The Membrane

Brief Description of the Project

anton.bondarenko@gmail.com

Membrane Proof of Concept: The Project Goals

To implement and maintain efficiently the necessary logic within the Home Network will require some extended remote application management capabilities which in turn assumes that some kind of application execution environment is installed on the key elements of the Home Network.

...

The main goal of this project from the implementation perspective is the choice and proving the architecture and capabilities of such execution environment.

Membrane Proof of Concept: The Project Goals

The main goals of this project from the functional / business application perspective are:

1) The choice of the relevant model and implementation of mechanisms for security application blocks deployment, integration and running.

2) The choice or development of the relevant method for describing and applying traffic processing rules (the rule engine).

3) The development of the core algorithms for consistency rules extraction; installing and application.

To implement and maintain efficiently the necessary logic within the Home Network will require some extended remote application management capabilities which in turn assumes that some kind of application execution environment is installed on the key elements of the Home Network.

...

The main goal of this project from the implementation perspective is the choice and proving the architecture and capabilities of such execution environment.

In the remaining part the attempt is made to show how these goals achieved using the SM.EXEC execution environment prototype implementation and a set of application blocks and state machines emulating home network elements.

Proof of Concept: The Solution scope

What is included	What is not included (and why)
- SM.EXEC reference implementation – the software event-based asynchronous architecture applied here to disaggregate the network elements. - State machines used to simulate all network elements in Use-Cases. - Application blocks for simulating network behavior. - Driver application putting it all in work and producing timed traces.	- Actual implementation of protocols – protocols are simulated in the minimal amount using JSON. - Networking layer – it is simulated using SM.EXEC queues, Membrane state machines and Membrane application functions.

The Solution Components

The solution consists of three main layers:

- SM.EXEC execution environment. This is a prototype implementation of the SM.EXEC architecture, pertially under development.
- The Membrane library consisting of application functions and state machine descriptions.
- The scenario setup - .c driver used for necessary instances initialization and initial event firing.

The project Repository

Folder or file(s)	Description
app/*.yaml, app/*.json	Definition files of the state machines used by the project (see the description below)
app/dtag_tc_apps.h, app/dtag_tc_apps.c	Application functions used in state machines
lib	3-rd party libraries for parsing JSON and for hash function calculation
test/dtmp/dt	dt_use_cases.c – The Membrane simulator driver
doc	TheMembrane_description.pdf (this file)
log	Sample output file(s) with the results of running the Membrane simulator

DTMP

= Deutsche Telecom Membrane Protocol

- A JSON mock protocol specially developed for this PoC.

- The current list of DTMP commands:

```
"originate",         // 1025
"report",            // 1026
"register",          // 1027
"set_flow",          // 1028
"request_flow",      // 1029
"set_rule",          // 1030
"get_rule",          // 1031
"deploy_rule",       // 1032
"receipt",           // 1033
"pending"            // 1034
```

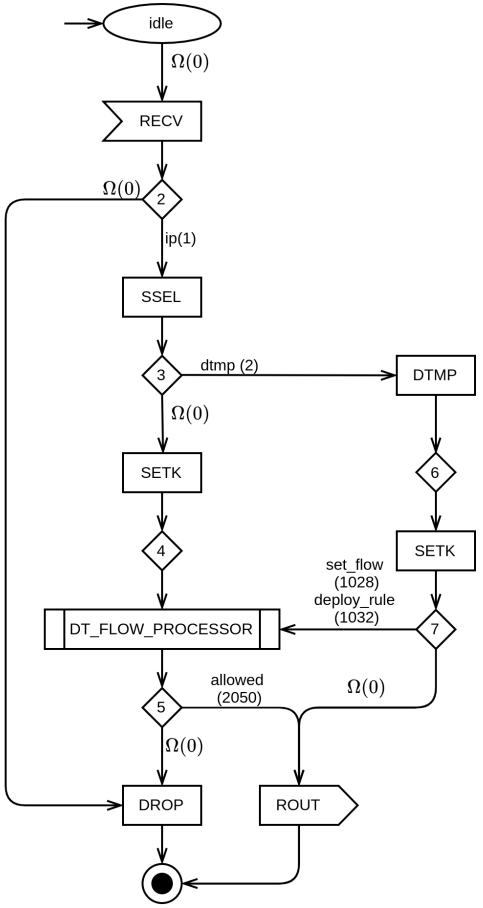
DTMP – Sample Messages
(translated to YAML for readability)

