Network Working Group
Internet Draft
Intended status: Standard Track
Expires: November 09, 2015

J. Strassner
Huawei Technologies

May 09, 2015

Generic Policy Information Model for Simplified Use of Policy Abstractions (SUPA)
draft-strassner-supa-generic-policy-info-model-01

Abstract

The Simplified Use of Policy Abstractions (SUPA) addresses the needs of operators and application developers to represent multiple types of policy rules. This document defines a single common extensible framework for representing different types of policy rules, in the form of a set of information models, that is independent of language, protocol, repository, and the level of abstraction of the content of the policy rule. This enables a common set of concepts defined in this set of information models to be mapped into different data models that use different languages, protocols, and repositories to optimize their usage. The definition of common policy concepts also provides better interoperability by ensuring that each data model can share a set of common concepts, independent of its level of detail or the language, protocol, and/or repository that it is using.

Specifically, this document defines three information models:

1. A framework for defining the concept of policy, independent of how policy is defined or used; this is called the SUPA Generic Policy Information Model (SGPIM)
2. A framework for defining a policy model that uses the event-condition-action paradigm; this is called the SUPA Eca Policy Rule Information Model (EPRIM)
3. A framework for defining a policy model that uses a declarative (e.g., intent-based) paradigm; this is called the SUPA Logic Statement Information Model (SLSIM)

The combination of the SGPIM and the EPRIM, or the SGPIM and the SLSIM, provide an extensible framework for defining policy that uses an event-condition-action or declarative representation that is independent of data repository, data definition language, query language, implementation language, and protocol.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.
Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This Internet-Draft will expire on October 26, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

Table of Contents (continued)

Table of Contents (continued)

Table of Contents (continued)

Table of Contents (continued)

1. Introduction

The Simplified Use Policy Abstractions (SUPA) addresses the needs of operators and application developers to represent multiple types of policy rules using a common structure for defining policy rules that is independent of language, protocol, repository, and the level of abstraction of the content of the policy rule. This common framework currently takes the form of a set of three information models. The SUPA Generic Policy Information Model (SGPIM) defines a common set of policy management concepts that are independent of the type of policy rule, while the SUPA ECA Policy Rule Information Model (EPRIM) and SUPA Logic Statement Information Model (SLSIM) define information models that are specific to the needs of Event-Condition-Action (ECA) policy rules and statements that are subsets of either Propositional Logic (PL) or First-Order Logic (FOL), respectively.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying [RFC2119] significance.

3. Terminology

This section defines acronyms, terms, and symbology used in the rest of this document.

3.1. Acronyms

CLI	Command Line Interface
CNF	Conjunctive Normal Form
DNF	Disjunctive Normal Form
ECA	Event-Condition-Action
EPRIM	ECA Policy Rule Information Model
FOL	First Order Logic
NETCONF	Network Configuration protocol
OAM&P	Operations, Administration, Management, and Provisioning
OID	Object IDentifier

PAP	Policy Administration Point
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
PL	Propositional Logic
PR	Policy Repository
PXP	Policy Execution Point
SGPIM	SUPA Generic Policy Information Model
SLSIM	SUPA Logic Statement Information Model
SUPA	Simplified Use of Policy Abstractions
TMF	TeleManagent Forum (TM Forum)
UML	Unified Modeling Language
URI	Uniform Resource Identifier
YANG	A data definition language for use with NETCONF
ZOOM	Zero-touch Orchestration, Operations, and Management

3.2. Definitions

This section defines the terminology that is used in this document.

3.2.1. Core Terminology

The following subsections define the terms "information model" and "data model".

3.2.1.1. Information Model

An information model is a representation of concepts of interest to an environment in a form that is independent of data repository, data definition language, query language, implementation language, and protocol.

Note: this definition is different than that of [RFC3198]. An information model is defined in [RFC3198] as: "An abstraction and representation of the entities in a managed environment, their properties, attributes, and operations, and the way that they relate to each other. It is independent of any specific repository, software usage, protocol, or platform." The SUPA definition is more specific, and corrects the following ambiguities:

- Most information models do not define operations; this is typically implementation-specific and a function of (at least) the language, protocol, and data repository used.
- It is unclear what the difference is between the terms "properties" and "attributes" (these are typically synonyms in modeling terminology)
- It is unclear what is meant by "software usage".
- It is unclear what is meant by "platform".

3.2.1.2. Data Model

A data model is a representation of concepts of interest to an environment in a form that is dependent on data repository, data definition language, query language, implementation language, and protocol (typically, but not necessarily, all three).

Note: this definition is different than that of [RFC3198]. A data model is defined in [RFC3198] as: "A mapping of the contents of an information model into a form that is specific to a particular type of data store or repository." The SUPA definition is more specific. For example, it takes into account differences between two implementations that use the same protocol, implementation language, and data repository, but which have different data definition and/or query protocols.

3.2.1.3. Container

A container is an object whose instances may contain zero or more additional objects, including container objects. A container provides storage, query, and retrieval of its contained objects in a well-known, organized way.

3.2.1.4. PolicyContainer

In this document, a PolicyContainer is a special type of container that provides at least the following three functions:

1. It uses metadata to define how its content is interpreted
2. It separates the content of the policy from the representation of the policy
3. It provides a convenient control point for OAMP operations

The combination of these three functions enables a PolicyContainer to define the behavior of how its constituent components will be accessed, queried, stored, retrieved, and how they operate.

3.2.2. Policy Terminology

The following terms define different policy concepts used in the SUPA Generic Policy Information Model (SGPIM). Note that the prefix "SUPA" is used for all classes and relationships defined in the SGPIM to ensure name uniqueness. Similarly, the prefix "supa" is defined for all SUPA class attributes.

3.2.2.1. SUPAPolicy

A SUPAPolicy is an abstract class that is a type of PolicyContainer.

SUPAPolicy is defined generically as a means to monitor and control the changing and/or maintaining of the state of one or more managed objects [1]. In this context, "manage" means that at least create, read, query, update, and delete unctions are supported.

A SUPAPolicy MUST have at least one SUPAPolicyStatement that are used to define the content of the policy. A SUPAPolicy MAY be qualified (i.e., may aggregate these objects to more completely specify the behavior of the SUPAPolicy) by a set of zero or more SUPAPolicySubjects, SUPAPolicyTargets, and/or SUPAPolicyMetadata objects. Note that these three classes are defined as abstract, in order to simplify mapping to, and optimization of, data models. When defined in an information model, the SUPAPolicy class MUST have separate aggregation relationships with the SUPAPolicySubject, SUPAPolicyTarget, and SUPAPolicyMetadata classes. When implemented in a data model, the set of SUPAPolicyStatement, SUPAPolicyTarget, SUPAPolicySubject, and SUPAPolicyMetadata object instances, SHOULD all be part of a single PolicyContainer object.

3.2.2.2. SUPAPolicyStatement

A SUPAPolicyStatement is an abstract class that contains an individual or group of related functions; this set of functions defines a set of actions to take. Examples of actions include getting information, stating facts about the system being managed, writing a change to a configuration of one or more managed objects, and querying information about one or more managed objects.

SUPAPolicyStatements are objects in their own right, which facilitates their reuse. SUPAPolicyStatements can also be combined in a whole-part (containment) relationship under a SUPAPolicy, thereby forming a SUPAECAPolicyRule or a SUPALogicStatement. When defined in an information model, a SUPAPolicyStatement MUST be represented as a separate object that aggregates its constituent components. However, a data model MAY map this definition to a more efficient form (e.g., flattening the SUPAPolicyStatement and its aggregated object instances into a single object instance).

3.2.2.3. SUPAECAPolicyRule

An Event-Condition-Action (ECA) Policy (SUPAECAPolicyRule) is an abstract class that MUST contain at least one SUPAPolicyStatement. Optionally, it MAY contain one or more SUPAPolicySubjects, one or more SUPAPolicyTargets, and one or more SUPAPolicyMetadata objects.

The SUPAPolicyStatement defines the content of the Policy Rule as a three-tuple, consisting of an event clause, a condition clause, and an action clause. Each of these three clauses MUST have at least one term corresponding to the type of clause that it is; it MAY have more than one.

These three terms collectively specify what triggers the evaluation of the SUPAECAPolicyRule, whether all of the conditions specified have been satisfied or not, and if the conditions are satisfied, the set of actions to be executed. This differentiates a SUPAECAPolicyRule from a SUPALogicStatement, which specifies what actions to perform, but not how to perform them.

If there are more than one term, then these terms MUST be combined using any combination of logical AND, OR, and NOT operators to form a Boolean clause (i.e., a clause whose value is either TRUE or FALSE). For example, a valid event clause could be: "three events of type A AND NOT an event of type B".

These three clauses enable the semantics of a SUPAECAPolicyRule to be clearly differentiated from the semantics of other types of SUPAPolicies that use SUPAPolicyStatements (and other parts of the SPGIM), such as SUPALogicStatements.

The semantics of a SUPAECAPolicyRule are defined as follows:

- The event clause defines a Boolean statement that, if true, MUST trigger the evaluation of the condition clause of the SUPAECAPolicyRule.
- The condition clause defines a Boolean statement that, if true, MUST start the execution of the actions of the SUPAECAPolicyRule.
- The action clause is an aggregation of actions that MUST be executed if the event and condition clauses so dictate.
- A SUPAECAPolicyRule MAY specify a set of SUPAPolicySubjects that have authored the SUPAECAPolicyRule.
- A SUPAECAPolicyRule MAY specify a set of SUPAPolicyTargets that define a set of managed objects that the actions of the SUPAECAPolicyRule MAY monitor and/or change their state.
- The behavior of the event, condition, and action clauses MAY be specified using one or more SUPAMetadata objects that have

been aggregated by the SUPAECAPolicyRule.

When defined in an information model, each of the event, condition, and action clauses MUST be represented as an aggregation between a SUPAECAPolicyRule (the aggregate) and a set of event, condition, or action objects (the components). However, a data model MAY map these definitions to a more efficient form (e.g., by flattening these three types of object instances, along with their respective aggregations, into a single object instance).

3.2.2.4. SUPALogicStatement

A SUPALogicStatement is an abstract class that MUST contain at least one SUPAPolicyStatement. A SUPALogicStatement defines what actions to take, but not how to execute those actions. This differentiates it from a SUPAECAPolicyRule, which explicitly defines what triggers the evaluation of the SUPAECAPolicyRule, what conditions must be satisfied in order to execute the actions of the SUPAECAPolicyRule, and what actions to execute. A SUPALogicStatement is commonly called declarative, or intent-based, policy.

This document defines two forms of a SUPALogicStatements. The first uses Propositional Logic (PL, see Section 3.2.4.2), while the second uses First-Order Logic (FOL, see Section 3.2.4.3).

Note that this document does not refer to a SUPALogicStatement as a "rule", since both types of SUPALogicStatements defined in this document are technically not "rules". Rather, they are types of zero-order and first-order logic statements.

If the SUPALogicStatement is expressed in PL, then it MUST consist of only the propositional connectives (i.e., negation, conjunction, disjunction, implication, and bi-implication (see Section 3.2.4.1). Furthermore, statements in a PL are limited to simple declarative propositions that MUST NOT use quantified variable or predicates.

If the SUPALogicStatement is expressed in FOL, then it MUST consist of a set of logical predicates (i.e., a Boolean-valued function). The predicate can use all propositional connectives as well as two additional quantifiers (i.e., the universal quantifier and the existential quantifier).

A logical predicate MUST consist of a head clause, and MAY also contain a body clause. This enables the semantics of a SUPALogicStatement to be clearly differentiated from the semantics of other types of SUPAPolicies that use SUPAPolicyStatements (and other parts of the SPGIM), such as SUPAECAPolicyRules. While in principle higher order logics can be defined, this document is limited to defining a SUPALogicStatement using either PL or FOL.

When implemented in an information model, each PL or FOL statement MUST be defined as objects (i.e., a subclass of the SUPALogicStatement class; see Section 7). When an FOL statement is implemented in an information model, both the head and body clauses MUST be defined as objects (or sets of objects). However, a data model MAY map either a PL statement or an FOL statement to a more efficient form (e.g., by flattening the head and body objects into a single object).

3.2.2.5. SUPAMetadata

Metadata is, literally, data about data. SUPAMetadata is an abstract class that contains prescriptive and/or descriptive information about the object(s) that it is attached to. While metadata can be attached to any information model element, this document only considers metadata attached to classes and relationships.

When defined in an information model, each instance of the SUPAMetadata class MUST have its own aggregation relationship with the set of objects that it applies to. However, a data model MAY map these definitions to a more efficient form (e.g., flattening the object instances into a single object instance).

3.2.2.6. SUPAPolicyTarget

SUPAPolicyTarget is an abstract class that defines a set of managed objects that may be affected by the actions of a SUPAPolicyStatement. A SUPAPolicyTarget may use one or more mechanisms to identify the set of managed objects that it affects; examples include OIDs and URIs.

When defined in an information model, each instance of the SUPAPolicyTarget class MUST have its own aggregation relationship with each SUPAPolicy that uses it. However, a data model MAY map these definitions to a more efficient form (e.g., flattening the SUPAPolicyTarget, SUPAMetadata, and SUPAPolicy object instances into a single object instance).

3.2.2.7. SUPAPolicySubject

SUPAPolicySubject is an abstract class that defines a set of managed objects that authored this SUPAPolicyStatement. This is required for auditability. A SUPAPolicySubject may use one or more mechanisms to identify the set of managed objects that authored it; examples include OIDs and URIs.

When defined in an information model, each instance of the SUPAPolicySubject class MUST have its own aggregation relationship with each Policy that uses it. However, a data model MAY map these definitions to a more efficient form (e.g., flattening the PolicySubject, Metadata, and Policy object instances into a single object instance).

3.2.3. Modeling Terminology

The following terms define different types of relationships used in the information models of the SUPA Generic Policy Information Model (SGPIM).

