draft-ietf-i2nsf-capability-02 Development Plans

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

I2NSF Meeting,
Montreal, Canada
July 18th, 2017

Introduction: the Context

- **NSFs are defined by Capabilities**
  - The set of features to be exposed to other 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 draft defines**
  - The concept of NSF Capabilities and their use using an info model and a Capability Algebra
    - The Capability Algebra enables a template approach to be used to describe the Capabilities of an NSF.

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

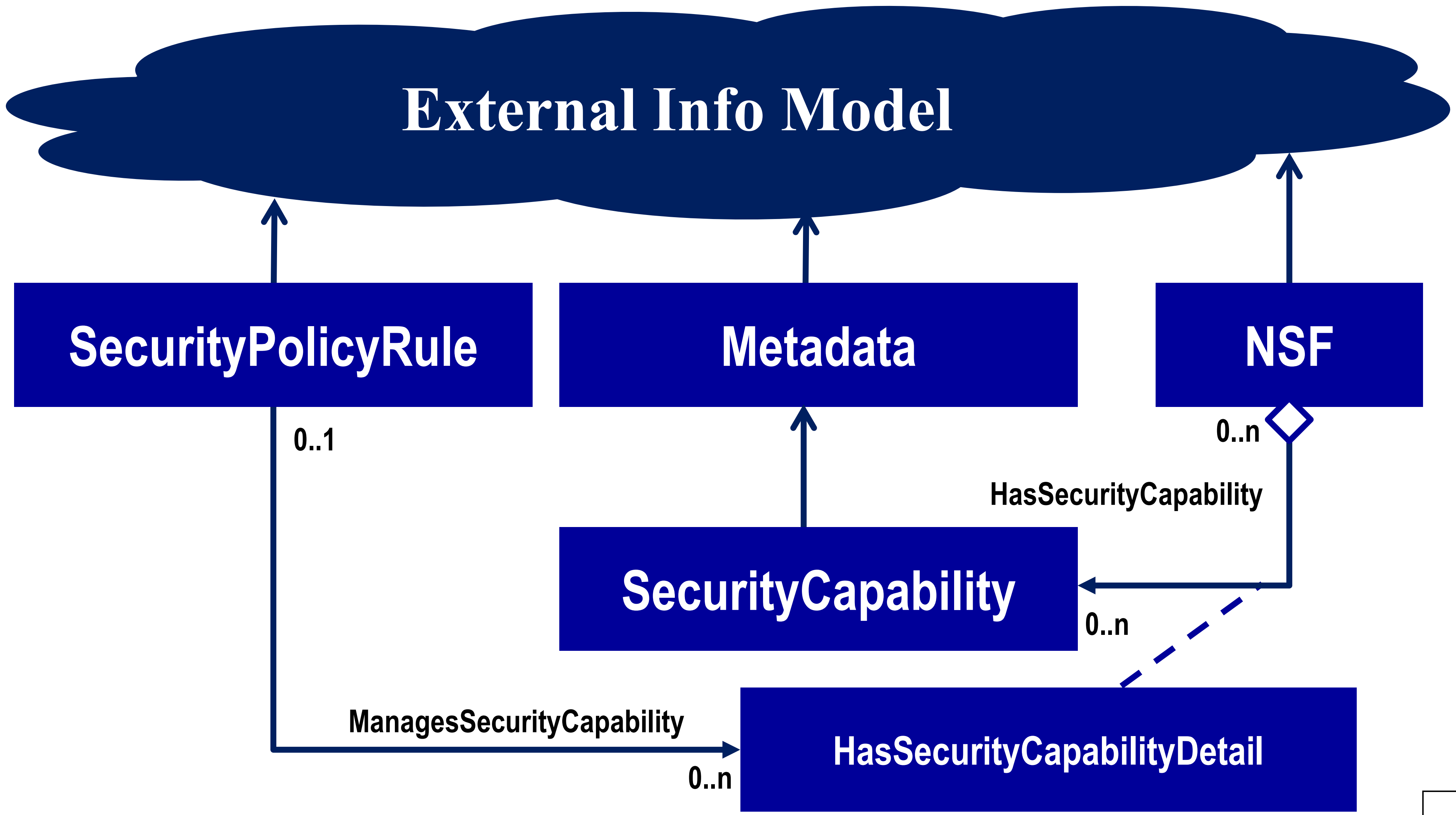
- **Security is independent of physical vs. virtual packaging**
- **Security is described by one or more Capabilities**
  - e.g., this NSF can filter packets (supports Allow and Deny actions) based on IP addresses (supports conditions IP source and destination)
- **Policies define how to manage Capabilities**
  - e.g. write rules like 'if IP source = 1.2.3.4 then Deny'
- **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*

