Development Plans

L. Xia, J. Strassner, C. Basile, D. Lopez

I2NSF Meeting, Bangkok, Thailand
November 7th, 2018
Introduction: the Context

- NSF are defined by Capabilities
  - The set of features to be exposed to other I2NSF components and NSFs, independent of the customer and provider interfaces
  - NSFs can be combined to provide security services
  - Every NSF SHOULD be described with the set of capabilities it offers.
  - Capabilities MAY have their access control restricted by policy (this is out of scope for this draft)

- This draft defines
  - The concept of NSF Capabilities and their use using an info model and a Capability Algebra
    - Ensures that the different actions of the Policy Rule do not conflict with each other
Conceptually, a Template of Templates

- Events, Conditions, and Actions are each Templates
  - Define a structure and organization of MTI attributes (and optionally, methods) that define behavior
  - Each may have metadata to further describe properties and operation and/or prescribe behavior

- Policy Rule is a Template of Templates
  - Defines a structure and organization of MTI components of a policy rule
  - Each may have metadata to further describe properties and operation and/or prescribe behavior

- Information Model used to describe the structure and semantics of these templates in a technology-neutral way
Key Abstractions

- Security is independent of physical vs. virtual packaging
- Security is described by one or more Capabilities
- Policies define how to manage Capabilities
- Policies are defined in an object-oriented info model

This enables
- NSF behavior to be defined using Capabilities
- Policy Rules to be defined to manage NSF behavior
- Capabilities and Policy Rules can be reused as is, or extended
The Current Model Uses ECA Policy Rules

- **Events**: significant occurrences the NSF is able to react to
- **Conditions**: how the NSF decides which actions to apply
- **Actions**: what operations to execute
- **PolicyRule**: a container that aggregates an Event, a Condition, and an Action (Boolean) clause

Behavior

- Actions MAY execute if Event and Condition (Boolean) clauses BOTH evaluate to TRUE; this is controlled by resolution strategy and metadata
  - Capability Algebra used to make resolution strategy decidable
- Default actions MAY be specified
Exemplary External Info Model (MCM)

Types of PolicyRules

Objects IN A PolicyRule

Decorator Pattern

Types of Decorated Objects

Clauses in a PolicyRule
Let’s review YANG construction guidelines

- Three key information modeling concepts that a data model SHOULD consistently represent: classes, class inheritance, and associations.
- Each class in the model is represented by a YANG identity and by a YANG grouping. The grouping enables us to define classes abstractly. Each grouping begins with two leaves (either defined in the grouping or inherited via a uses clause), which provide common functionality.
  - One leaf is used for the system-wide unique identifier for this instance
  - The second leaf is an identityref which is set to the identity of the instance. It is read-write in the YANG formalism due to restrictions on the use of MUST clauses.
- Subclassing is done by defining an identity and a grouping for the new class. The identity is based on the parent identity, and is given a new name to represent this class. The new grouping uses the parent grouping. It refines the entity-class of the parent (the second leaf), replacing the default value of the entity-class with the correct value for this class.
Associations are represented by the use of instance-identifiers and association classes. Association classes are classes, using the above construction, which contain leaves representing the set of instance-identifiers for each end of the association, along with any other properties the information model assigns to the association.

The two associated classes each have a leaf with an instance-identifier that points to the association class instance.

Each instance-identifier leaf is defined with a must clause. That must clause references the entity-class of the target of the instance-identifier, and specifies that the entity class type must be the same as, or subclassed from, a specific named class. Thus, associations can point to any instance of a selected class, or any instance of any subclass of that target.

Note: It is impossible in YANG to retain the difference between associations, aggregations, and compositions. This is mitigated by the use of association classes.
The concrete class tree is constructed as follows. The YANG model defines a container for each class that is defined as concrete by the information model. That container contains a single list, keyed by an appropriate instance-identifier. The content of the list is defined by a uses clause referencing the grouping that defines the class.

Example on next slide:
module: ietf-supapolicy
  +--rw supa-encoding-clause-container
    |   +--rw supa-encoding-clause-list* [supa-policy-ID]
    |   |     +--rw entity-class? identityref
    |   |     +--rw supa-policy-ID string
    |   |     +--rw supa-policy-name? string
    |   |     +--rw supa-policy-object-description? string
    |   |     +--rw supa-has-policy-metadata-agg-ptr* instance-identifier
    |   |     +--rw supa-policy-clause-deploy-status identityref
    |   |     +--rw supa-has-policy-clause-part-ptr* instance-identifier
    |   |     +--rw supa-policy-clause-has-decorator-agg-ptr* instance-identifier
    |   |     +--rw supa-encoded-clause-content string
    |   |     +--rw supa-encoded-clause-language enumeration
    +--rw supa-policy-variable-container
      |   +--rw supa-policy-variable-list* [supa-policy-ID]
      |   |     +--rw entity-class? identityref
      |   |     +--rw supa-policy-ID string
      |   |     +--rw supa-policy-name? string
      |   |     +--rw supa-policy-object-description? string
      |   |     +--rw supa-has-policy-metadata-agg-ptr* instance-identifier
      |   |     +--rw supa-policy-clause-has-decorator-part-ptr? instance-identifier
      |   |     +--rw supa-has-decorated-policy-component-part-ptr? instance-identifier
      |   |     +--rw supa-pol-clause-constraint* string
      |   |     +--rw supa-pol-clause-constraint-encoding? identityref
      |   |     +--rw supa-has-decorated-policy-component-agg-ptr* instance-identifier
      |   |     +--rw supa-pol-comp-constraint* string
      |   |     +--rw supa-pol-comp-constraint-encoding? identityref
      |   |     +--rw supa-policy-term-is-negated? boolean
      |   |     +--rw supa-policy-variable-name? string
Main Updates in -04

- Re-organize the document structure (no more new contents): create a new section 3.4 (Modelling NSF Features as Security Capabilities), and move the existing sections into it, in which:
  - 3.4.1 - Matched Policy Rule, 3.4.2 Conflict, Resolution Strategy and Default Action and 3.4.3 I2NSF Condition Clause Operator Types are logically closely related, to clarify how to construct a Policy Rule and all of the key components
  - 3.4.4 - Uses of the capability information model: clarify the “GNSF” concept
  - 3.4.5 - A Syntax to Describe the Capability of an NSF and 3.4.6 - Capability Algebra are together to describe the representation of NSF Capability and how to manipulate them with a formal way (Capability Algebra)

- Add Section 4 (Considerations on the Practical Use of the CapIM) to describe how our IM serves the purposes of I2NSF WG and allows solving issues that WG wanted to solve: maybe better as an Appendix
**Next Step**

- Further content improvement of section 4 (considering moving it to Appendix), one more round of document text polishing

- Provide examples of the YANG generation rules in Appendix

- Next version (-05) for WGLC?

- Analyze existing DMs in the light of the Capability Model and contribute it as a supporting Internet Draft
  - Not as part of this document to avoid unmanageable forward references
Questions?

“Create like a god. Command like a king. Work like a slave”
- Constantin Brancusi