.

Policy 9: Contract Route Audit

The **Contract** component randomly selects a route to audit, and decides upon the outcome of the audit.

Policy 10: Contract Audit Penalty

If an audit outcome indicates that a route was fulfilled, but determines that

- the route participants were not compensated as promised by the request initiator, or
- the route contains extraneous self-routing, or
- the route was inefficient (more hops than required),

then the **Contract** component broadcasts the audit outcome to all **Clients**.

2 The cadCAD Specification of the STATUS Network

The **cadCAD** representation of the STATUS network uses the state and policy spaces to implement both the state update map f and the policy (input map) g over a defined state space X (cf. Section 1). It does so by defining logic that provides, conditional upon parts of the update space remaining fixed, a set of *partial state update blocks*, or PSUBs, which indicate how to update a state given its current value and the value of one or more policy variables that impact its state transition. Once updated, state variables are also used to create *metrics*, which provide qualities of the network that are used to assess the network's performance relative to its system goals. Examples of metrics that have been implemented to date in the STATUS network **cadCAD** representation include e.g. network traffic and value (tokens) in transit, the distribution of waiting times from a service request to its fulfillment, and statistics related to the depth of knowledge that **Clients** have regarding their neighborhoods (the Distributed Hash Table, or DHT, that each **Client** keeps).

The full **cadCAD** specification diagram is presented in Figure 1, which provides a bird's-eye view of the flows between and among system components. Rows in this Figure represent the 'event steps' that are taken in moving from a current state (first row) to a future state (fourth row) via policy decisions (second row) and their impact (third row). Columns in the Figure represent the different PSUBs that update state variables according to the processes, such as routing, trust updating and rewarding/penalizing, that the system is designed to implement.[5]

Succeeding figures are presented as 'maps' (Maps 1 - 10), which are visualizations examining the different PSUB subsystems presented in Figure 1. The Maps provide a visual mapping between the the **cadCAD** structural framework described in Section 1 and its implementation.

[5] PSUBs in Figure 1 are non-consecutively numbered from left to right due to the evolution of the specification as it was developed; the associated Maps are numbered consecutively for clarity in what follows.

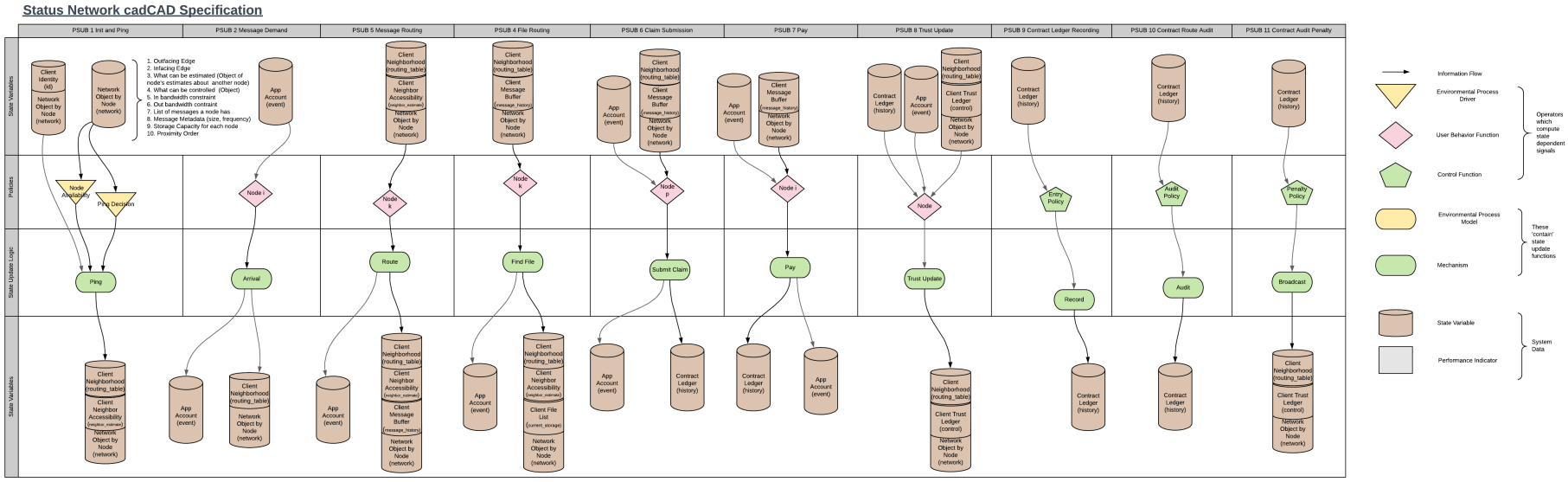
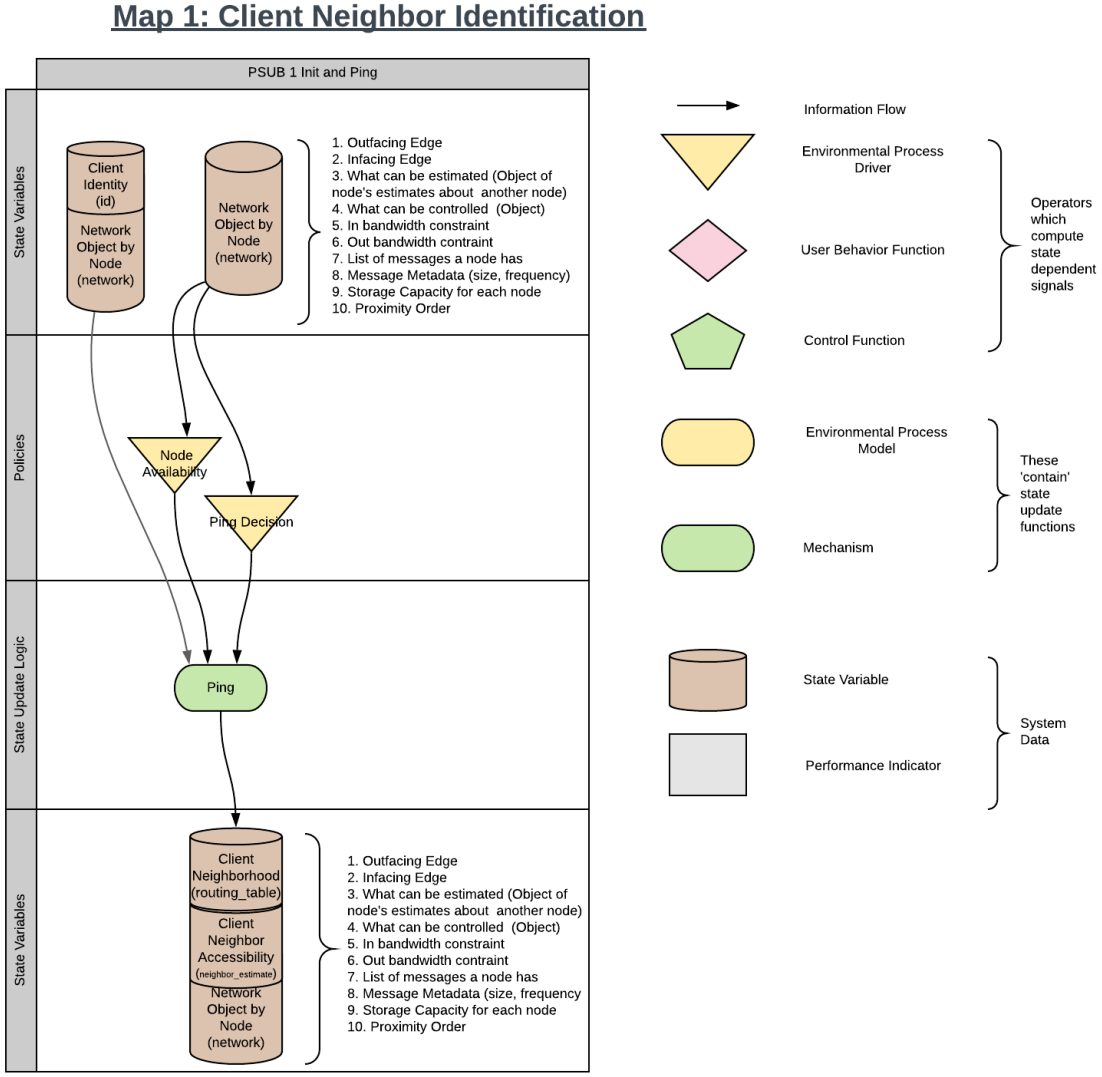