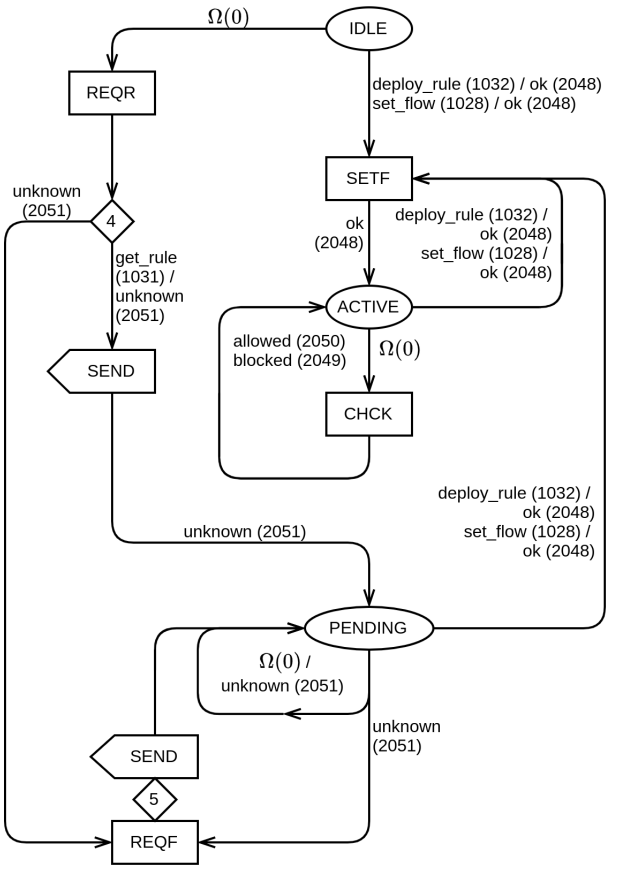
```yaml
---
proto: ip
from: router
to: controller
to_report: true
data:
  proto: dtmp
  command: get_rule
  data:
    to_role: role1
    from_role: role2
    proto: proto0
    key: key0
...


---
proto: ip
from: controller
to: router
to_report: true
data:
  proto: dtmp
  command: deploy_rule
  data:
      from_role: role1
      to_role: role2
      proto: proto1
      decision: allowed
      key: key0
...
```

```yaml
---
proto: ip
from: user
to: controller
to_report: true
data:
  proto: dtmp
  command: set_rule
  data:
      role_from: addr1
      role_:to: addr2
      proto: proto1
      decision: allowed
...
```

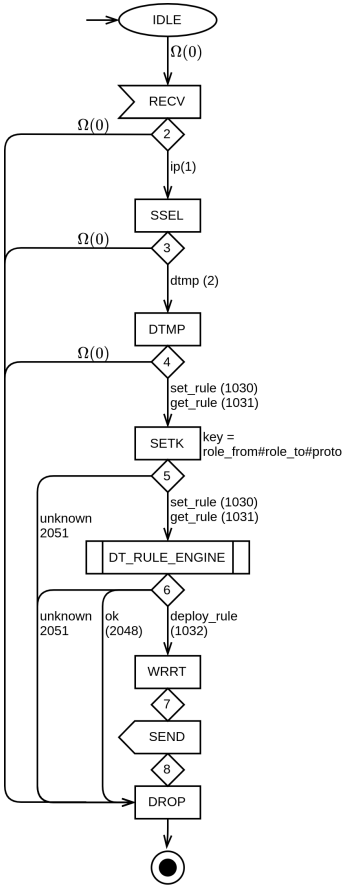
```yaml
---
proto: ip
from: router
to: user
to_report: true
data:
  proto: dtmp
  command: request_flow
  data:
    to: addr1
    from: addr2
    proto: proto0
...


---
proto: ip
from: user
to: router
to_report: true
data:
  proto: dtmp
  command: set_flow
  data:
      from: addr1
      to: addr2
      proto: proto1
      decision: allowed
      [to_role: role1]
      [from_role: role2]
...
```