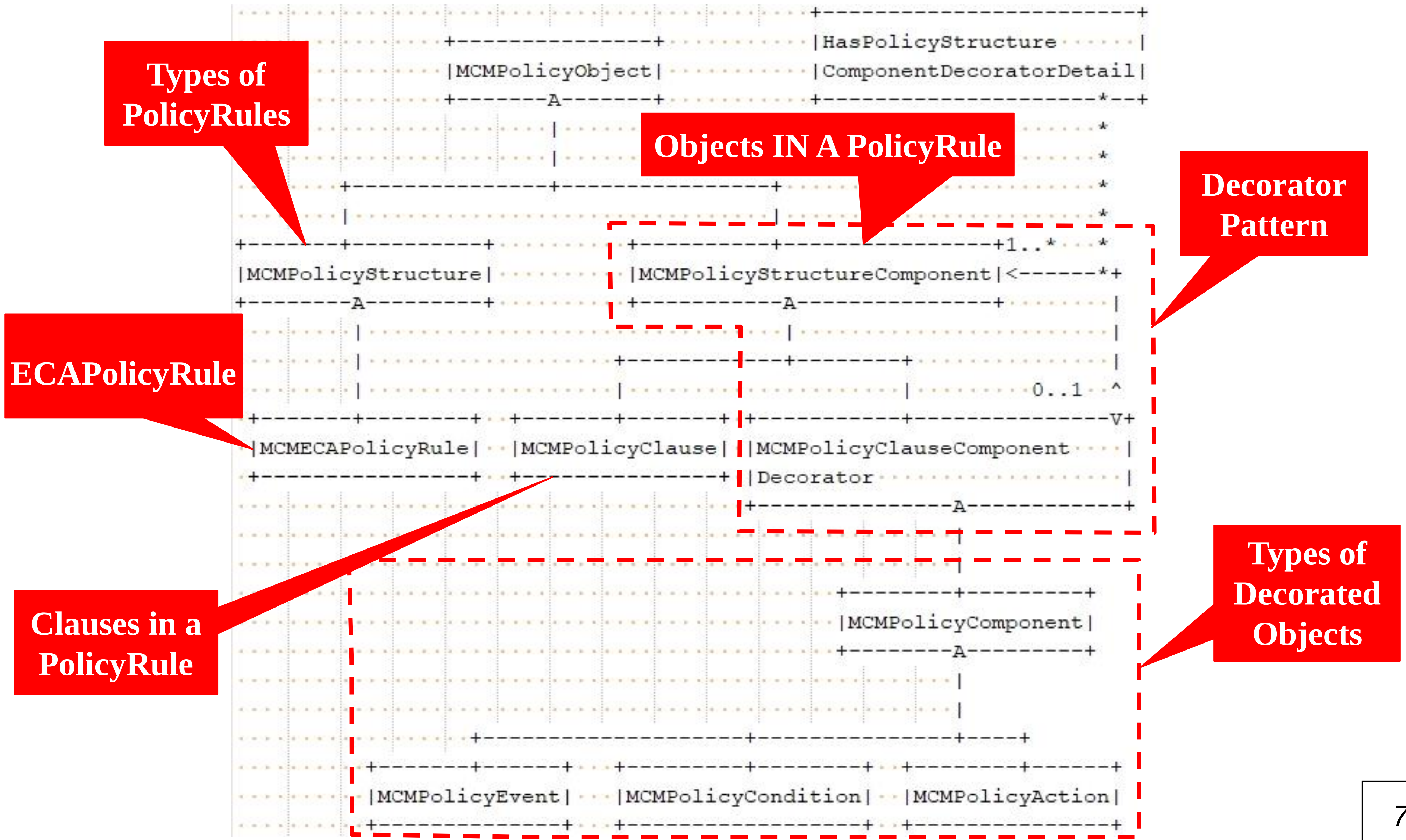
The ECA Policy Rule Model

- **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

Conceptual Operation



Exemplary External Info Model (MCM)



YANG Generation (1)

- **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.

YANG Generation (2)

- 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.

YANG Generation (3)

- 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:

Example YANG

```
module: ietf-supa-policy
   +--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
```

Questions?

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