3.2.3.1. Inheritance

Inheritance makes an entity at a lower level of abstraction (e.g., the subclass) a type of an entity at a higher level of abstraction (e.g., the superclass). A subclass does NOT change the characteristics or behavior of the superclass that it inherits from. However, a subclass MAY add new attributes and relationships that distinguish it from the attributes and relationships defined by its superclass.

3.2.3.2. Relationship

A relationship is a generic term that represents how a first set of entities interact with a second set of entities. A recursive relationship sets the first and second entity to the same entity. There are three basic types of relationships, as defined in the subsections below: associations, aggregations, and compositions.

3.2.3.3. Association

An association represents a generic dependency between a first and a second set of entities.

3.2.3.4. Aggregation

An aggregation is a stronger type (i.e., more restricted semantically) of association, and represents a whole-part dependency between a first and a second set of entities. Three objects are defined by an aggregation: the first entity, the second entity, and a new third entity that represents the combination of the first and second entities. The entity owning the aggregation is referred to as the "aggregate", and the entity that is aggregated is referred to as the "part".

3.2.3.5. Composition

A composition is a stronger type (i.e., more restricted semantically) of aggregation, and represents a whole-part dependency with two important behaviors. First, an instance of the part is included in at most one instance of the aggregate at a time. Second, any action performed on the composite entity (i.e., the aggregate) is propagated to its constituent part objects. For example, if the composite entity is deleted, then all of its constituent part entities are also deleted. This is not true of aggregations or associations - in both, only the entity being deleted is actually removed, and the other entities are unaffected.

3.2.3.6. Association Class

A relationship may be implemented as an association class. This is used to define the relationship as having its own set of features.

More specifically, if the relationship is implemented as an association class, then the attributes of the association class, as well as other relationships that the association class participates in, may be used to define the semantics of the relationship.

If the relationship is not implemented as an association class, then no additional semantics (beyond those defined by the type of the relationship) are expressed by the relationship.

3.2.3.7. Multiplicity

A specification of the range of allowable cardinalities that a set of entities may assume. This is always a pair of ranges, such as 1 - 1 or 0..n - 2..5.

3.2.3.8. Navigability

A relationship may have a restriction on the ability of an object at one end of the relationship to access the object at the other end of the relationship. In this document, two choices are possible:

1. Each object is navigable by the other, which is indicated by NOT providing any additional symbology, or
2. An object A can navigate to object B, but object B cannot navigate to object A. This is indicated by an open-headed arrow pointing to the object that cannot navigate to the other object. In this example, the arrow would be pointing at object B.

3.2.3.9. Abstract Class

An abstract class is a class that cannot be directly instantiated.

3.2.3.10. Concrete Class

A concrete class is a class that can be directly instantiated.

3.2.4. Mathematical Logic Terminology

This section defines terminology for mathematical logic.

3.2.4.1. Predicate

A predicate is a Boolean-valued function (i.e., a function whose values are interpreted as either TRUE or FALSE, depending on the values of its variables).

3.2.4.2. Logic Operators

A logical connective is a symbol or word that defines how to connect two or more sentences in a language.

3.2.4.2.1. Propositional Logic Connectives

There are five propositional logic connectives, defined as follows:

- Negation, or a logical NOT operator, is an operation that, when applied to a proposition, produces a new proposition "not p", which has the opposite truth value of p.
- Conjunction, or a logical AND operator, is an operation on two logical values that produces a value of TRUE if and only if both of its operands are TRUE.
- Disjunction, or a logical OR operator, is an operation on two logical values that produces a value of FALSE if and only if both of its operands are FALSE.
- Implication, or the conditional operator, is used to form statements of the form "if <proposition A> is TRUE, then <proposition B> is also TRUE (i.e., this statement is FALSE only when A is TRUE and B is FALSE).
- Bi-implication, or the bi-conditional operator, is used to form statements of the form "<proposition A> is TRUE if and only if <proposition B> is TRUE (i.e., this statement is TRUE if and only if both propositions are FALSE or if both propositions are TRUE).

3.2.4.2.2. First Order Logic Quantifiers

Quantification specifies the number of objects that satisfies a formula. This document uses two such quantifiers, which are defined as follows:

- Universal quantification asserts that a predicate within the scope of this operator is TRUE of every value of a variable of the predicate. It is commonly interpreted as "for all".
- Existential quantification asserts that a predicate within the scope of this operator is TRUE for at least one value of a variable of the predicate. It is commonly interpreted as "there exists, "there is at least one", or "for some".

3.2.4.3. Propositional Logic

Propositional Logic (PL) may be simply defined as a language consisting of a set of statements; the value of each statement is either TRUE or FALSE. More formally, a (propositional) Argument consists of a sequence of Premises and a Conclusion. An Argument is valid if the Conclusion is TRUE whenever all Premises are TRUE.

PL may be thought of as a set of declarative propositions.

3.2.4.4. First-Order Logic

First-Order Logic (FOL) may be simply defined as a language consisting of a set of statements; each statement is a predicate.

A predicate is a Boolean-valued function (i.e., the value of the function evaluates to either TRUE or FALSE, depending on the value of its variables). Predicates can also be compared.

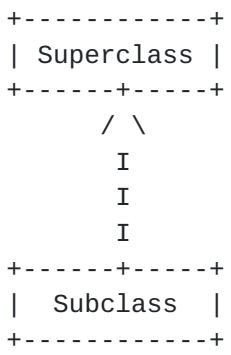
FOL uses quantified variables. The universal quantifier and/or the existential quantifier can be used to define what values can be instantiated by the predicated variables.

3.3. Symbology

The following symbology is used in this document:

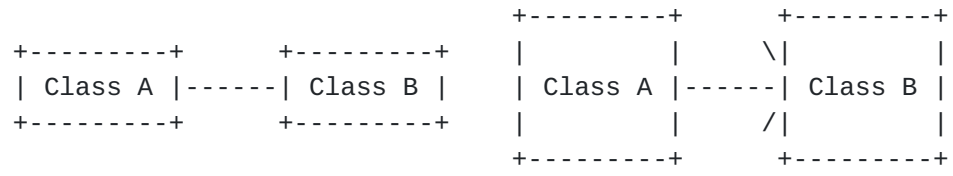
3.3.1. Inheritance

Inheritance: a subclass inherits the attributes and relationships of its superclass, as shown below:



3.3.2. Association

Association: Class B depends on Class A, as shown below:

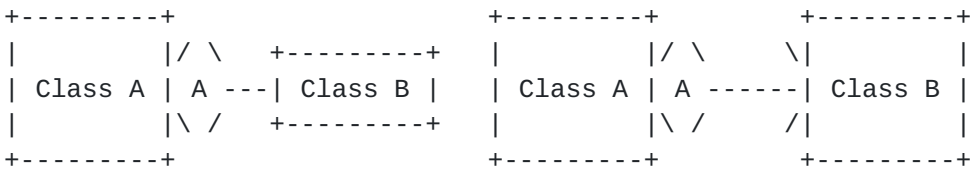


association with no navigability restrictions

association with navigability restrictions

3.3.3. Aggregation

Aggregation: Class B is the part, Class A is the aggregate, as shown below:

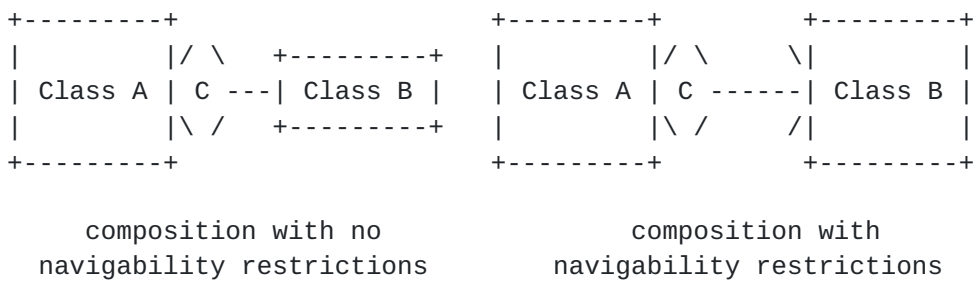


aggregation with no navigability restrictions

aggregation with navigability restrictions

3.3.4. Composition

Composition: Class B is the part, Class A is the composite, as shown below:

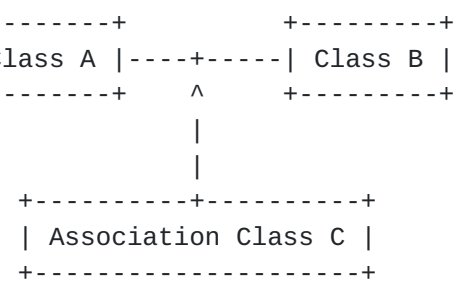


composition with no navigability restrictions

composition with navigability restrictions

3.3.5. Association Class

Association Class: Class C is the association class implementing the relationship D between classes A and B



3.3.6. Logical Connectives

The following defines a mapping between the typical mathematical symbols used for logical connectives (most of which are in extended ASCII) and the symbols that will be used in this document.

Connective	ASCII CODE	UNICODE Code	Meaning
Negation	172	U+00AC	"NOT"
Conjunction	8743	U+2227	"AND"
Disjunction	8744	U+2228	"OR"
Implication	8658	U+21D2	"IMPLIES"
Bi-implication	8660	U+21D4	"IF AND ONLY IF"

3.3.7. Quantifiers

The following defines a mapping between the typical mathematical symbols used for quantifiers and the symbols that will be used in this document.

Quantifier	ASCII Code	Unicode Code	Symbol Used
Universal	8704	U+2200	"FOR ALL"
Existential	8707	U+2203	"THERE EXISTS"

4. Policy Abstraction Architecture

This section describes the motivation for the policy abstractions that are used in SUPA. In summary, the following abstractions are provided:

- The SGPIM defines a technology-neutral information model that can express the concept of Policy.
- This version of this document restricts the expression of Policy to either an event-condition-action tuple, a FOL predicate, or a combination of these statements.
- Since these two representations are very different in syntax and content, the content of a Policy is abstracted from its representation:
  - Both SUPAECAPolicyRules and SUPALogicStatements are types of SUPAPolicies
  - Both SUPAECAPolicyRules and SUPALogicStatements are constructed from SUPAPolicyStatements
  - The syntax of a SUPAECAPolicyRule, and hence its representation, is different from that of a SUPALogicStatement
  - A SUPAPolicy MAY use SUPAECAPolicyRules and/or SUPALogicStatements
  - A SUPAPolicy consists of one or more SUPAPolicyStatements, and optionally may specify one or more SUPAPolicyTarget, SUPAPolicySubject, and SUPAPolicyMetadata objects
- A SUPAPolicy MUST contain at least one SUPAPolicyStatement; it MAY contain more than one.
- A SUPAECAPolicyRule defines the set of events and conditions that are responsible for executing its actions; it MUST have an event clause, a condition clause, and an action clause.
- A SUPALogicStatement expresses facts that it believes to be true without defining how those facts are computed, and provides an efficient query mechanism for retrieving facts. Each SUPAPolicyStatement MUST be expressed as a function-free Horn clause; there are a number of additional restrictions that are covered in Section 7.
- SUPAMetadata MAY be defined for any type of SUPAPolicyStatement (as well as for individual objects that make up a SUPAPolicyStatement).
- SUPAMetadata MAY be prescriptive and/or descriptive in nature.
- A SUPAPolicyTarget is a set of managed objects that the actions of the SUPAPolicy are applied to.
- A SUPAPolicySubject is a set of managed objects that authored the SUPAPolicy.

4.1. Motivation

The power of policy management is its applicability to many different types of systems. There are many different actors that can use a policy management system, including end-users, operators, application developers, and administrators. Each of these constituencies have different concepts and skills, and use different terminology. For example, an operator may want to express an operational rule that states that only Platinum and Gold users can use streaming multimedia applications. As a second example, a network administrator may want to define a more concrete policy rule that looks at the number of dropped packets and, if that number exceeds a programmable threshold, changes the queuing and dropping algorithms used.

Both of these examples are commonly referred to as "policy rules", but they take very different forms, since they are at very different levels of abstraction and likely authored by different actors. The first was very abstract, and did not contain any technology-specific terms, while the second was more concrete, and likely used technical terms of a general (e.g., IP address range, port numbers) as well as a vendor-specific nature (e.g., specific algorithms implemented in a particular device).

Note that these two policy rules could affect each other. For example, Gold and Platinum users might need different device configurations to give the proper QoS markings to their streaming multimedia traffic. This is very difficult to do if a common policy model does not exist.

More importantly, the users of these two policies likely have different job responsibilities. They may have no idea of the concepts used in each policy. Yet, their policies need to interact in order for the business to provide the desired service. Hence, the need for a common policy framework.

4.2. SUPA Approach

The purpose of the SUPA Generic Policy Information Model (SGPIM) is to define a common framework for expressing policies at different levels of abstraction. SUPA uses the SGPIM as a common vocabulary for representing concepts that are common to expressing policy, but which are independent of language, protocol, repository, and level of abstraction. This enables different policies at different levels of abstraction to form a continuum, where more abstract policies can be translated into more concrete policies, and vice-versa.

It may be necessary to translate the form of a PolicyRule from a general to a more specific form (while keeping the abstraction level the same). For example, the declarative policy "Every network attached to a VM must be a private network owned by someone in the same group as the owner of the VM" may be translated to more formal form (e.g., Datalog, or the Congress version of Datalog). It may also be necessary to translate a Policy to a different level of abstraction. For example, the previous Policy may need to be translated to a form that network devices understand. A common framework for expressing policies that is independent of the level of abstraction is required in order to form such a continuum.

4.3. SUPA Generic Policy Information Model Overview

Figure 1 illustrates the approach for representing policy rules in SUPA. The top two layers are defined in this document; the bottom layer (Data Models) are defined in separate documents.