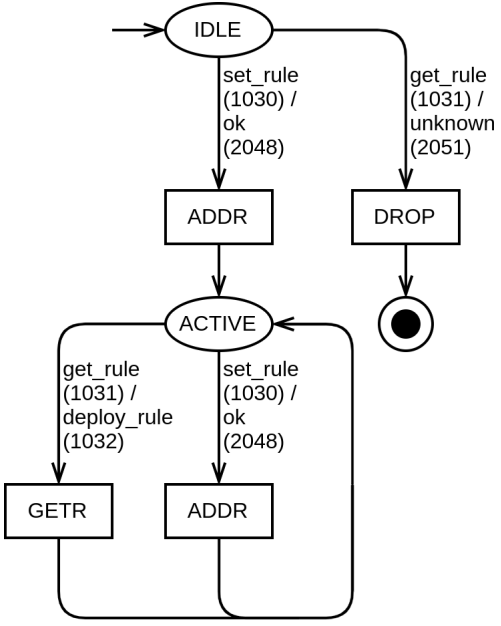
Membrane State Machines: The Router and Flow processor



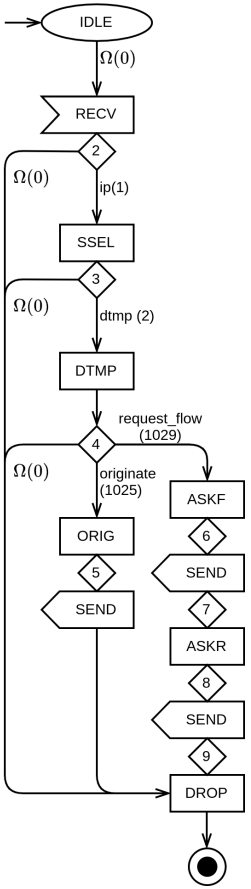


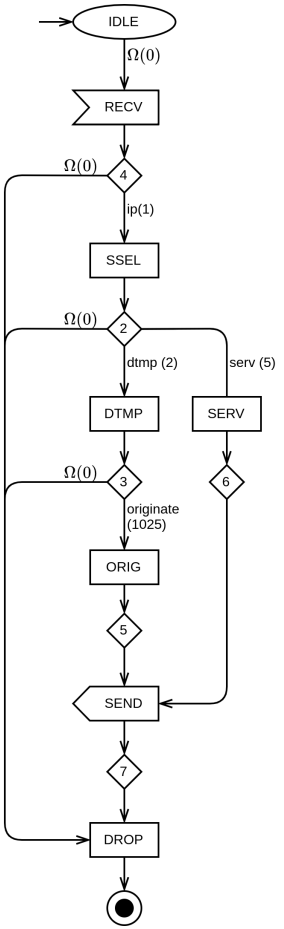
Membrane State Machines: The Controller and Rule Engine



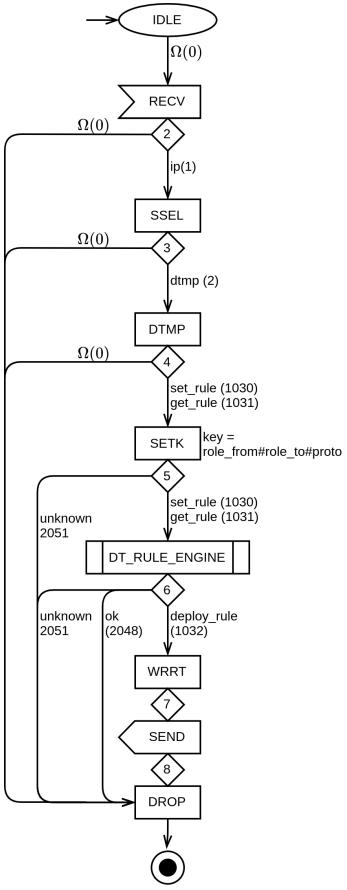


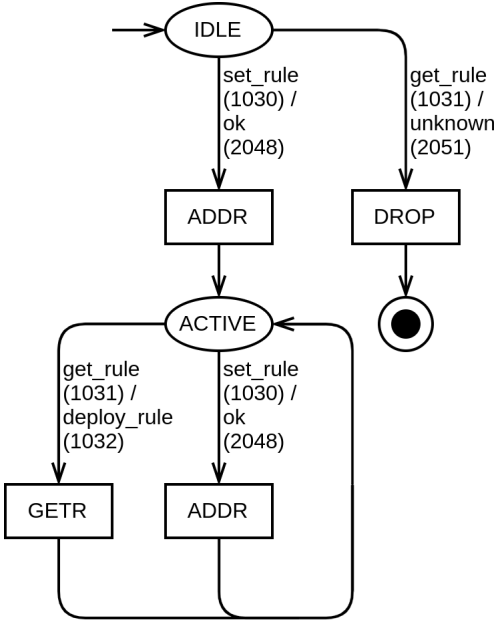
Membrane State Machines: The Generic Client and Server