Figure 1: **cadCAD** System Specification of the STATUS Network

2.1 Client Neighbor Identification

Identifying the neighbors of a Client (Policy 1, p. 7) is depicted in Map 1, which outlines PSUB #1 “Init and Ping”. The local State variables are the Client identity, Client neighborhood and Client neighbor accessibility, along with a global “State” given by the network itself. Using the Client neighbor identification policy, a Client can decide whether or not to send a ping to determine Client availability. The ping itself updates the state of the Client’s neighborhood as a result of the ping, and updates the Client neighbor accessibility accordingly.

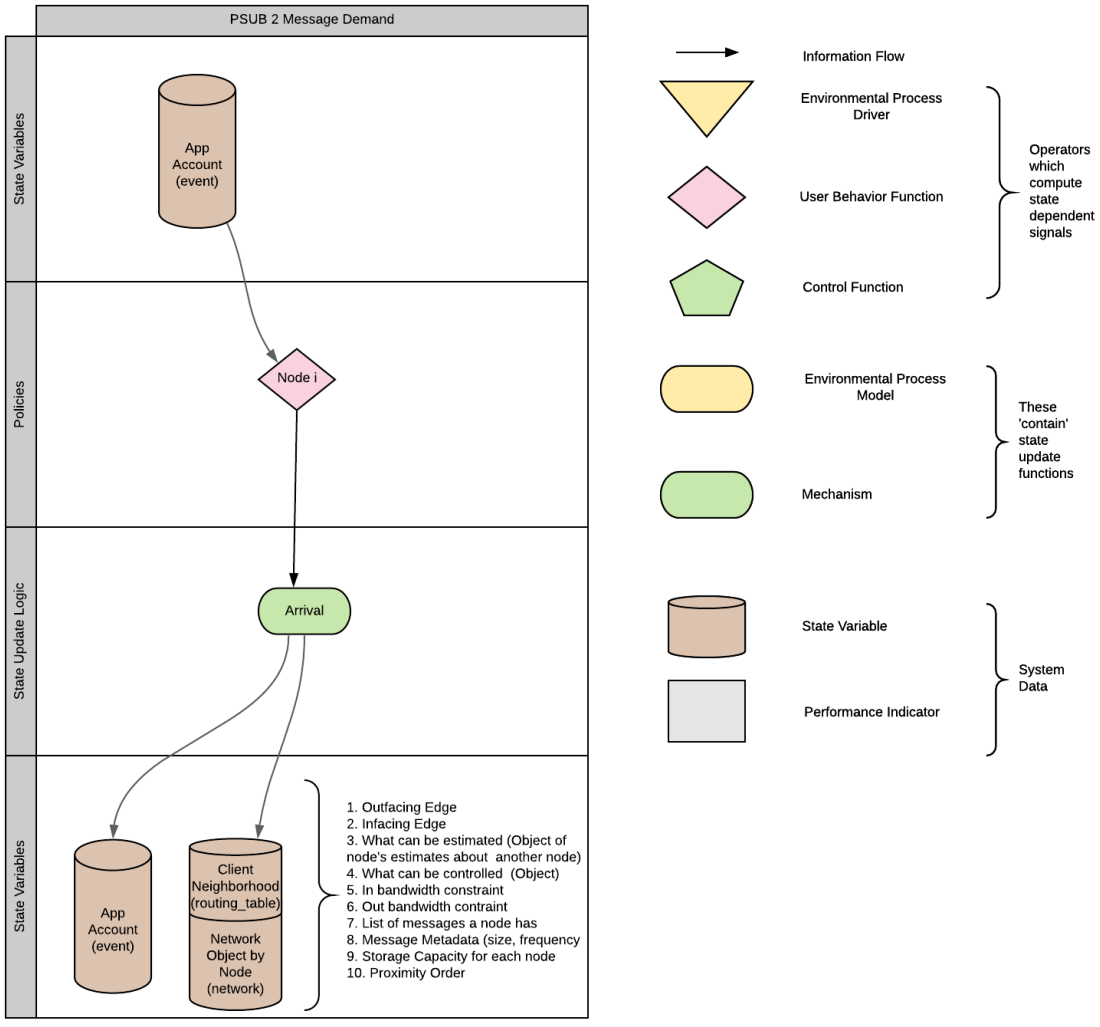
Map 1: Client Neighbor Identification



2.2 Client Message Demand

The demand for a Message by a Client (Policy 2, p. 8) is given in Map 2 and represents PSUB #2 “Request Initiation”. The state of the Client is its connected App, which fires an Event when the Entity using the App initiates a request. The Message demand policy then initiates the request, updating the App that the request has been initiated, and also the Client Message buffers for those Clients in the neighborhood of the initiating Client. As in Client neighbor identification, there is also a global book-keeping State representing the network (*NB* in what follows the network State will be suppressed for brevity).