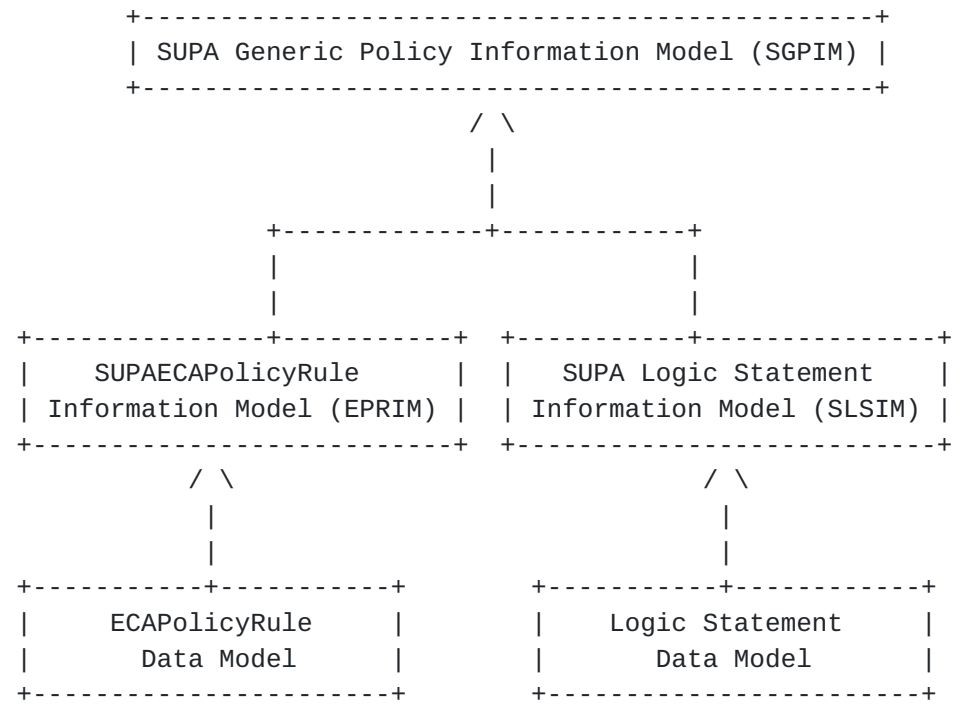


Figure 1: Overview of SUPA Policy Rule Abstractions

Conceptually, the SGPIM defines a set of objects that define the key elements of a Policy independent of how it is represented or its content. As will be shown, there is a significant difference between SUPAECAPolicyRules (see Section 6) and SUPALogicStatements (see Section 7). In principle, other types of SUPAPolicies could be defined, but the current charter is restricted to using these

two types of SUPAPolicies as exemplars.

The SGPIM defines the following concepts:

- o SUPAPolicy: the root of the SPGIM model
- o SUPAPolicyAtomic: a Policy that can be used in a stand-alone manner
- o SUPAPolicyComposite: used to build hierarchies of Policies
- o SUPAPolicyStatement: used to define the content of a SUPAPolicy
- o SUPAPolicyTerm: used to define variables, operators, and values in a SUPAPolicyStatement
- o SUPAPolicySubject: the author of a SUPAPolicy
- o SUPAPolicyTarget: the managed object that a SUPAPolicy monitors and/or controls the state of
- o SUPAPolicyMetadata: specifies descriptive and/or prescriptive information about a SUPAPolicy object

A SUPAPolicy object serves as a single root of the SUPA system (i.e., all other classes in the model are subclasses of the SUPAPolicy class). This simplifes code generation and reusability. Note that this is NOT true of either [4] or [6].

SUPA Policies are defined as either a stand-alone or a hierarchy of PolicyContainers. A PolicyContainer specifies the structure, content, and optionally, subject, target, and metadata information for the Policy.

A SUPAPolicy takes one of two forms: (1) an ECA Policy, and/or (2) a declarative set of statements. Note that unlike other approaches (except [2] and [5], these two different types of Policies may be combined.

Both a SUPAECAPolicyRule and a SUPALogicalStatement are made up of one or more SUPAPolicyStatements, which define the content of the Policy. Three types of SUPAPolicyStatements are available; one is generic, and can be used by any type of Policy, while the other two are specific to an ECA or a declarative Policy, respectively.

A SUPAPolicyStatement may be made up of SUPAPolicyTerms. In addition, specific objects for constructing ECA Policies and declarative Policies are also provided.

This set of classes enables each different types of Policies to be defined by an information model that refines the generic concepts of the SGPIM as described above. For example, a SUPAECAPolicyRule, as well as a SUPALogicStatement, are both subclasses of the SUPAPolicyAtomic class. Therefore, both can be used as part of a hierarchy of Policies or in a stand-alone manner. As another examples, a SUPALogicClause and a SUPABooleanClause are both subclasses of SUPAPolicyStatement, and are used to create

SUPALogicStatements and SUPAECAPolicyRules, respectively.

4.4. Structure of SUPA Policies

This section describes the overall design of the SGPIM.

4.4.1. ECA Policy Rule Structure

A SUPAECAPolicyRule is a statement that consists of an event clause, a condition clause, and an action clause. This type of Policy explicitly defines the current and desired states of the system being managed. It may be represented as follows:

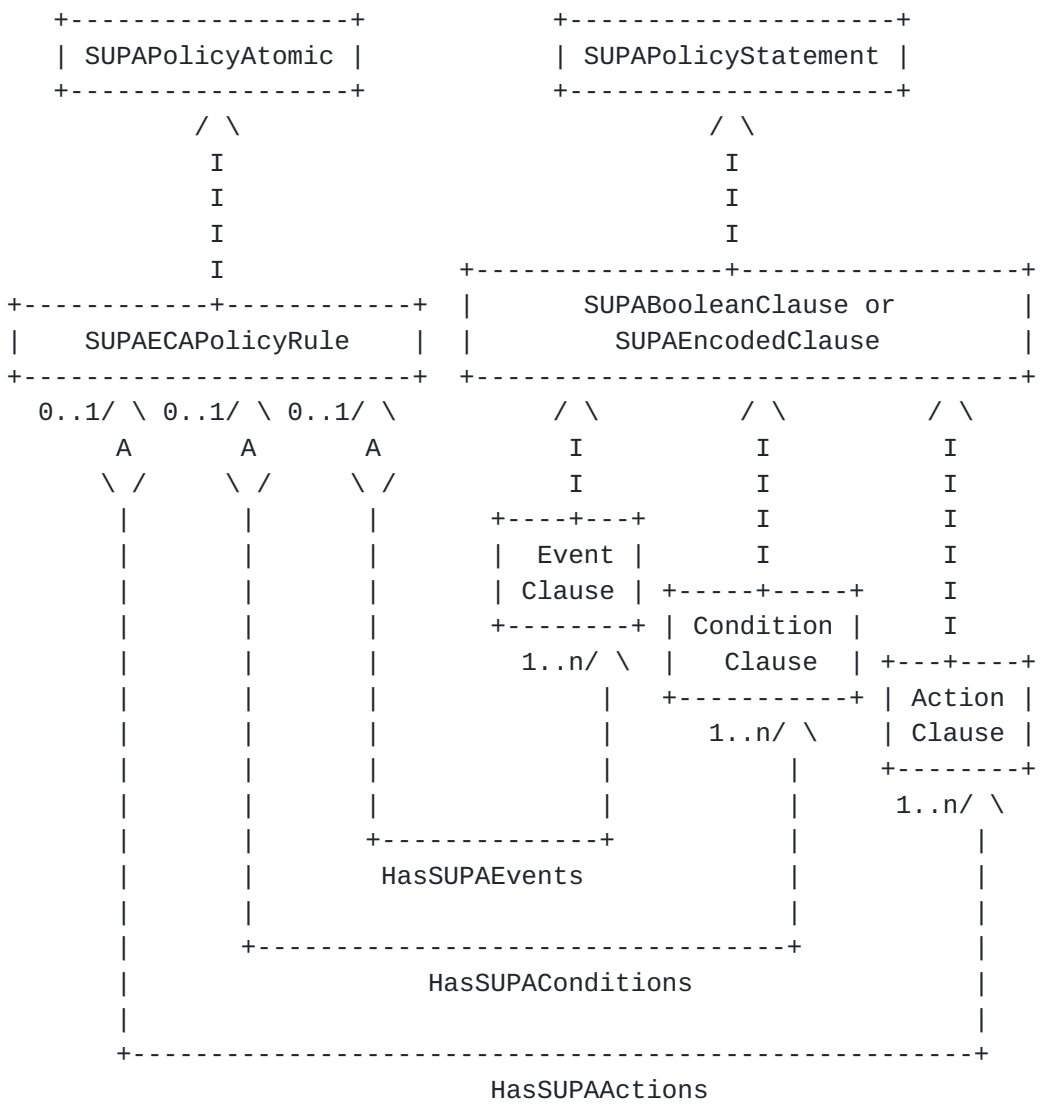


Figure 2: Overview of SUPA Policy Rule Abstractions

4.4.2. Logical Statement Structure

A SUPALogicStatement is either a set of PL or FOL statements.

A SUPAPLStatement is a set of propositions that form a (single) conclusion. A proposition is either TRUE or FALSE. A proposition be created from simpler propositions combined using Propositional Logic Connectives (see Section Propositions (see Section 3.2.4.2.1.). It may be conceptualized as follows:

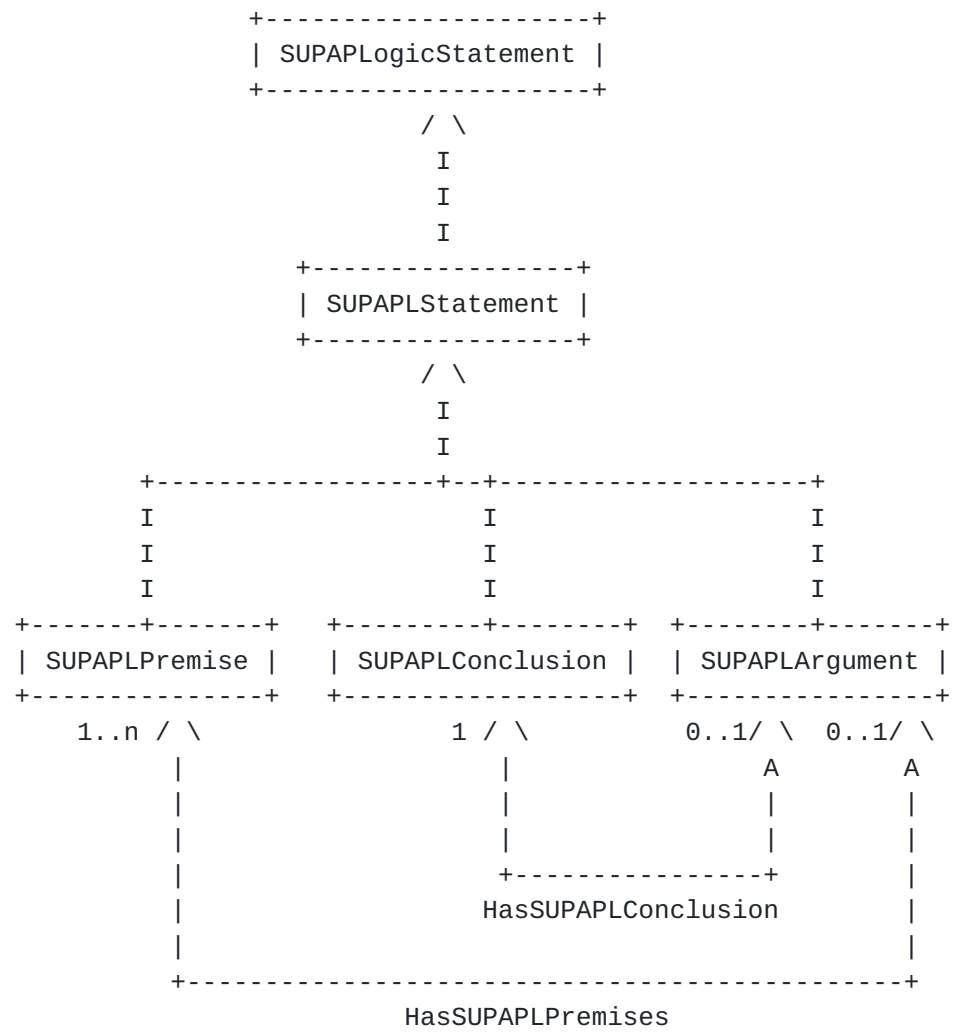


Figure 3: Overview of SUPA Propositional Logic Abstractions

As shown in Figure 3, a SUPAPLArgument consists of a set of one or more SUPAPLPremises and a single SUPAPLConclusion. The multiplicity of the two aggregations is 0..1 on the aggregate side to enable SUPAPLPremises and SUPAPLConclusions to be created and stored indepedent of being used in a SUPAPLArgument.

In PL, each possible atomic fact requires a separate propositional symbol. This can lead to a large amount of premises required to form a conclusion.

FOL provides a richer knowledge representation by using:

- objects (i.e., terms), which define individual entities
- properties (i.e., unary predicates on terms), which distinguishes objects from each other
- relations (i.e., n-ary predicates on terms), which define facts among a set of objects, and
- functions (i.e., the mapping from one set of terms to another set of terms).

