

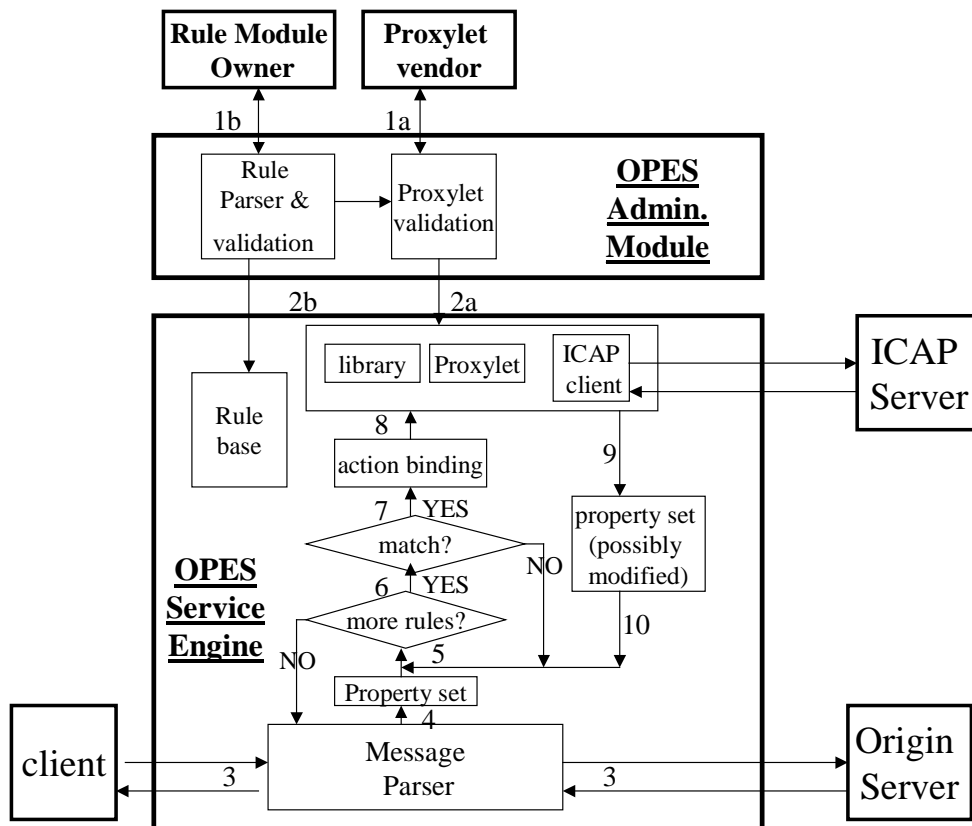
OPES Architecture for Rules Processing and Service Execution

Lily Yang

The following diagram represents my current understanding of the OPES architecture, after having read the Internet drafts on both the OPES framework and the Proxy Specification Rule Language (PSRL). In the center of this architecture are two OPES functional modules – the Administration Module and the Service Engine. The Administration Module interfaces with the parties that authorize the OPES services – rule module owners, with the help of proxylet vendors. This interaction is characterized as the proxylets/rules-downloading phase. The Service Engine is directly involved in the data-path (client-to-server, server-to-client, request served from cache) and hence the interaction is characterized as the Service Execution phase.

Conceptually, the OPES Administration Module and the OPES Service Engine are two separate functional modules. The Service Engine is in the content path, while the Admin Module is not. Practically, they can reside either on one physical box or two separate boxes – it is totally up to the OPES vendor. It could be offered in one box as a OPES all-in-one appliance (more suitable for small ISPs), or it could be set up with one central Admin box and many edge OPES Service devices (ideal for CDN).

This document simply summarizes the inner working of these two phases and lists some issues that I consider important in each phase.



Proxylets/Rules Downloading Phase:

Assumptions:

- 1) Trust or relationship is established between the proxylet/rule owners and the OPES box owners before the download can happen.
- 2) The download happens in a separate path than the content path. There is the alternative model in which the proxylets/rules download happens with the content in the data path – however, this model would have different implications to the overall architecture design and hence not assumed here.
- 3) Rules/Proxylets are validated at the OPES Admin Module before taking effect on the content path at the OPES Service Engine.

Steps involved:

Step 1a:

Proxylet Owners or 3rd party vendors send the proxylet and meta-data (for purposes like naming, versioning, identification, authentication, and authorization, etc.) to the OPES Admin Module.

Step 1b:

Rule Module Owners, i.e., service initiators (either Content Providers, Clients, or Access Providers) express their desire to run particular service and the associated condition in the form of Rule Module XML file, and send the XML file to OPES Admin Module possibly owned by another domain.