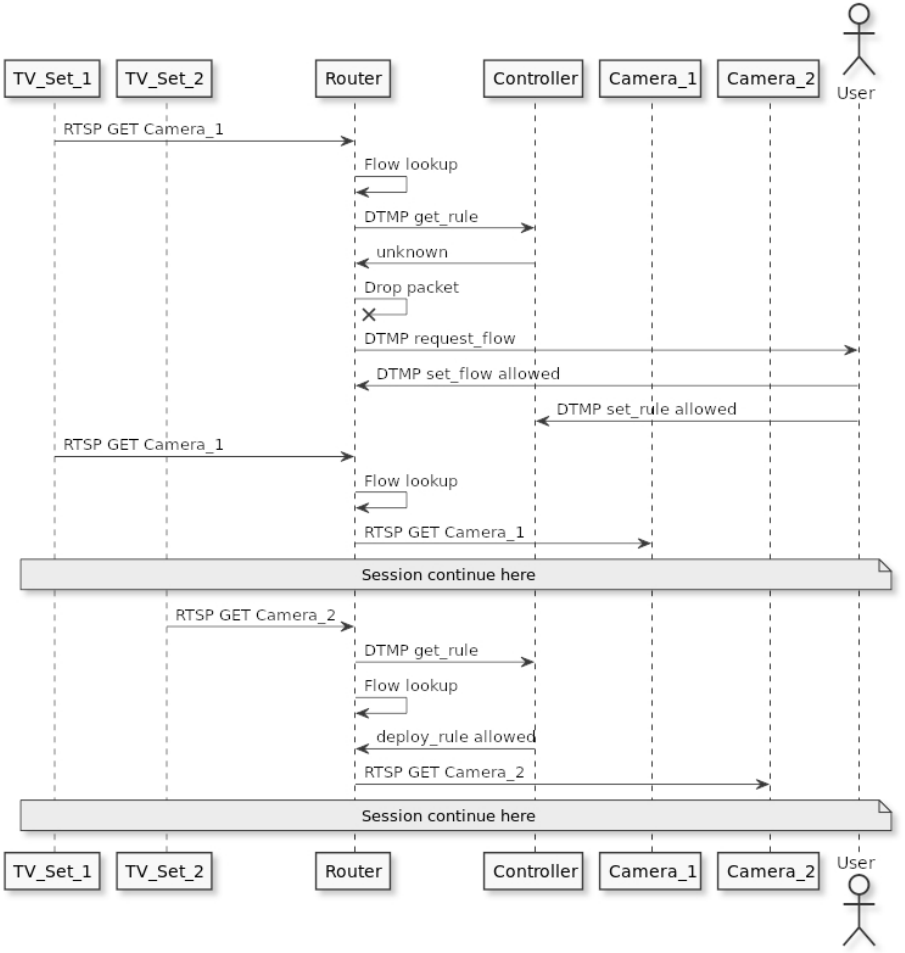


Membrane State Machines: The SIP Server and Session Portions





Membrane State Machines: The Sample Session Call Flow



The Next Steps Towards Production Membrane Solution

1. To develop the production grade SM.EXEC core version(SM.EXEC.core).

2. To implement necessary tools for SM.EXEC applications development automation (SM.EXEC framework).

3. Port SM.EXEC to the target platform(s).

4. Disaggregate target network modules (IPTV, SIP, MQTT, Routing etc.), compose SM.EXEC state machines and re-combine disaggregated solutions.

5. Complete the design of the Membrane Protocol v.1 (DTMP).

6. Build testing and modeling environment based on prepared SM-s and mocked JSON protocol simulation.

7. Having done the 1) - 6) start implementing new versions of home network boxes (routers, gateways, controllers etc.).