FOL may be conceptualized as follows:

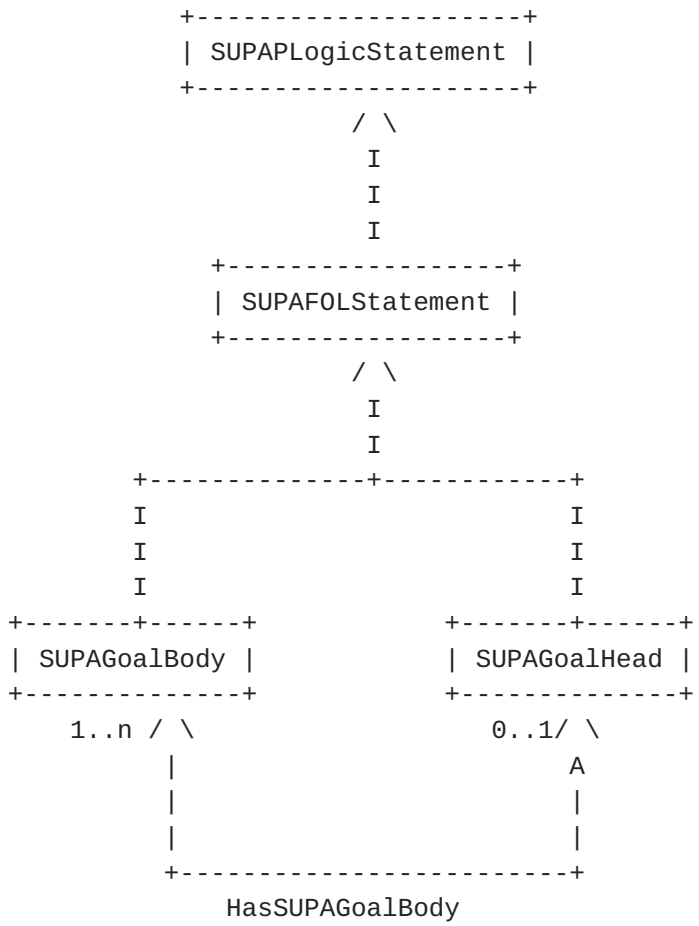


Figure 4: Overview of SUPA First Order Logic Abstractions

FOL Syntax may be described using the following grammar:

```
Sentence
  : AtomicSentence
  | Sentence Connective Sentence
  | (Quantifier Variable)+ Sentence
  | 'NOT' Sentence
  | function '(' Sentence ')'
  ;
```

4.5. SGPIM Assumptions

Most policy models (e.g., [2], [4], and [6]) are built as part of an overarching model. SUPA DOES NOT assume that it is the "root class of everything". Rather, the SUPA information model is built as a single inheritance model fragment to accommodate inserting the SUPA model into another model (e.g., the root of the SUPA model becomes a subclass of the other model). This is shown in Figure 5.

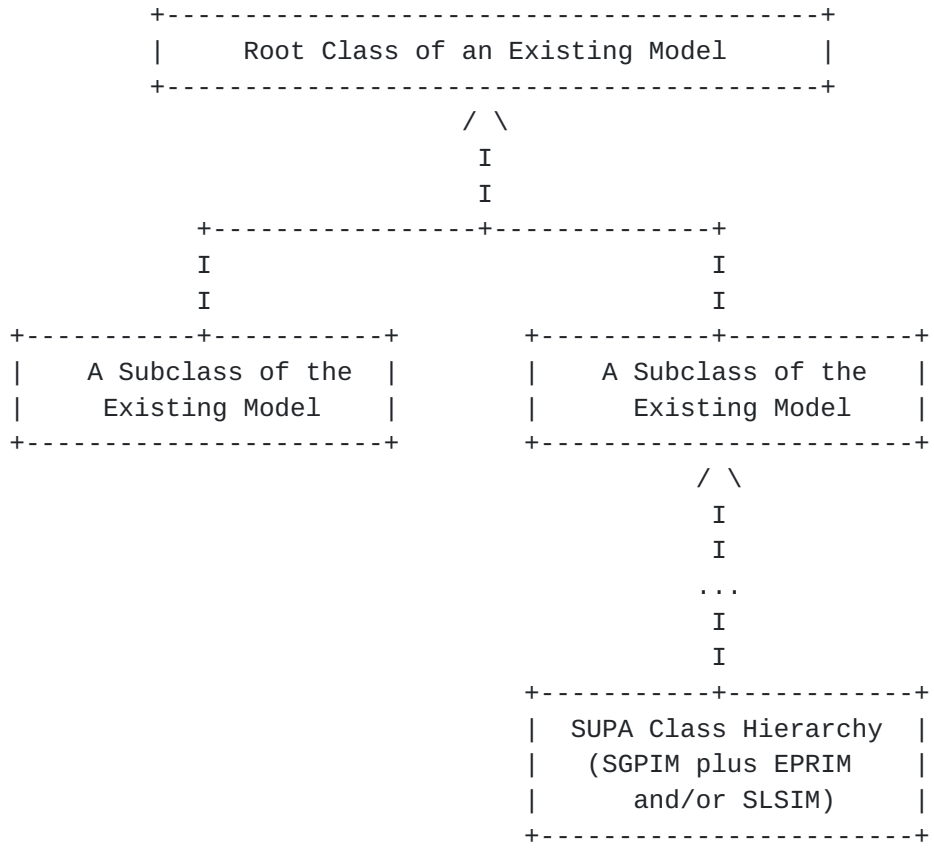


Figure 5: Integrating SUPA into an Existing Model

4.6. Scope of Previous Work

Insert intro paragraph and reference SUPA Gap Analysis [6]. Some salient points on previous policy models:

- [RFC3060] and [RFC3460] only define a policy rule that consists of a condition clause and an action clause; it does not define an ECA policy rule, nor does it define a LogicStatement
- [4] is more elaborate than [RFC3060] and [RFC3460], but suffers from the same limitations
- [5] defines four types of policies (i.e., ECA, Goal, UtilityFunction, and Promise), but does not have the detail defined in this document

Rest to be finished. Sections will include:

- Description of, and problems with, [RFC3060]
- Description of, and problems with, [RFC3460]
- Should this section also talk about CIM or SID? I personally think that this should be in the gap analysis...

5. SGPIM Model

This section defines the classes and relationships of the SGPIM.

5.1. Overview

The overall class definition is shown in Figure 6. SUPAPolicy is the root of the SUPA class hierarchy. For implementations, it is assumed that SUPAPolicy is subclassed from a class from another model. In Figure 6, indentation represents subclassing.

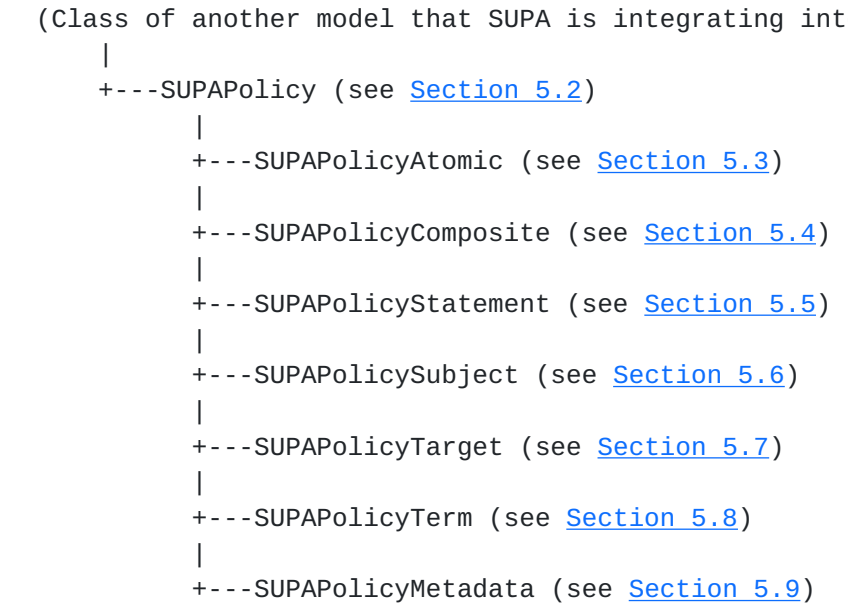


Figure 6: Main Classes of the SPGIM

The following subsections define the classes of the SGPIM. If a class has attributes, those attributes are also defined. Relationships are defined according to the class that is the "owner", or primary actor, participating in the relationship.

Classes, attributes, and relationships that are marked as "mandatory" MUST be part of a conformant implementation. Classes, attributes, and relationships that are marked as "optional" SHOULD be part of a conformant implementation.

5.2. The Abstract Class "SUPAPolicy"

This is a mandatory abstract class. This class is the root of the SUPA class hierarchy. It defines the common attributes and relationships that all SUPA subclasses inherit. All SUPA classes inherit from this class.

Figure 7 shows the SUPAPolicy class, and two of its subclasses (SUPAPolicyAtomic and SUPAPolicyComposite). This is an implementation of the composite pattern [3], which enables a SUPAPolicy to be made up of a stand-alone object (an instance of a SUPAPolicyAtomic class) or a hierarchy of objects (i.e., instances of one or more SUPAPolicyAtomic and SUPAPolicyComposite classes). The use of this software pattern enables SUPA Policies to be designed as individual objects and/or hierarchies of objects.

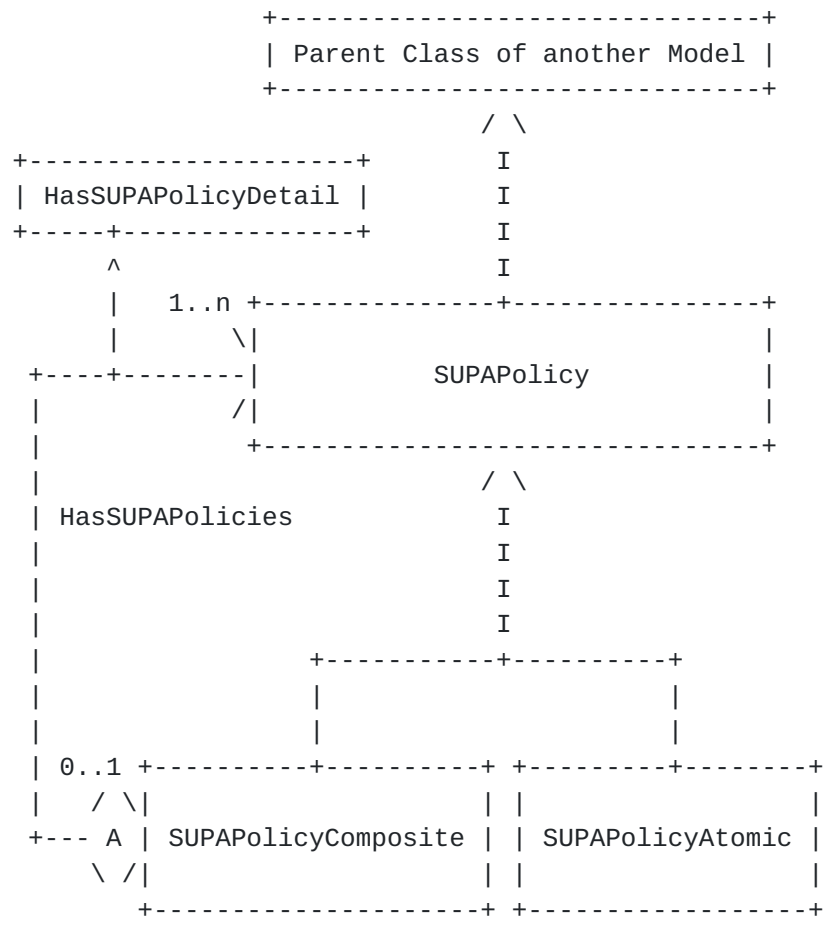


Figure 7: The SUPAPolicy Class Hierarchy

Note that a SUPAPolicy is a PolicyContainer object. A SUPAPolicyAtomic as well as a SUPAPolicyComposite are also PolicyContainer objects. SUPAPolicy was abstracted from DEN-ng [2], and a version of this class is in the process of being added to the policy framework defined in the TM Forum ZOOM model [5].

In figure 7:

- Both SUPAPolicyComposite and SUPAPolicyAtomic inherit from SUPAPolicy
- The diamond with an enclosed "A" represents an aggregation (see Section 3.2.3.4)
- The HasSUPAPolicies aggregation is implemented as an association class (see Section 3.2.3.6)
- The multiplicity of the HasSUPAPolicies aggregation is 0..1 - 1..n (zero or one SUPAPolicyComposite object instances can aggregate one or more SUPAPolicy object instances, see Section 3.2.3.7)
- The arrow pointing at SUPAPolicy restricts the navigability of this aggregation (see Section 3.2.3.8)

5.2.1. SUPAPolicy Attributes

This section defines the attributes of the SUPAPolicy class. These attributes are inherited by all subclasses of the SUPAPolicy class.

5.2.1.1. The Attribute "supaObjectIDContent"

This is a mandatory attribute that represents part of the object identifier of an instance of this class. It is a string attribute, and defines the content of the object identifier. It works with another class attribute, called supaObjectIDFormat, which defines how to interpret this attribute. These two attributes form a tuple, and together enable a machine to understand the syntax and value of an object identifier for the object instance of this class. This is based on the DEN-ng class design [2].

One of the goals of SUPA is to be able to generate different data models that support different types of protocols and repositories. This means that the notion of an object ID must be generic. In this way, different naming schemes, such as those depending on URIs, FQDNs, primary key - foreign key relationships, and UUIDs can all be accommodated.

5.2.1.2. The Attribute "supaObjectIDFormat"

This is a mandatory attribute that represents part of the object identifier of an instance of this class. It is a string attribute, and defines the format of the object identifier. It works with another class attribute, called supaObjectIDContent, which defines the content of the object ID. These two attributes form a tuple, and together enable a machine to understand the syntax and value of an object identifier for the object instance of this class. This is based on the DEN-ng class design [2].

5.2.1.3. The Attribute "supaPolicyName"

This is an optional string attribute that defines the name of this Policy. This enables any existing generic naming attribute to be used for generic naming, while allowing this attribute to be used to name Policy entities in a common manner. Note that this is NOT the same as the commonName attribute of the Policy class defined in RFC3060 [RFC3060], as that attribute is intended to be used with just X.500 cn attributes.

5.2.2. SUPAPolicy Relationships

This section defines the relationships of the SUPAPolicy class.

5.2.2.1. The Relationship "HasSUPAPolicies"

This is a mandatory aggregation that defines the set of SUPAPolicies that are contained in the instance of this particular SUPAPolicyComposite object. The multiplicity of this relationship is defined as 0..1 on the aggregate (SUPAPolicyComposite) side, and 1..n on the part (SUPAPolicy) side. This means that this relationship is optional, but if it is instantiated, then one or more SUPAPolicy objects are contained in this particular SUPAPolicyComposite object. The semantics of this aggregation are implemented using the HasSUPAPolicyDetail association class.