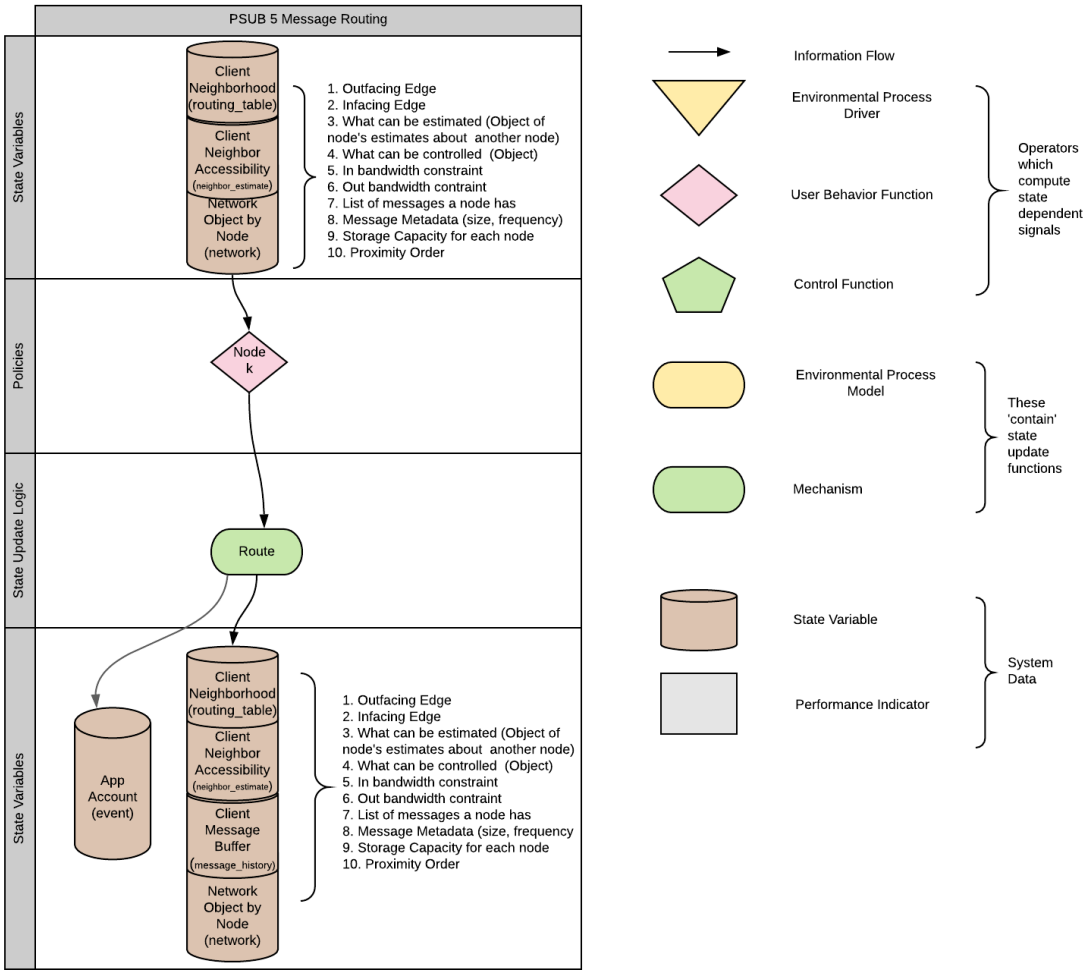
Map 2: Client Message Demand



2.3 Client Message Routing

Once a Message has been initiated it is routed through the network (cf. Policy 3, p. 8), as given in PSUB #3, "Message Routing" and in detail in Map 3. Taking the Client neighborhood, neighborhood accessibility, and the overall network as States, a Client decides whether or not to route an incoming Message according to the delivering Client's trust score. The Message payload Message is updated (to include the Client if the message is passed on, or to do nothing if the Message is discarded)–in **cadCAD** this is reflected in the "route" mechanism, accounting for the Message's status. A Client may or may not update an App on routing, and will update its neighborhood and accessibility States on the basis of any acknowledgments ("acks") it receives upon trying to route a Message. The Client's Message buffer may be updated, although an implementation may not update the buffer if the Message was discarded by the Client.