Step 2a:

Inside the OPES Admin Module, the proxylet is validated against the sandbox, any local policy constraint set up by Admin Module, etc., before it can be registered as legal resident in the proxylet database (could be just a table) and be loaded into the OPES service engine.

Step 2b:

Inside the OPES Admin Module, the Rule Parser receives the Rule Module XML file, parse the file, validate that the associated action item with each rule (either proxylet or call to proxylet library or ICAP) does exist and is supported either as a legal local resident, or remote service. It also checks for any conflict with the rest of the other rule modules, and if conflicts are detected, either resolve or simply report back to the module owner. Upon successful completion of this step, the rule module is accepted into the rule base in the OPES service engine. The rule base is a more efficient binary representation of the rules in the engine.

Issues:

- 1) We need to standardize the representation of Rule Module and Proxylet. PSRL is the attempt to specify XML rule language. We need to do similar thing for

Proxylet – mostly, define the meta-data for naming, versioning, authorization, authentication, multiple language support, etc.

- 2) Do we need to standardize the protocol for step 1 (between proxylets/rules owner and OPES Admin Modules)? One can argue that since some trust relationship has to be in place before this communication happens, and it does not happen within the content path, email or ftp would do the job just fine.
- 3) Do we need to standardize the protocol or API for step 2 (between Admin Module and Service Engine)? One can argue that this protocol/interface could be vendor proprietary and hence no need to standardize. But do we like to see interoperability here between vendors? QUESTION: What is OPES about if there is no standard way to interoperate between boxes? Would be down to a rule/proxylet definition format plus control policies? I'd say that's an area we want to have open either through protocols or APIs.
- 4) How exactly do we resolve the conflict in the Rule Base?
- 5) How exactly do we define the “sandbox” around proxylet so that we can practically validate proxylet?

OPES Service Execution Phase (for HTTP request/response):

Assumptions:

- 1) No assumption is made about the OPES “proxy” functionality like a cache proxy being the foundation of the OPES Service Engine. In another word, OPES Service Engine does not rely upon a caching mechanism to be functional. A cache is considered simply one of many services offered by an OPES box. This flexible design would allow us to build OPES Service Engine on top of a variety of different intermediary devices.
- 2) Currently only HTTP is supported by OPES. However, we do consider it important to have a flexible architecture to accommodate other protocols. To support a new protocol, at the very least, the Message Parser and the Rule Language Spec need to support the headers of the new protocol.

Steps involved:

Step 3:

Requests or responses are presented to the Message Parser to parse the message. This could happen at any of the four processing points (1, 2, 3 and 4) as defined by the OPES framework and PSRL rule language spec.

Step 4:

Message Parser presents its output in the form of Property Set to the OPES Rule Engine. This Property Set includes all the header information and additional information supported by PSRL rule spec.

Step 5:

The Rule Engine is actually a rule-matching iteration loop – it checks against each rule in the Rule Base once and only once, and hence infinite looping is avoided. Once all the

rules have been examined, the Rule Engine is exited and the control is returned to the Message Parser to continue on normal message flow.

Step 6:

For each rule in the rule base, the Property Set is checked according to the rule-matching pattern. If a rule is not matched, the loop continues to the next rule in the rule base. If a rule is matched, it advances to the action-binding step.

Step 7:

Once a rule is matched, the action binding step takes place to bind the action to the appropriate code (a proxylet, a call to the proxylet library, or a ICAP service).

Step 8:

The actual action code is executed. In the case of remote callout service like ICAP, ICAP client would start communicating with the remote ICAP server to carry out the service.

Step 9:

The action code itself might modify the headers or add new headers, and hence the Property Set (headers, etc.) could be modified once the action is done.

Step 10:

The modified Property Set is presented back to the rule-matching loop for next rule.

Issues:

- 1) The Rule Engine loop is executed against the “modified” property set instead of the original property set. This could have some performance and implementation complexity issue. Do we really see a need for this? Can’t we write the rule modules such that we can achieve the same results with loops on original property set instead?
- 2) It might be desirable to define a standard API between the Message Parser and the Rule Engine (step 4) so that OPES Service Engine can be easily implemented over a variety of different intermediary devices.
- 3) A common library API is needed to support some important functionalities like accounting (hit count), etc.
- 4) We need to carefully consider the “sandbox” (or constraint) of the proxylet and find a feasible way to reinforce that on the OPES box to protect the system. For example, a timeout might be needed to limit the CPU time a proxylet can take. Questions like what kind of local and network resource a proxylet can use needs to be carefully answered.