5.2.2.2. The Association Class "HasSUPAPolicyDetail"

This is a mandatory concrete association class that defines the semantics of the HasSUPAPolicies aggregation. This enables the attributes and relationships of the HasSUPAPolicyDetail class to be used to constrain which SUPAPolicy objects can be aggregated by this particular SUPAPolicyComposite object instance. Attributes will be added to this class at a later time.

5.3. The Abstract Class "SUPAPolicyAtomic"

This is a mandatory abstract class. This class is a type of PolicyContainer.

A SUPAPolicyAtomic class represents a SUPA Policy that can operate as a single, stand-alone, manageable object. Put another way, a SUPAPolicyAtomic object can NOT be modeled as a set of hierarchical SUPAPolicy objects; if this functionality is required, then a SUPAPolicyComposite object must be used.

No attributes are currently defined for the SUPAPolicyAtomic class. It serves as a superclass for the different types of SUPA Policies that are defined. In this release, both a SUPAECAPolicyRule (see Section 6) as well as a SUPALogicStatement (see Section 7) are defined as subclasses of the SUPAPolicyAtomic class.

SUPAPolicy was abstracted from DEN-ng [2], and a version of this class is in the process of being added to the policy framework defined in the TM Forum ZOOM model [5].

5.4. The Concrete Class "SUPAPolicyComposite"

This is a mandatory concrete class. This class is a type of PolicyContainer.

A SUPAPolicyComposite class represents a SUPA Policy as a hierarchy of Policy objects, where the hierarchy contains instances of a SUPAPolicyAtomic and/or SUPAPolicyComposite object. Each of the SUPA Policy objects, including the outermost SUPAPolicyComposite object, are separately manageable. More importantly, the SUPAPolicyComposite object can aggregate any SUPAPolicy subclass. Hence, it can be used to form hierarchies of SUPAPolicies as well as associate SUPAPolicySubjects and/or SUPAPolicyTargets to a given SUPAPolicy.

SUPAPolicy was abstracted from DEN-ng [2], and a version of this class is in the process of being added to the policy framework defined in the TM Forum ZOOM model [5].

5.4.1. SUPAPolicyComposite Attributes

This section defines the attributes of the SUPAPolicyComposite class. The combination of these two attributes provides a more flexible and powerful solution compared to [RFC3060] and [RFC3460].

5.4.1.1. The Attribute "supaPCIsMatchAll"

This is an optional Boolean attribute. If its value is TRUE, then ALL SUPAPolicies that are contained in this SUPAPolicyComposite object will be evaluated, regardless of whether a SUPAPolicy fails to execute correctly or not. If its value is FALSE, then only the FIRST SUPAPolicy contianed in this SUPAPolicyComposite object will be evaluated. The default value is TRUE.

5.4.1.2. The Attribute "supaPCFailureStrategy"

This is an optional non-negative enumerated integer attribute, whose values are used to define what action(s) should be taken if a failure occurs when executing a SUPAPolicy object that is contained in this SUPAPolicyComposite object. Values include:

0: undefined
1: stop execution
2: attempt rollback on failed policy
3: attempt rollback on all policies
4: ignore failure and continue

A value of 0 can be used as an error condition. A value of 1 means that ALL execution is stopped, and that other SUPAPolicies that otherwise would have been executed are ignored. A value of 2 means that execution is stopped, and a rollback of that SUPAPolicy (and ONLY that SUPAPolicy) is attempted. A value of 3 means that execution is stopped, and all SUPAPolicies that have been previously executed (including the one that just failed) are rolled back. A value of 4 means that any failure will be ignored, and all SUPAPolicies contained in this SUPAPolicyComposite object will be executed.

5.4.2. SUPAPolicyComposite Relationships

This section defines the relationships of SUPAPolicyComposite.

5.4.2.1. The Aggregation "HasSUPAECAPolicyRules"

This is a mandatory aggregation that defines the set of SUPAECAPolicyRules that are contained in the instance of this particular SUPAECAPolicyRuleComposite object. The multiplicity of this relationship is defined as 0..1 on the aggregate (SUPAECAPolicyRuleComposite) side, and 1..n on the part (SUPAECAPolicyRule) side. This means that one or more SUPAECAPolicyRules can be contained in a SUPAECAPolicyRuleComposite object instance. However, a SUPAECAPolicyRule does not have to be associated with a SUPAECAPolicyRuleComposite; this is necessary to enable SUPAECAPolicyRuleAtomic object instances to be used in a stand-alone manner. The semantics of this aggregation are implemented using the HasSUPAECAPolicyRulesDetail association class.

5.4.2.2. The Association Class "HasSUPAECAPolicyRulesDetail"

This is a mandatory concrete association class that defines the semantics of the HasSUPAECAPolicyRules aggregation. This enables the attributes and relationships of this association class to be used to constrain which SUPAECAPolicyRule objects can be aggregated by this particular SUPAECAPolicyRuleComposite object instance. Attributes will be added to this class at a later time.

5.5. The Abstract Class "SUPAPolicyStatement"

This is a mandatory abstract class that separates the representation of a SUPAPolicy from its implementation. This abstraction is missing in [RFC3060], [RFC3460], and [4]. There are three principal subclasses of SUPAPolicyStatement:

- o SUPAEncodedClause, which is a mechanism to directly encode the content of the SUPAPolicyStatement into a set of attributes; this is described in more detai lin Section 5.5.2.
- o SUPABooleanClause, which defines a SUPAPolicyStatement as a set of one or more clauses; multiple clauses may be combined with Boolean AND and OR operators. This defines a SUPAPolicy as a completely reusable set of SUPAPolicy objects that are structured in an ECA form, and is described in more detail in Section 6.10.
- o SUPALogicClause, which defines a SUPAPolicyStatement as either a fact or a clause; both are expressed in first-order logic. This defines a SUPAPolicy as a completely reusable set of SUPAPolicy objects that are structured in FOL, and is described in more detail in Section 7.5.

A SUPAPolicy MAY be constructed using any combination of the above three subclasses.

Both SUPAECAPolicyRules (see Section 6) and SUPALogicStatements (see section 7) MUST use SUPAPolicyStatements to define their content. This enables the content of these different types of Policy to be represented in a common manner.

Both SUPAECAPolicyRules and SUPALogicStatements MAY use a SUPAEncodedClause to define their content.

SUPAECAPolicyRules SHOULD also use a SUPABooleanClause to define its content, while SUPALogicStatements SHOULD also use a SUPALogicClause to define its content.

SUPAPolicyStatement was abstracted from DEN-ng [2], and a version of this class is in the process of being added to the policy

framework defined in the TM Forum ZOOM model [5].

A class diagram showing SUPAPolicyStatement is shown in Figure 8. Note that in Figure 8:

- SUPAPolicyStatement, SUPAPolicyAtomic, and SUPAPolicyComposite are subclasses of SUPAPolicy
- A SUPAEncodedClause is a subclass of SUPAPolicyStatement, and may be used by either a SUPAECAPolicyRule or a SUPALogicStatement
- Both the HasSUPAPolicyStatements and the HasSUPAPolicies aggregations are implemented as association classes

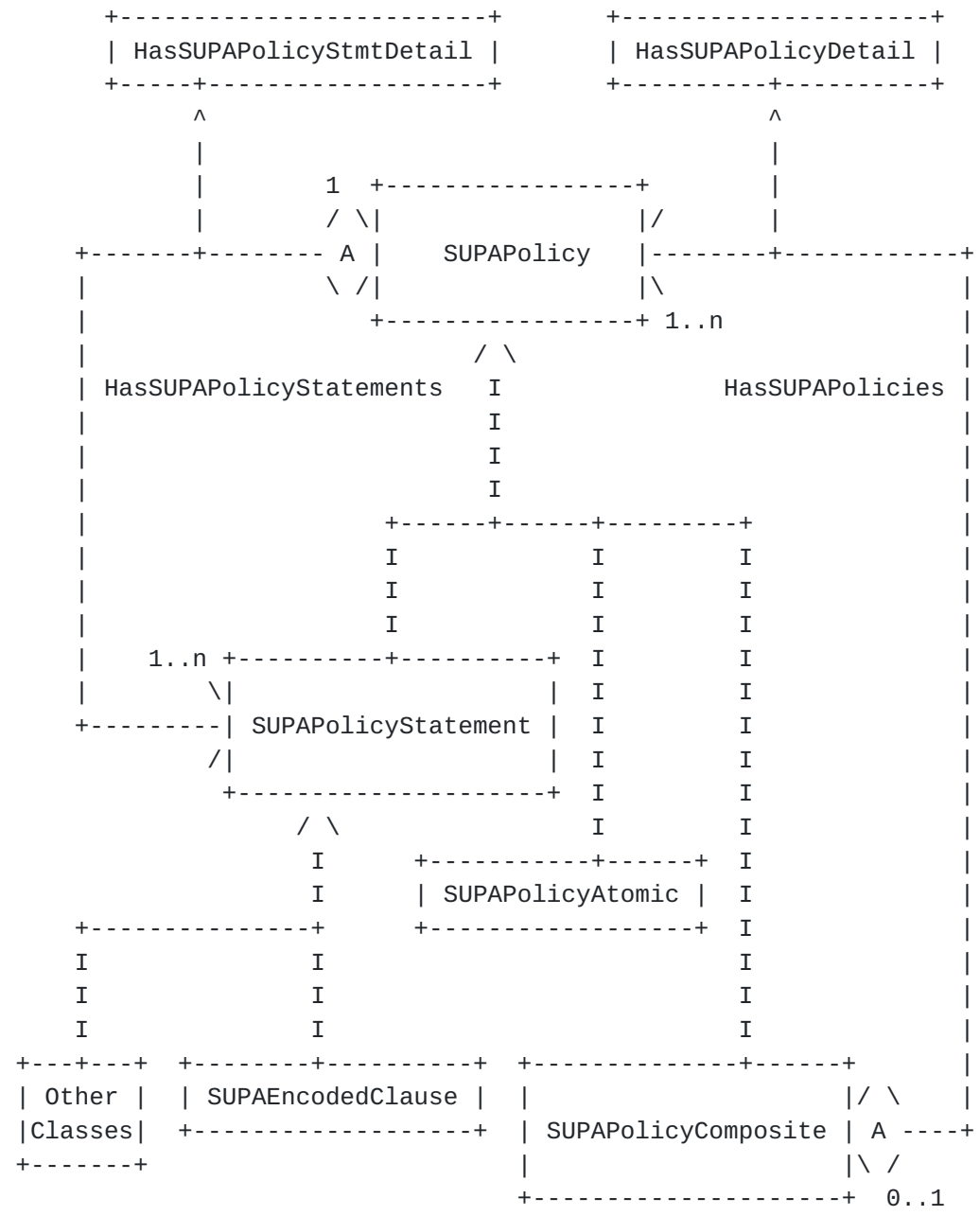


Figure 8: SUPAPolicyStatements and SUPAPolicy Classe

5.5.1. SUPAPolicyStatement Attributes

This section defines the attributes of the SUPAPolicyStatement class. These attributes are inherited by all subclasses of the SUPAPolicyStatement class.

5.5.1.1. The Attribute "supaPolicyStmtAdminStatus"

This is an optional attribute, which is an enumerated non-negative integer. It defines the current administrative status of this SUPAPolicyStatement.

This attribute can be used to place this particular SUPAPolicyStatement into a specific administrative state, such as enabled, disabled, or in test. Note that since a SUPAPolicy (e.g., a SUPAECAPolicyRule or a SUPALogicStatement) is made up of SUPAPolicyStatements, this enables all or part of a SUPAPolicy to be administratively controlled. Values include:

0: Unknown (an error state)
1: Enabled
2: Disabled
3: In Test (i.e., no operational traffic can be passed)

Value 0 denotes an error that prevents this SUPAPolicyStatement from being used. Values 1 and 2 mean that this SUPAPolicyStatement is administratively enabled or disabled, respectively. A value of 4 means that this SUPAPolicyStatement is in a special test mode.

5.5.1.2. The Attribute "supaPolicyStmtExecStatus"

This is an optional attribute, which is an enumerated non-negative integer. It defines whether this SUPAPolicyStatement is currently in use and, if so, what its status is.

This attribute can be used to place this particular SUPAPolicyStatement into a specific execution state, such as enabled, disabled, or in test. Note that since a SUPAPolicy (e.g., a SUPAECAPolicyRule or a SUPALogicStatement) is made up of SUPAPolicyStatements, this enables all or part of a SUPAPolicy to be administratively controlled.

Values include:

0: Unknown (an error state)
1: Working (i.e., in use and no errors reported)
2: Not Working (i.e., in use, but errors have been reported)
3: In Test (i.e., no operational traffic can be passed)
4: Available (i.e., could be used, but currently isn't)
5: Not Available (i.e., not available for use)

Value 0 denotes an error that prevents this SUPAPolicyStatement from being used. Values 1-3 mean that this SUPAPolicyStatement is in use; in addition, this SUPAPolicyStatement is working correctly, not working correctly, or in a special test state, respectively. Values 4-5 mean that this SUPAPolicyStatement is not currently in use; a value of 4 means that it is available and could be used, while a value of 5 means that it is unavailable.

5.5.2. SUPAPolicyStatement Subclasses

As stated before, the primary purpose of SUPAPolicyStatement is to define a common type of Policy statement that can be used to represent policy content regardless of the type of SUPAPolicy that is being used (e.g., it is independent of the requirements of a SUPAECAPolicyRule or a SUPALogicStatement). The SGPIM currently defines one subclass of SUPAPolicyStatement, called a SUPAEncodedClause, which can be used by both SUPAECAPolicyRules as well as SUPALogicStatements. Note that clauses dedicated to the specific use of a SUPAECAPolicyRule and a SUPALogicStatement are defined in Sections 6 and 7, respectively.

5.5.2.1. The Concrete Class "SUPAEncodedClause"

This is a mandatory concrete class that specializes (i.e., is a subclass of) a SUPAPolicyStatement. It defines a generalized extension mechanism for representing SUPAPolicyStatements that have not been modeled with other SUPAPolicy objects. Rather, the Policy Clause is directly encoded into the attributes of the SUPAEncodedClause. Note that other subclasses of SUPAPolicyStatement use SUPAPolicy objects to define their content.