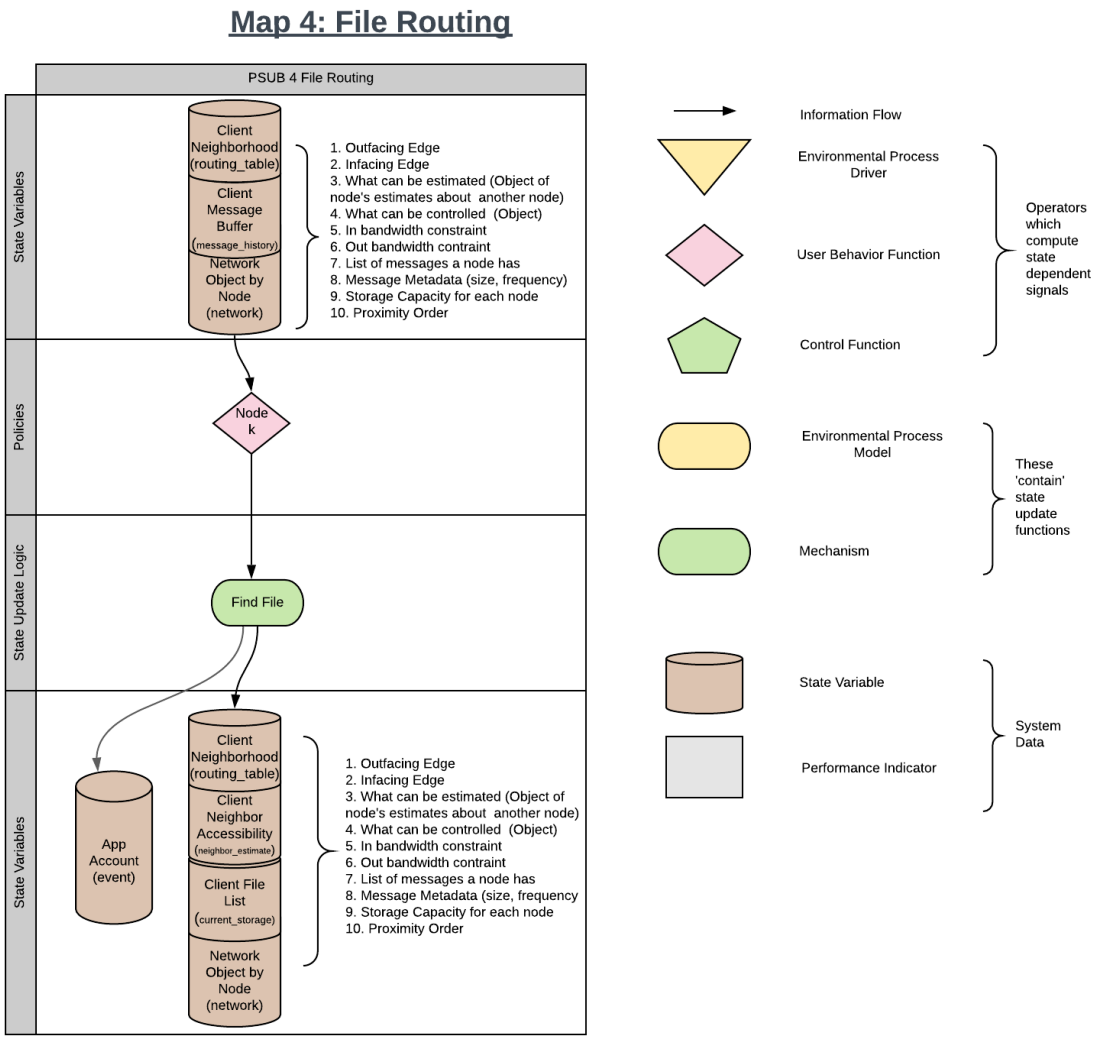
Map 3: Client Message Routing



2.4 Client File Routing

The mechanics of File routing are similar to Message routing, as described in Map 3, above. File routing uses Policy 4, p. 8 and is depicted in Map 4 for PSUB #4 "File Routing". Taking the Client neighborhood, its Message buffer, and the overall network as States, a Client decides whether or not to serve a File that an incoming Message has requested. If the File is served it is found on the Client's system (in **cadCAD** this is reflected in the "Find File" mechanism), and passed on to the requesting Client. A Client may or may not update an App on routing, and will update its neighborhood and accessibility States on the basis of any acknowledgments ("acks") it receives upon trying to serve the File. The Client's File list will also be updated.