This class uses two of its attributes (supaPolicyClauseContent and supaPolicyClauseFormat) for defining the content and format of a vendor-specific policy statement. This allows direct encoding of the policy statement, without having the "overhead" of using other objects. However, note that while this method is efficient, it does not reuse other SUPAPolicy objects. Rather, it can be thought of as a direct encoding of the policy statement. SUPAEncodedClause was abstracted from DEN-ng [2].

5.5.2.1.1. The Attribute "supaClauseContent"

This is a mandatory string attribute, and defines the content of this encoded clause of this clause. It works with another attribute of the SUPAEncodedClause class, called supaClauseFormat, which defines how to interpret this attribute. These two attributes form a tuple, and together enable a machine to understand the syntax and value of the encoded clause for the object instance of this class. This is based on the DEN-ng class design [2].

5.5.2.1.2. The Attribute "supaClauseFormat"

This is a mandatory string attribute, and defines the format of this encoded clause. It works with another attribute of the SUPAEncodedClause class, called supaClauseContent, which defines the content (i.e., the value) of the encoded clause. These two attributes form a tuple, and together enable a machine to understand the syntax and value of the encoded clause for the object instance of this class. This is based on the DEN-ng class design [2].

5.5.2.1.3. The Attribute "supaClauseResponse"

This is an optional Boolean attribute that emulates a Boolean response of this clause, so that it may be combined with other subclasses of the SUPAPolicyStatement that provide a status as to their correctness and/or evaluation state.

5.5.3. SUPAPolicyStatement Relationships

This section defines the relationships of SUPAPolicyStatement.

5.5.3.1. The Aggregation "HasSUPAPolicyStatements"

This is a mandatory aggregation that defines the set of SUPAPolicyStatements that are contained in the instance of this particular SUPAPolicy object. This defines a SUPAPolicy object as being made up of at least one SUPAPolicyStatement. The multiplicity of this relationship is defined as 1 on the aggregate (SUPAPolicy) side, and 1..n on the part (SUPAPolicyStatement) side. This means that this relationship is mandatory, and each SUPAPolicy object is made up of at least one SUPAPolicyStatement object. The semantics of this aggregation are implemented using the HasSUPAPolicyStmtDetail association class.

5.5.3.2. The Association Class "HasSUPAPolicyStmtDetail"

This is a mandatory concrete association class that defines the semantics of the HasSUPAPolicyStatements aggregation. This enables the attributes and relationships of the HasSUPAPolicyStmtDetail class to be used to constrain which SUPAPolicyStatement objects can be aggregated by this particular SUPAPolicy object instance. Attributes will be added to this class at a later time.

5.6. The Abstract Class "SUPAPolicySubject"

This is an optional class that defines the set of managed entities that authored, or are otherwise responsible for, this SUPAPolicyStatement. Note that a SUPAPolicySubject does NOT evaluate or execute SUPAPolicies. Its primary use is for auditability. A SUPAPolicySubject SHOULD be mapped to a role (e.g., using the role-object pattern, as DEN-ng does). A class diagram is shown in Figure 9.

Figure 9. SUPAPolicySubject and SUPAPolicyTarget

SUPAPolicySubject was abstracted from DEN-ng [2], and a version of this class is in the process of being added to the policy framework defined in the TM Forum ZOOM model [5].

In Figure 9:

- o SUPAPolicySubject and SUPAPolicyTarget are both subclasses of SUPAPolicy
- o Both the HasSUPAPolicyTargets amd the HasSUPAPolicySubjects aggregations are implemented as association classes
- o The multiplicity of both of the above aggregations are 0..1 on the aggregate (SUPAPolicy) side and 0..n on the target (i.e., SUPAPolicySubject and SUPAPolicyTarget, respectively) side. This means that both aggregations are optional. If either is instantiated, then a SUPAPolicy MAY contain zero or more SUPAPolicySubject object instances and MAY contain zero or more SUPAPolicyTarget object instances.

5.6.1. SUPAPolicySubject Attributes

Attributes will be added to this class at a later time.

5.6.2. SUPAPolicySubject Relationships

This section defines the relationships of the SUPAPolicySubject class.

5.6.2.1. The Relationship "HasSUPAPolicySubjects"

This is an optional aggregation that defines the set of SUPAPolicySubjects that are contained in the instance of this particular SUPAPolicy object. This defines the set of entities that authored this particular SUPAPolicy object. The multiplicity of this relationship is defined as 0..1 on the aggregate (SUPAPolicy) side, and 0..n on the part (SUPAPolicySubject) side. This means that this relationship is optional, but if it is implemented, then this particular SUPAPolicy object was authored by this set of SUPAPolicySubjects. The semantics of this aggregation are implemented using the HasSUPAPolicySubjDetail association class.

5.6.2.2. The Association Class "HasSUPAPolicySubjDetail"

This is an optional concrete association class that defines the semantics of the HasSUPAPolicySubjects aggregation. This enables the attributes and relationships of the HasSUPAPolicySubjDetail class to be used to constrain which SUPAPolicySubject objects can be used to author this particular SUPAPolicy object instance.

Attributes will be added to this class at a later time.

5.7. The Abstract Class "SUPAPolicyTarget"

A PolicyTarget is a set of managed entities that a SUPAPolicy is applied to. This is determined by two conditions. First, the set of managed entities that are to be affected by the SUPAPolicy must all agree to play the role of a SUPAPolicyTarget. In general, a managed entity may or may not be in a state that enables SUPAPolicies to be applied to it to change its state; hence, a negotiation process may need to occur between the SUPAPolicySubject and the SUPAPolicyTarget, wherein the SUPAPolicyTarget consents to have SUPAPolicies applied to it.

Second, a SUPAPolicyTarget must be able to either process (either directly or with the aid of a proxy) SUPAPolicies or receive the results of a processed SUPAPolicy and apply those results to itself. If a proposed SUPAPolicyTarget meets both of these conditions, it SHOULD set its supaPolicyTargetEnabled Boolean attribute to a value of TRUE.

A SUPAPolicySubject SHOULD be mapped to a role (e.g., using the role-object pattern). Figure 9 shows a class diagram of the SUPAPolicyTarget.

SUPAPolicyTarget was abstracted from DEN-ng [2], and a version of this class is in the process of being added to the policy framework defined in the TM Forum ZOOM model [5].

5.7.1. SUPAPolicyTarget Attributes

The following subsections define the attributes of a SUPAPolicyTarget.

5.7.1.1. The Attribute "supaPolicyTargetEnabled"

This is an optional Boolean attribute. If its value is TRUE, then this indicates that this SUPAPolicyTarget is currently able to have SUPAPolicies applied to it. Otherwise, this SUPAPolicyTarget is not able to have SUPAPolicies applied to it.

5.7.2. SUPAPolicyTarget Relationships

This section defines the relationships of the SUPAPolicyTarget class.

5.7.2.1. The Relationship "HasSUPAPolicyTargets"

This is an optional aggregation that defines the set of SUPAPolicyTargets that are contained in the instance of this particular SUPAPolicy object. This defines the set of entities that will be operated on by this particular SUPAPolicy object. The multiplicity of this relationship is defined as 0..1 on the aggregate (SUPAPolicy) side, and 0..n on the part (SUPAPolicyTarget) side. This means that this relationship is optional, but if it is implemented, then this particular SUPAPolicy object will operate on this set of SUPAPolicyTargets. The semantics of this aggregation are implemented using the HasSUPAPolicyTgtDetail association class.

5.7.2.2. The Association Class "HasSUPAPolicyTgtDetail"

This is an optional concrete association class that defines the semantics of the HasSUPAPolicyTargets aggregation. This enables the attributes and relationships of the HasSUPAPolicyTgtDetail class to be used to constrain which SUPAPolicyTarget objects can be operated on by this particular SUPAPolicy object instance.

Attributes will be added to this class at a later time.

5.8. The Abstract Class "SUPAPolicyTerm"

This is a mandatory abstract class that is the parent of SUPAPolicy objects that are general purpose in nature, and which are not subclasses of SUPAPolicyAtomic, SUPAPolicyComposite, SUPAPolicyStatement, SUPAPolicySubject, or SUPAPolicyTarget.

The principal subclasses of SUPAPolicyTerm that are defined in this version of this document are SUPAPolicyVariable, SUPAPolicyOperator, and SUPAPolicyValue. These terms enable generic statements to be created from a set of reusable terms.

SUPAPolicyTerm is defined as an abstract class for two reasons:

1. This enables a single aggregation (SUPAPolicyTermsInStmt; see section 5.8.2.1) to be used to specify which object instances of which SUPAPolicyTerm subclasses are contained by a particular SUPAPolicyStatement object instance. Otherwise, a set of three aggregations would be required.
2. This enables a single class (SUPAPolicyTermsInStmtDetail; see section 5.8.2.2) to be used as a superclass to define which one of its subclasses participates in this relationship. The advantage of this design is that as more SUPAPolicyTerm subclasses are added in the future, the SUPAPolicyStatement object is not affected.

Note that this design emphasizes flexibility and genericity of the model. Specifically, this means that the concept of creating a SUPAPolicyStatement can take a generic form, consisting of the tuple {PolicyVariable, PolicyOperator, PolicyValue}. Note that this is one option for constructing SUPAPolicyStatements, and is not mandatory; hence, the multiplicity of the SUPAPolicyTermsInStmt aggregation (see Section 5.8.2.) is 0..n - 0..n.

This design is in marked contrast to most existing designs. For example, [RFC3060], [RFC3460], and [4] do not define an ECA Policy Rule; rather, they are limited to a Policy Rule that only has a condition clause and an action clause. Note that there is no mechanism for the system to trigger when a Policy Rule should be evaluated (because there is no event clause). In addition, [RFC3060], [RFC3460], and [4] do not define any type of logic statement (or, for that matter, any other type of Policy Rule).

SUPAPolicyTerm was abstracted from DEN-ng [2].

5.8.1. SUPAPolicyTerm Attributes

Currently, SUPAPolicyTerm defines two attributes, as described in the following subsections.

5.8.1.1 The Attribute "supaPolTermExprContent"

This is an optional string attribute that defines the content of an expression whose value defines the set of objects that are part of this SUPAPolicyTerm. It works with another class attribute, called supaPolicyTermExprFormat, which defines how to interpret this attribute. These two attributes form a tuple, and together enable a machine to understand the syntax and value of different expressions used to define the set of objects that are part of this SUPAPolicyTerm. This is based on the DEN-ng class design [2].

5.8.1.2. The Attribute "supaPolTermExprFormat"

This is a mandatory attribute that represents part of the object identifier of an instance of this class. It is a string attribute, and defines the format of the object identifier. It works with another class attribute, called supaObjectIDContent, which defines the content of the object ID. These two attributes form a tuple, and together enable a machine to understand the syntax and value of an object identifier for the object instance of this class. This is based on the DEN-ng class design [2].

5.8.2. SUPAPolicyTerm Relationships

Currently, SUPAPolicyTerm participates in a single relationship, as described in the following subsection.

5.8.2.1. The Aggregation "SUPAPolicyTermsInStmt"

This is a mandatory aggregation that defines the set of SUPAPolicyTerms that are contained in this SUPAPolicyStatement. A SUPAPolicyStatement can, in this version of this document, take two different forms that have different content requirements; these are SUPAECAPolicyRules and SUPALogicStatements. Therefore, the multiplicity of this relationship is defined as 0..n on the aggregate (SUPAPolicyStatement) side, and 0..n on the part (SUPAPolicyTerm) side. This means that a SUPAPolicyStatement does not have to contain a SUPAPolicyTerm; this is typically true for SUPALogicStatement. However, if a SUPAPolicyStatement does require one or more SUPAPolicyTerms, then those may be defined using this aggregation. The semantics of this aggregation are implemented using the SUPAPolicyTermsInStmtDetail association class.

5.8.2.2. The Association Class "SUPAPolicyTermsInStmtDetail"

This is a mandatory abstract association class that defines the semantics of the SUPAPolicyTermsInStmt aggregation. This enables the attributes and relationships of the SUPAPolicyTermsInStmtDetail class to be used to constrain which SUPAPolicyTerm objects can be aggregated by this particular SUPAPolicyStatement object instance.

The preferred design is to keep this association class abstract, and create three subclasses from it that constrain the set of SUPAPolicyVariables, SUPAPolicyOperators, and SUPAPolicyValues that are used with this particular SUPAPolicyStatement. This provides a direct and simple mapping to optimized data models. Alternatively, appropriate attributes could be added to this association class to define the constraint, but such attributes would also have to take into account the type of PolicyTerm subclass that is being constrained.

Attributes will be added to this class at a later time.

5.8.3. SUPAPolicyTerm Subclasses

The following three subsections define three subclasses of the SUPAPolicyTerm class.

5.8.3.1. The Concrete Class "SUPAPolicyVariable"

This is a mandatory abstract class that defines information that forms a part of a SUPAPOlicyStatement. It specifies a concept or attribute that should be compared to a value, as specifed in this SUPAPolicyStatement. If it is used in a SUPAECAPolicyRule, then its value MAY be able to be changed at any time. However, if it is used in a SUPALogicStatement, then it is typically bound to an expression, and keeps a single value during its entire lifetime. SUPAPolicyVariable was abstracted from DEN-ng [2].

The value of a SUPAPolicyVariable is typically compared to the value of a SUPAPolicyValue using the type of operator defined in a SUPAPolicyOperator.

SUPAPolicyVariables are used to abstract the representation of a SUPAPolicyRule from its implementation. Therefore, the design of SUPAPolicyVariables depends on two important factors. First, some SUPAPolicyVariables are restricted in the values and/or the data type that they may be assigned. For example, port numbers cannot be negative, and they cannot be floating-point numbers. Thus, any SUPAPolicyVariable can have a set of constraints associated with it that restrict the value, data type, and other semantics of the SUPAPolicyVariable when used in a particular SUPAPolicyStatement. Second, there is a high likelihood that specific applications will need to use their own variables that have specific meaning to a particular application.

SUPAPolicyVariable constraints are implemented using an AGGREGATION THAT WILL BE DEFINED IN THE NEXT VERSION OF THIS DOCUMENT.

SUPAPolicyVariable extensibility is accommodated by providing two subclasses of SUPAPolicyVariable: model-based variables and application-defined variables. THESE WILL BE DEFINED IN THE NEXT VERSION OF THIS DOCUMENT.

5.8.3.2. The Concrete Class "SUPAPolicyOperator"

This is a mandatory concrete class for modeling different types of operators that are used in a SUPAPolicyStatement. The restriction of the type of operator used in a SUPAPolicyStatement restricts the semantics that can be expressed in that SUPAPolicyStatement. SUPAPolicyOperator was abstracted from DEN-ng [2].

5.8.3.2.1. The Attribute "supaPolOpType"

This is a mandatory non-negative enumerated integer that specifies the various types of operators that are allowed to be used in this particular SUPAPolicyStatement. Values include:

0: Unknown
1: Match
2: Greater than
3: Greater than or equal to
4: Less than
5: Less than or equal to
6: Equal to
7: Not equal to
8: IN
9: NOT IN
10: SET
11: CLEAR

Note that 0 is an unacceptable value. Its purpose is to support dynamically building a SUPAPolicyStatement by enabling the application to set the value of this attribute to a standard default value if the real value is not yet known.

5.8.3.3. The Concrete Class "SUPAPolicyValue"

The SUPAPolicyValue class is a mandatory abstract class for modeling different types of values and constants that occur in a PolicyStatement. SUPAPolicyValue was abstracted from DEN-ng [2].

The value of a SUPAPolicyVariable is typically compared to the value of a SUPAPolicyValue using the type of operator defined in a SUPAPolicyOperator.

SUPAPolicyValues are used to abstract the representation of a SUPAPolicyRule from its implementation. Therefore, the design of SUPAPolicyValues depends on two important factors. First, just as with SUPAPolicyVariables (see Section 5.8.3.1), some types of SUPAPolicyValues are restricted in the values and/or the data type that they may be assigned. Second, there is a high likelihood that specific applications will need to use their own variables that have specific meaning to a particular application.

SUPAPolicyVariable constraints are implemented using an AGGREGATION THAT WILL BE DEFINED IN THE NEXT VERSION OF THIS DOCUMENT.

SUPAPolicyValue extensibility is accommodated by providing two subclasses of SUPAPolicyValue: model-based values and application-defined values. THESE WILL BE DEFINED IN THE NEXT VERSION OF THIS DOCUMENT.

5.9. The Abstract Class "SUPAPolicyMetadata"

THIS WILL BE DEFINED IN THE NEXT VERSION OF THIS DOCUMENT.

SUPAPolicyMetadata was abstracted from DEN-ng [2]. A more complete representation of metadata, as defined in [2], is in the process of being added to the policy framework defined in the TM Forum ZOOM model [5].

5.9.1. SUPAPolicyMetadata Attributes

THIS WILL BE DEFINED IN THE NEXT VERSION OF THIS DOCUMENT.

5.9.2. SUPAPolicyMetadata Relationships

THIS WILL BE DEFINED IN THE NEXT VERSION OF THIS DOCUMENT.

6. SUPA ECAPolicyRule Information Model

This section defines the classes, attributes, and relationships of the SUPA ECAPolicyRule Information Model (EPRIM).

6.1. Overview

Conceptually, the EPRIM is a set of subclasses that specialize the concepts defined in the SGPIM for representing the components of a Policy that uses ECA semantics. This is shown in Figure 10.

Figure 10: The EPRIM Refining the SGPIM

Specifically, the EPRIM specializes the SUPAPolicyAtomic class to create a SUPAECAPolicyRule; it also specializes the SUPAPolicy class to create a SUPAECAComponent, and the SUPAPolicyStatement to create a SUPABooleanClause. The SUPAECAPolicyRule uses the rest of the SGPIM infrastructure to define a complete Policy model according to ECA semantics.

The overall strategy for refining the SGPIM is as follows:

- SUPAECAPolicyRule is defined as a subclass of the SGPIM SUPAPolicyAtomic class
- A SUPAECAPolicyRule has event, condition, and action clauses; each of these are created by either a SUPABooleanClause or a SUPAEncodedClause
- A SUPAECAComponent defines SUPAEvent, SUPACondition, and SUPAAction objects that are used to create the event, condition, and action clauses of a SUPAECAPolicyRule
- Both a SUPABooleanClause and a SUPAEncodedClause inherit the HasSUPAECAComponents aggregation, so both of these types of clauses can use SUPAECAComponents in their construction
- Both a SUPABooleanClause and a SUPAEncodedClause inherit the SUPAPolicyTermsInStmt aggregation, so both of these types of clauses can use SUPAPolicyTerms in their construction
- An optional set of SGPIM SUPAPolicySubjects can be defined to represent the authoring of a SUPAECAPolicyRule
- An optional set of SGPIM SUPAPolicyTargets can be defined to represent the set of managed entities that will be affected by this SUPAECAPolicyRule
- An optional set of SUPAPolicyMetadata can be defined for any of the objects that make up a SUPAECAPolicyRule and/or a SUPAECAComponent

6.2. Constructing a SUPAECAPolicyRule

There are several different ways to construct a SUPAECAPolicyRule. The simplest approach is as follows:

- Define three types of SUPABooleanClauses (see Section 6.7), one each for the event, condition, and action clauses that make up a SUPAECAPolicyRule (see Section 6.4)
- Define a set of SUPAEvent, SUPACondition, and SUPAAction objects (see Section 6.5.1, 6.5.2, and 6.5.3, respectively), and associate each with the SUPABooleanClause that represents the event, condition, and action clauses, respectively, of the SUPAECAPolicyRule
- Define a SUPAECAPolicyRule, which is a subclass of the SGPIM SUPAPolicyAtomic class (see Section 5.3)

- Aggregate the three SUPABooleanClauses into the SUPAECAPolicyRule
- Optionally, define a set of SUPAPolicySubjects and SUPAPolicyTargets, and aggregate them into the SUPAECAPolicyRule
- Optionally, define SUPAPolicyMetadata for any of the above objects, and aggregate them to the SUPAPolicy objects that the SUPAPolicyMetadata applies to

6.3. Working With SUPAECAPolicyRules

A SUPAECAPolicyRule is a type of SUPAPolicy. It is a tuple that MUST have three clauses, defined as follows:

- The event clause defines a Boolean expression that, if TRUE, triggers the evaluation of its condition clause (if the event clause is not TRUE, then no further action for this policy rule takes place).
- The condition clause defines a Boolean expression that, if TRUE, enables the actions in the action clause to be executed (if the condition clause is not TRUE, then no further action for this policy rule takes place).
- The action clause is a set of actions, whose execution MAY be controlled by the SUPAMmetadata of the policy rule.

Each of the three clauses can be constructed from either a SUPAEncodedClause or a SUPABooleanClause. The advantage of using SUPAEncodedClauses is simplicity, as the content of the clause is encoded directly into the attributes of the SUPAEncodedClause. The advantage of using SUPABooleanClauses is reusability, since each term in each clause is potentially a reusable object.

Since a SUPABooleanClause is a subclass of a SUPAPolicyStatement (see Section 5.5), it can aggregate SUPAPolicyTerm objects as well as SUPAECAComponent objects. Therefore, a SUPAECAPolicyRule can be built entirely from objects defined in the SGPIM. As will be shown in Section 7.3, this is also true for SUPALogicStatements.

The construction of a SUPAECAPolicyRule is shown in Figure 11, and is explained in Section 6.4.

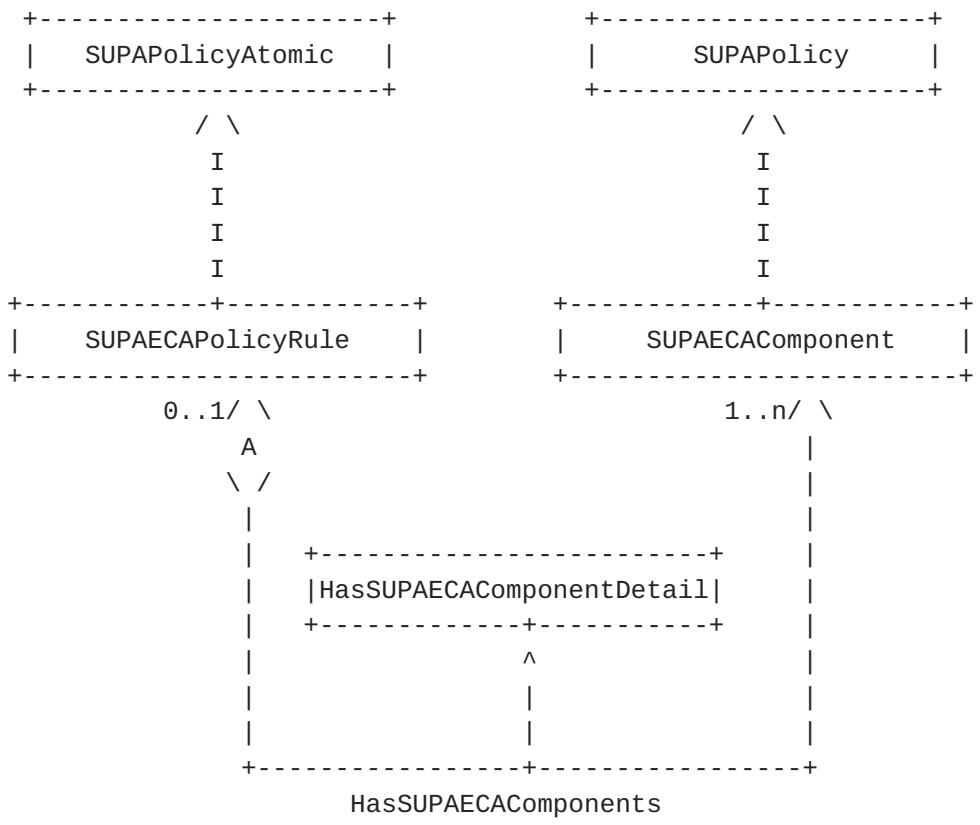


Figure 11. SUPAECAPolicyRule Clauses

NOTE: This is a simplified design, inspired from [2]. The HasSUPAECAComponents aggregation is implemented using the HasSUPAECAComponentDetail association class. This is an abstract class further described in Section 6.4.2. It has three concrete subclasses, one each that correspond to the three subclasses of SUPAECAComponent (i.e., SUPAEvent, SUPACondition, and SUPAAction), which are all concrete. This enables one aggregation to define a set of constraints between a SUPAPolicyStatement and the set of Events, Conditions, and/or Actions that it can contain.

6.4. The Concrete Class "SUPAECAPolicyRule"

This is a concrete mandatory class. In keeping with the original DEN-ng model [1], this class is a PolicyContainer that contains PolicyEvents, PolicyConditions, PolicyActions, and optionally, PolicyMetadata. As such, it doesn't have an inherent relationship with PolicySubject or PolicyTarget; these all represent the specific semantics for a particular SUPAPolicy. Hence, such semantics are defined in an instance of the SUPAPolicyComposite class that contains a SUPAECAPolicyRule, if they are required.

The semantics of a SUPAECAPolicyRule may be conceptualized as follows:

```
ON RECEIPT OF <policy-event-clause>
   IF <policy-condition-clause> EVALUATES TO TRUE
      THEN EXECUTE <policy-action-clause>
   END
END
```

In the above, a policy-event-clause, policy-condition-clause, and a policy-action-clause are each instances of either a SUPAEncodedClause or a SUPABooleanClause.

This class was based on the ECAPolicyRule class of [2].

6.4.1. SUPAECAPolicyRule Attributes

Currently, the SUPAECAPolicyRule defines two attributes, as described in the following subsections.

6.4.1.1. The Attribute "supaECAPRDeployStatus"

This is an optional attribute, which is an enumerated, non-negative integer. It defines the current deployment status of this SUPAECAPolicyRule. Both operational and test mode values are included in its definition. Values include:

0: undefined
1: deployed and enabled
2: deployed and in test
3: deployed but not enabled
4: ready to be deployed
5: not deployed

6.4.1.2. The Attribute "supaECAPRExecStatus"

This is an optional attribute, which is an enumerated, non-negative integer that defines the current execution status of this SUPAECAPolicyRule. Both operational and test mode values are included in its definition. Values include:

0: undefined
1: executed and SUCEEDED (operational mode)
2: executed and FAILED (operational mode)
3: currently executing (operational mode)
4: executed and SUCEEDED (test mode)
5: executed and FAILED (test mode)
6: currently executing (test mode)

6.4.2. SUPAECAPolicyRule Relationships

Currently, the SUPAECAPolicyRule does not define any relationships; rather, it uses those of the SPGIM and ERPIM to construct its ECA Policy Rules.

6.4.2. SUPAECAPolicyRule Subclasses

The composite pattern is applied to the SUPAECAPolicyRule class, enabling it to be used as either a stand-alone policy rule or as a hierarchy of policy rules.

6.4.2.1. The Concrete Class "SUPAECAPolicyRuleAtomic"

This is a mandatory concrete class. This class is a type of PolicyContainer. SUPAECAPolicyRuleAtomic was abstracted from DEN-ng [2].

A SUPAECAPolicyRuleAtomic class represents a SUPA ECA Policy Rule that can operate as a single, stand-alone, manageable object. Put another way, a SUPAECAPolicyRuleAtomic object can NOT be modeled as a set of hierarchical SUPAECAPolicyRule objects; if this is required, then a SUPAECAPolicyRuleComposite object must be used.

Attributes afor the SUPAECAPolicyRuleAtomic class WILL BE DEFINED IN THE NEXT VERSION OF THIS DOCUMENT.