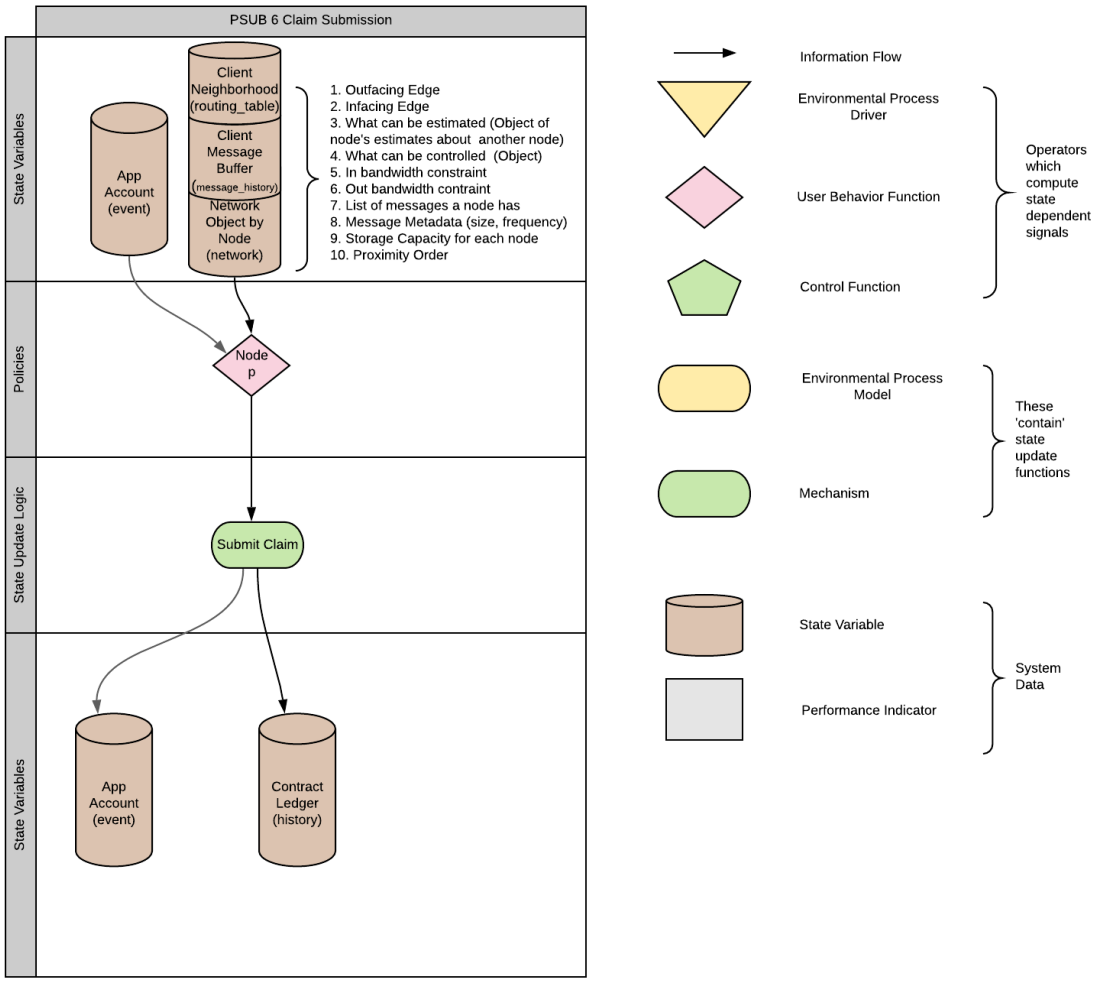
Map 4: Client File Routing



2.5 Claim Submission

When a **Client** believes a request has been fulfilled, it can submit a claim to the **Contract** component (individually, or in batch). This Claim submission Policy (Policy 5, p. 8) is detailed in Map 5 and is **cadCAD**'s PSUB #5, "Claim Submission". The policy acts on a **Client**'s neighborhood and **Message** buffer to select one or more fulfilled requests, and decides whether or not to submit the route(s) of the fulfilled request(s) to the **Contract** component (this is the "Submit Claim" mechanism in **cadCAD**). There may also be confirmation sent to the **Client**'s **App**, updating the **Entity** of the submission.

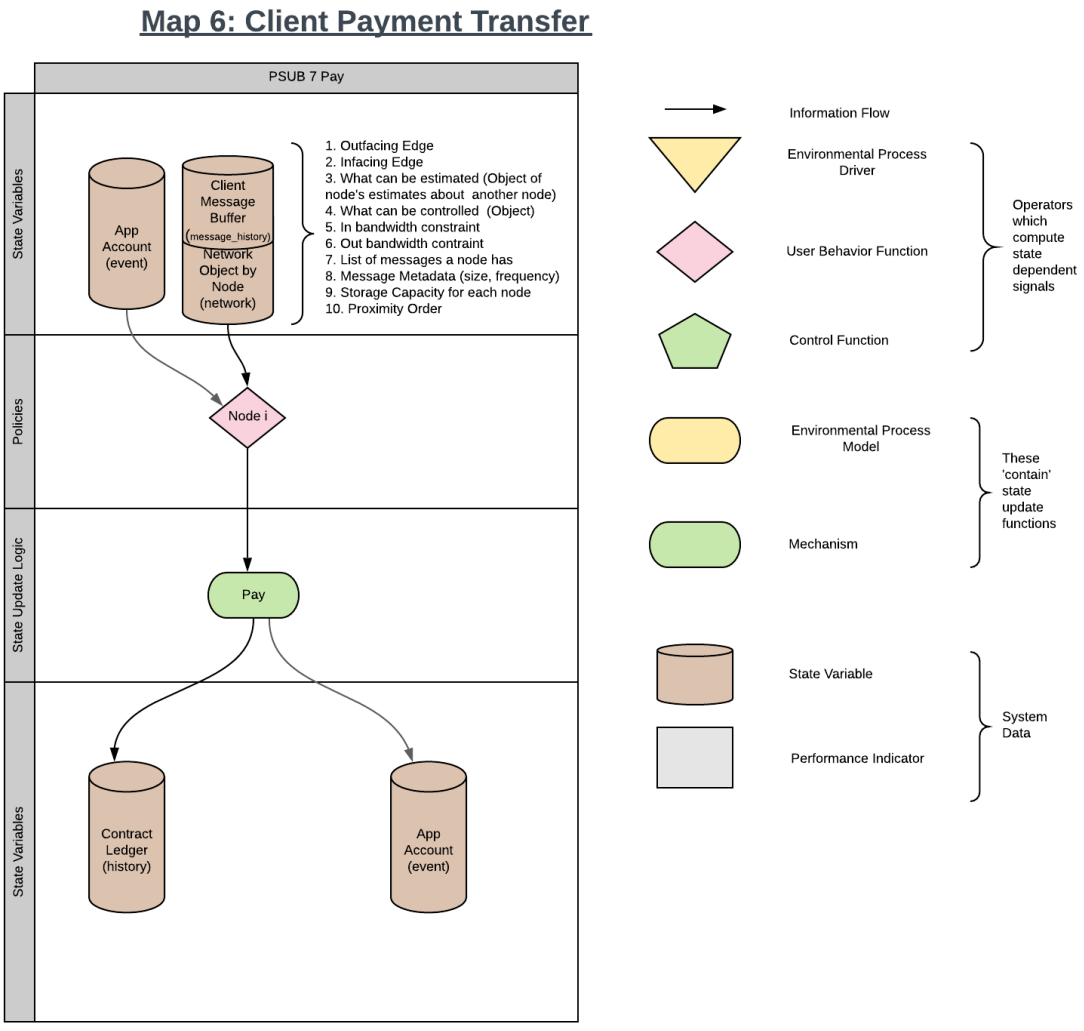
Map 5: Claim Submission



2.6 Client Payment Transfer

If a Client receives notification from the Contract component that a request it initiated has been fulfilled, it must decide whether or not to honor the promised payment (Policy 6, p. 8). **cadCAD** PSUB #6, "Client Payment Transfer", is depicted in Map 6 and describes this policy process. Taking the Account State of its App and its own record of Messages in its buffer, a Client must choose whether or not to release the funds promised in its initiating request in response to a Contract component Event signifying that the request was fulfilled. The "Pay" mechanism in **cadCAD** implements the State update of the policy decision, resulting in an update of the Contract ledger (recording a successful payment) and an updated Account balance held by the App.

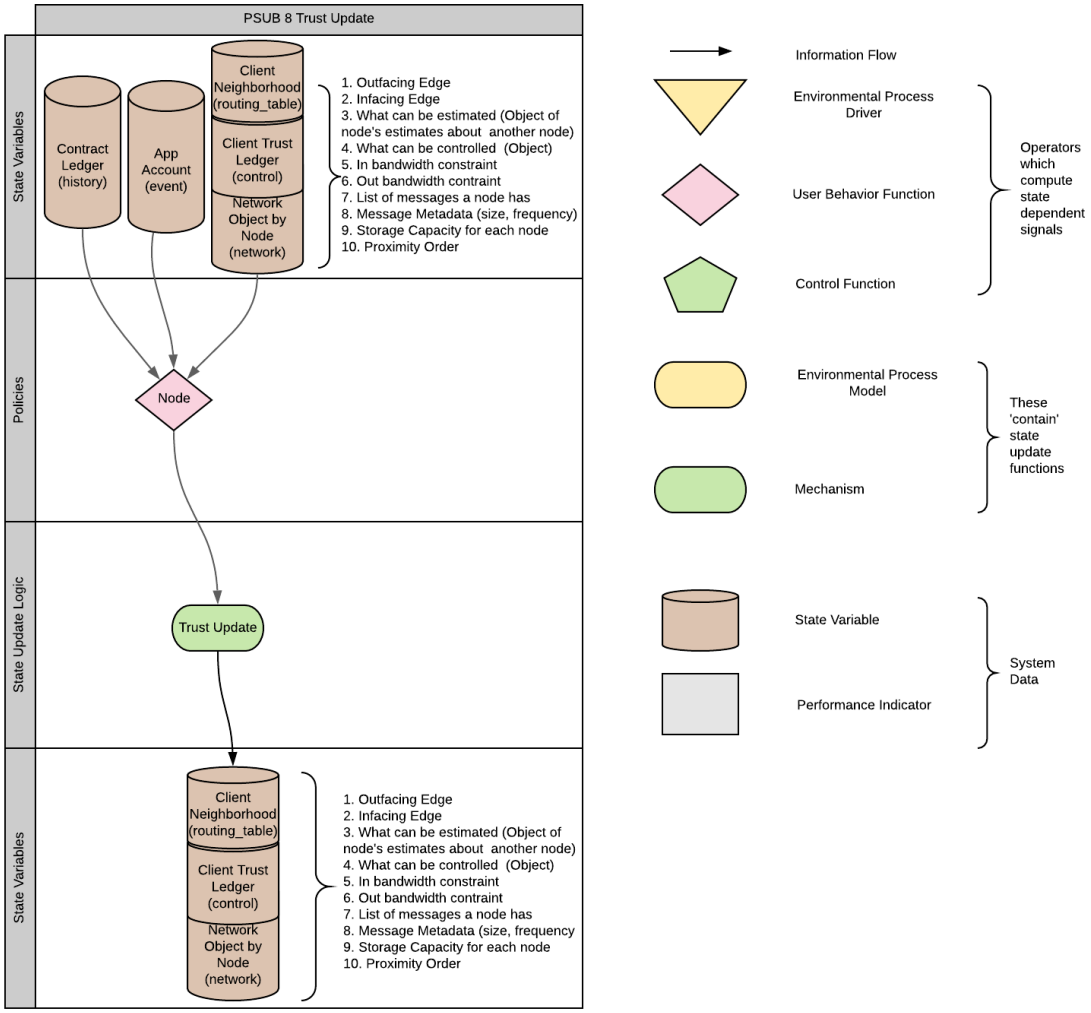
Map 6: Client Payment Transfer



2.7 Client Trust Update

If a Client experiences payment or non-payment for services rendered, or receives audit information from the Contract component about another Client, it can decide whether or not to update its trust ledger (Policy 7, p. 8). Map 7 is a detailed view of **cadCAD** PSUB #7. It shows the relevant States to the decision—the Contract ledger, an App Account, and the Client's neighborhood and trust ledger—and applies the decision on updating the trust score. This updates the trust ledger (in **cadCAD**, this is the "Trust Update" mechanism) and may or may not update the Client's neighborhood, depending upon whether or not a Client has been "blacklisted" as a result of the update.

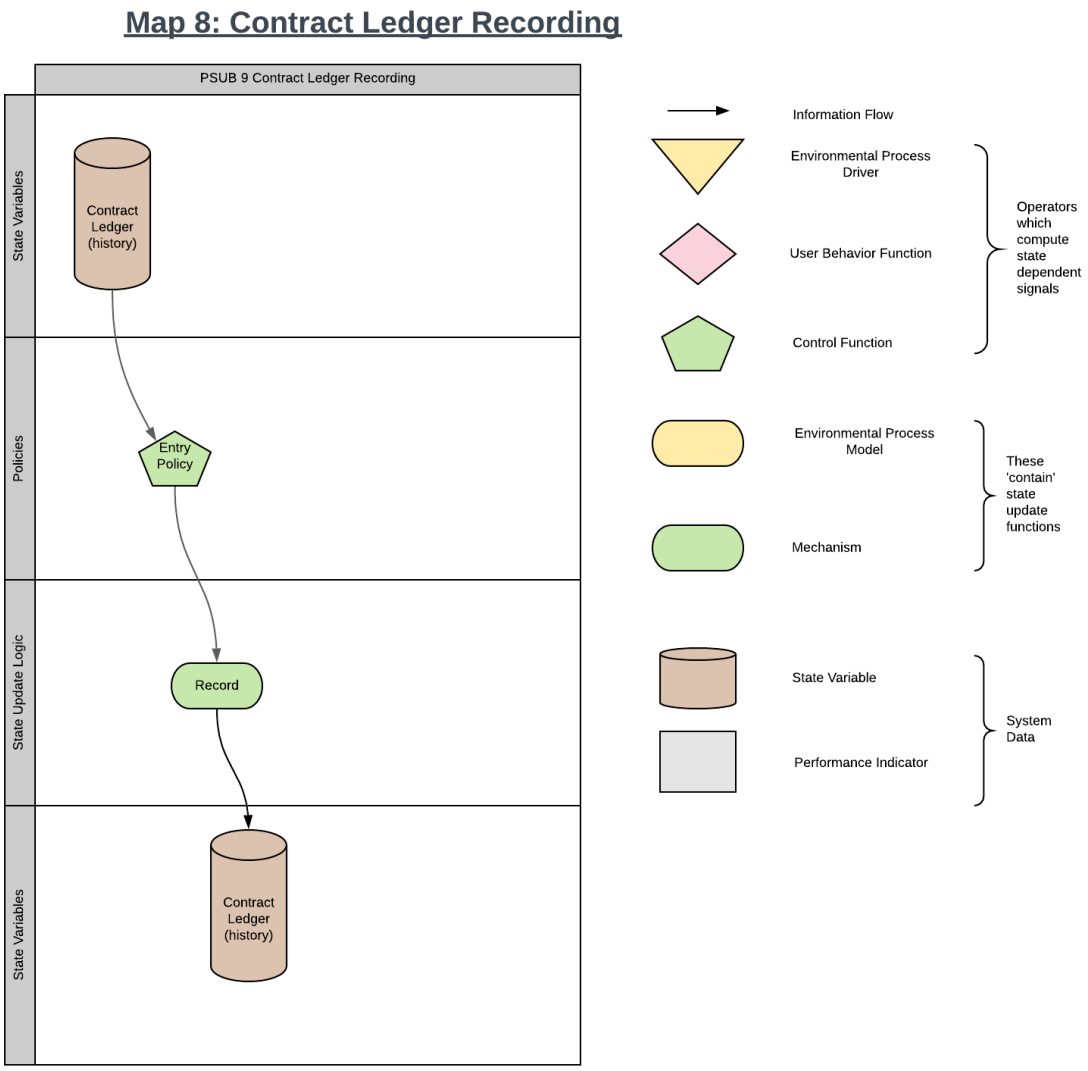
Map 7: Client Trust Update



2.8 Contract Ledger Recording

When the **Contract** component receives one or more routes indicating fulfilled service requests, it updates its **Contract** ledger appropriately (Policy 8, p. 8). The **cadCAD** PSUB #8, "Contract Ledger Recording", is given in Map 8 and shows the updating of the **Contract** ledger using **cadCAD**'s "Record" mechanism.

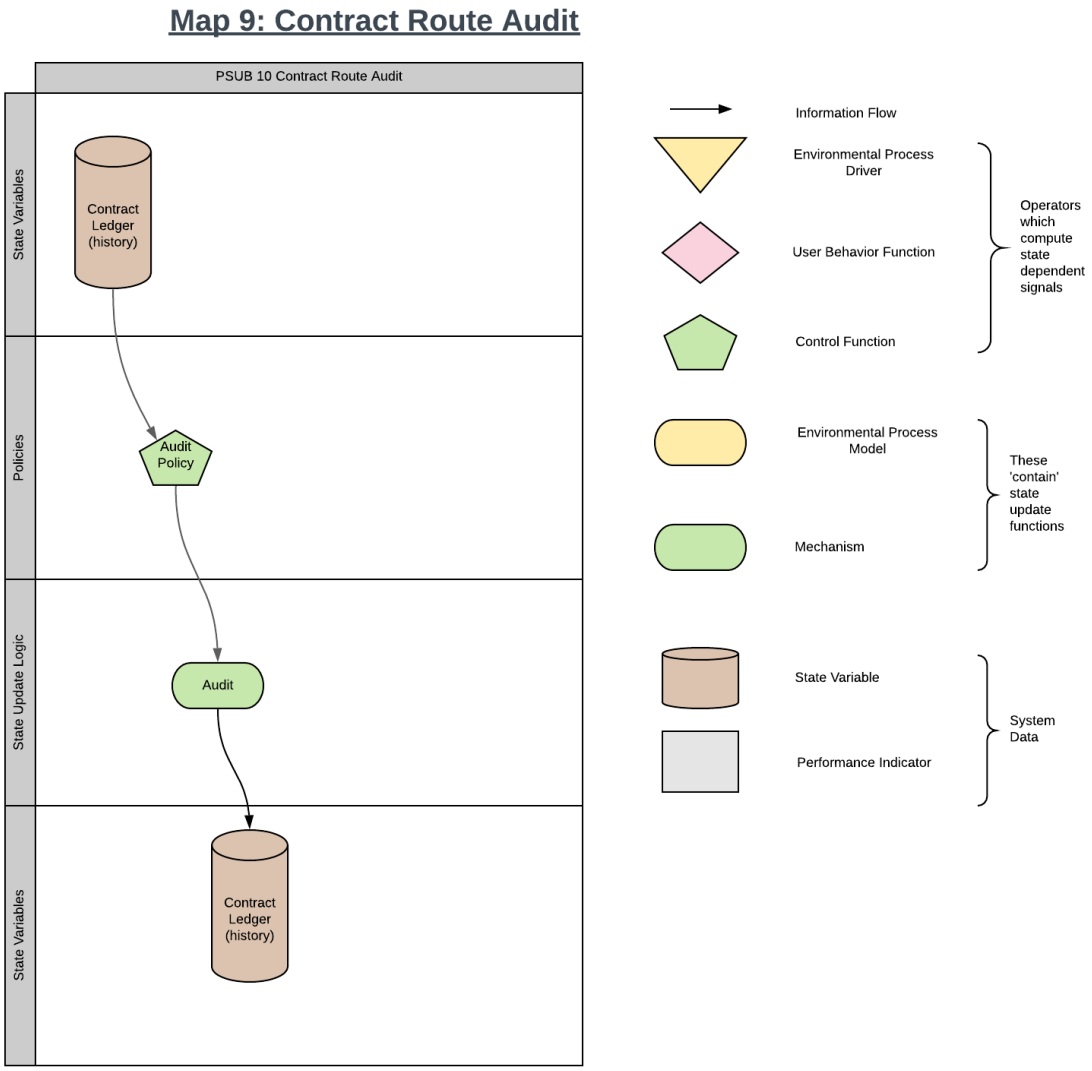
Map 8: Contract Ledger Recording



2.9 Contract Route Audit

The **Contract** component's audit policy (Policy 9, p. 8) is given in **cadCAD** PSUB #9 and depicted in Map 9. The audit policy is an algorithm which randomly selects a route in the **Contract** ledger to audit, and then conducts the audit to see if the route fulfils its required conditions. The **Contract** ledger **State** update that results is implemented in **cadCAD** by the "Audit" mechanism.

Map 9: Contract Route Audit



2.10 Contract Audit Penalty

If the outcome of a **Contract** component audit indicates that the selected route does not fulfil its required conditions, the **Contract** will broadcast the audit outcome to all **Clients** (cf. Policy 10, p. 9). This is **cadCAD**'s PSUB #10 and is given in Map 10. Using the **Contract** ledger as the record of all routes, the policy identifies the **Client** (or **Clients**) that have violated one or more of the conditions for a fulfilled request, and broadcasts (in **cadCAD**, this is the "Broadcast" mechanism) the result to the network. This broadcast may update the **Client** trust ledger (depending upon the Client Trust Update policy) and associated **Client** neighborhood.

Map 10: Contract Audit Penalty