6.4.2.2. The Concrete Class "SUPAECAPolicyRuleComposite"

This is a mandatory concrete class. This class is a type of PolicyContainer. SUPAECAPolicyRuleComposite was abstracted from DEN-ng [2]

A SUPAECAPolicyRuleComposite class represents a SUPA ECA Policy Rule as a hierarchy of Policy objects, where the hierarchy contains instances of a SUPAECAPolicyRuleAtomic and/or SUPAECAPolicyRuleComposite object. Each of the SUPA Policy objects, including the outermost SUPAECAPolicyRuleComposite object, are separately manageable. More importantly, the SUPAECAPolicyRuleComposite object can aggregate any SUPAECAPolicyRule subclass. Hence, it can be used to form hierarchies of SUPAECAPolicyRules as well as associate SUPAPolicySubjects and/or SUPAPolicyTargets to a given SUPAECAPolicyRule.

Attributes afor the SUPAECAPolicyRuleComposite class WILL BE DEFINED IN THE NEXT VERSION OF THIS DOCUMENT.

6.5. SUPAPolicyStatement Subclasses

Section 5.5.2 defines a common subclass of SUPAPolicyStatement, called SUPAEncodedClause, which any SUPAPolicy (rule or predicate) can use. This section describes another specialization of the SGPIM SUPAPolicyStatement class for use in constructing (only) SUPAECAPolicyRule objects.

The SUPAPolicyStatement class, and its subclasses, are based on similar classes in [2].

6.5.1. Designing SUPAPolicyStatements Using SUPABooleanClauses

A SUPABooleanClause specializes a SUPAPolicyClause, and defines a Boolean statement consisting of a standard structure in the form of a PolicyVariable, a PolicyOperator, and a PolicyValue. This design is based on the DEN-ng model [2]. For example, this enables the following Boolean clause to be defined:

```
Foo >= Bar AND Baz
```

where Foo is a PolicyVariable, >= is a PolicyOperator, and Bar is a PolicyValue. Note that in this approach, each of these three terms (i.e., the PolicyVariable, PolicyOperator, and PolicyValue) are subclasses of the SUPAPolicyTerm class, which is defined in Section 5.8). This enables the EPRIM, in conjunction with the SGPIM, to be used as a reusable class library. This encourages interoperability, since each element of the clause is itself an object defined by SUPA.

The addition of a negation in the above statement is provided by the supaTermIsNegated Boolean attribute in the SUPAPolicyTerm class. An entire clause is indicated as negated using the supaBoolIsNegated Boolean attribute in the SUPABooleanClause class.

A PolicyStatement is in Conjunctive Normal Form (CNF) if it is a conjunction (i.e., a sequence of ANDed terms), where each term is a disjunction (i.e., a sequence of ORed terms). Every statement that consists of a combination of AND, OR, and NOT operators can be written in CNF.

A PolicyStatement is in Disjunctive Normal Form (DNF) if it is a disjunction (i.e., a sequence of ORed terms), where each term is a conjunction (i.e., a sequence of ANDed terms). Every statement that consists of a combination of AND, OR, and NOT operators can be written in DNF.

The supaBoolISCNF Boolean attribute of the SUPABooleanClause class is TRUE if this SUPABooleanClause is in CNF, and FALSE otherwise.

The construction of more complex clauses, which consist of a set of simple clauses in conjunctive or disjunctive normal form (as shown in the above example), is provided by using the composite pattern [3] to construct two subclasses of SUPABooleanClause. These are called SUPABooleanClauseAtomic and SUPABooleanClauseComposite, and are defined in Sections 6.5.2.1 and 6.5.2.2, respectively. This enables instances of either a SUPABooleanClauseAtomic and/or a SUPABooleanClauseComposite to be aggregated into a SUPABooleanClauseComposite object.

6.5.2. The Abstract Class"SUPABooleanClause"

This is a mandatory abstract class that defines a clause as the following three-tuple:

{PolicyVariable, PolicyOperator, PolicyValue}

The composite pattern [3] is used in order to construct complex Boolean clauses from a set of SUPABooleanClause objects. This is why SUPABooleanClause is defined to be abstract - only instances of the SUPABooleanAtomic and/or SUPABooleanComposite classes can be used to construct a SUPABooleanClause.

Figure 12 below shows the composite pattern applied to the SUPABooleanClause class.

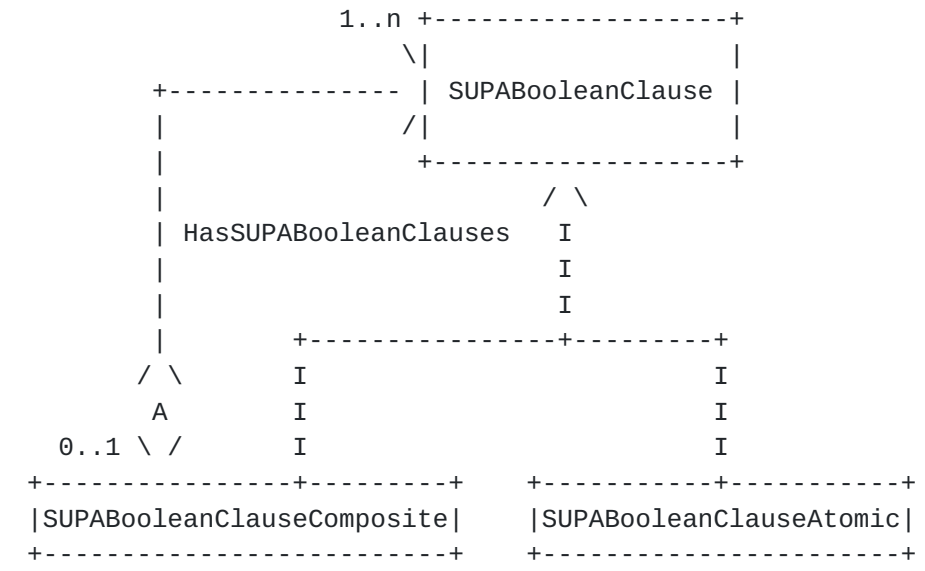


Figure 12. The Composite Pattern Applied to a SUPABooleanClause

The advantage of a SUPABooleanClause is that it is formed entirely from SUPAPolicy objects. This enhances both reusability as well as interoperability. Since this involves compositing a number of objects, data model implementations MAY optimize a SUPABooleanClause according to their application-specific needs (e.g., by flattening the set of classes that make up a SUPABooleanClause object into a single object).

6.5.2.1. SUPABooleanClause Attributes

The following sections define attributes of a SUPABooleanClause.

6.5.2.1.1. The Attribute "supaBoolIsNegated"

This is a mandatory Boolean attribute. If the value of this attribute is TRUE, then this SUPABooleanClause is negated.

6.5.2.2. SUPABooleanClause Relationships

The following subsections define the relationships of a SUPABooleanClause.

6.5.2.2.1. The Relationship "HasSUPABooleanClauses"

This is a mandatory aggregation that defines the set of SUPABooleanClauses that are aggregated by this SUPABooleanClauseComposite. This will either form a complete SUPABooleanClause from multiple clauses (which can be made up of SUPABooleanClauseAtomic and/or SUPABooleanClauseComposite object instances) or define another level in the SUPABooleanClause object hierarchy. The multiplicity of this relationship is 0..1 on the aggregate (SUPABooleanClauseComposite) side, and 1..n on the part (SUPABooleanClause) side. This means that one or more SUPABooleanClauses are aggregated and used to define this SUPABooleanClauseComposite object. The 0..1 cardinality on the SUPABooleanClauseComposite side is necessary to enable SUPABooleanClauses to exist (e.g., in a PolicyRepository) before they are used by a SUPABooleanClauseComposite.

6.5.3. SUPABooleanClause Subclasses

SUPABooleanClause defines two subclasses, as shown in Figure 12. They are both described in the following subsections.

6.5.3.1. The Abstract Class "SUPABooleanClauseAtomic"

This is a mandatory abstract class that represents a SUPABooleanClause that can operate as a single, stand-alone, manageable object. Put another way, a SUPABooleanClauseAtomic object can NOT be modeled as a set of hierarchical clauses; if this functionality is required, then a SUPABooleanClauseComposite object must be used.

No attributes are currently defined for the SUPABooleanClauseAtomic class. Its primary purpose is to aggregate SUPAPolicyVariable, SUPAPolicyOperator, and SUPAPolicyValue objects to form a complete SUPABoolean clause. As such, this class is defined as abstract to simplify data model optimization and mapping.

The three primary subclasses of the SUPABooleanClauseAtomic class are shown in Figure 13.

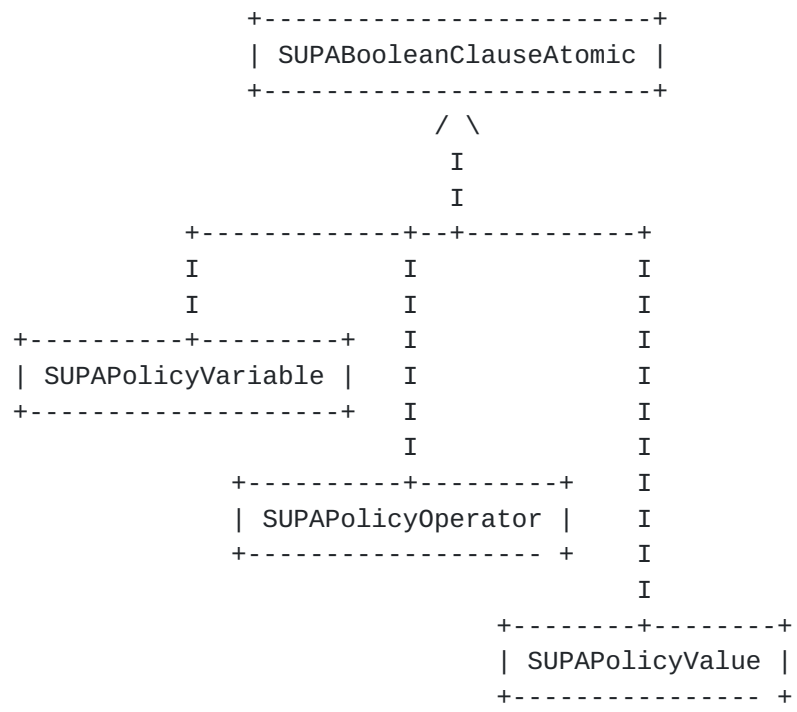


Figure 13. SUPABooleanClauseAtomic Subclasses

6.5.3.1.1. The Abstract Class "SUPAPolicyVariable"

This is a mandatory abstract class. It is similar to the PolicyVariable class of [RFC3460], but there are some important differences in the SUPA version of this class that make the SUPA version more generic than the version defined in [RFC3460]. The problems in the definition of the [RFC3460] version of this class are discussed in Section 6.5.3.1.1.1, and the SUPAPolicyVariable

class definition is defined in Section 6.5.3.1.1.2.

6.5.3.1.1.1. Problems with the RFC3460 Version of PolicyVariable

First, [RFC3460] says: "Variables are used for building individual conditions". While this is true, variables can also be used for building individual actions. This is reflected in the SUPAPolicyVariable definition.

Second, [RFC3460] says: "The variable specifies the property of a flow or an event that should be matched when evaluating the condition." While this is true, variables can be used to test many broader things than "just" a flow or an event. This is reflected in the SUPAPolicyVariable definition.

Third, in [RFC3460], defining constraints for a variable is limited to associating the variable with a PolicyValue. This is both cumbersome (because associations are costly), and not scalable, because it is prone to proliferating PolicyValue classes for every constraint (or range of constraints) that is possible. Therefore, in SUPA, this mechanism is replaced with using an association to a generic SUPAConstraint object.

Fourth, [RFC3460] is tightly bound to the DMTF CIM schema [4]. The CIM is a data model (despite its name), because:

- It uses keys and weak relationships, which are both concepts from relational algebra and thus, not technology-independent
- It has its own proprietary modeling language
- It contains a number of concepts that are not defined in UML (including overriding keys for subclasses)

6.5.3.1.1.2. The Abstract Class "SUPAPolicyVariable"

This class is based on a similar class defined in [2].

To be finished in the next version of this document.

The big question to be answered is whether to keep the PolicyImplicitVariable and PolicyExplicitVariable subclasses of [RFC3460] or not.

6.5.3.1.2. The Concrete Class "SUPAPolicyOperator"

This is a mandatory abstract class. Note that there is no equivalent to this class in [RFC3460], which causes a number of problems in the overloading of the semantics of an operator for defining clauses in an ECA policy rule. This class is based on a similar class defined in [2].

This will be defined in the next version of this document.

6.5.3.1.3. The Abstract Class "SUPAPolicyValue"

This is a mandatory abstract class. It is similar to the PolicyValue class of [RFC3460], but there are some important differences in the SUPA version of this class that make the SUPA version more generic than the version defined in [RFC3460]. The problems in the definition of the [RFC3460] version of this class are discussed in Section 6.5.3.1.3.1, and the SUPAPolicyVariable class definition is defined in Section 6.5.3.1.3.2.

6.5.3.1.3.1. Problems with the RFC3460 Version of PolicyValue

This will be defined in the next version of this document.

6.5.3.1.3.2. The Abstract Class "SUPAPolicyValue"

This class is based on a similar class defined in [2].

This will be defined in the next version of this document.

6.5.4. The Abstract Class "SUPABooleanClauseComposite"

This will be defined in the next version of this document.

6.5.4.1. SUPABooleanClauseComposite Attributes

This will be defined in the next version of this document.

6.5.4.2. SUPABooleanClauseComposite Relationships

This will be defined in the next version of this document.

6.6. The Abstract Class "SUPAECAComponent"

The principal subclasses of SUPAPolicyTerm that are defined in this version of this document are SUPAPolicyEvent, SUPAPolicyCondition, and SUPAPolicyAction.

SUPAPolicyTerm is defined as an abstract class for two reasons:

1. This enables a single aggregation (SUPAPolicyTermsInStmt; see section 5.8.2.1) to be used to specify which object instances of which SUPAPolicyTerm subclasses are contained by a particular SUPAPolicyStatement object instance. Otherwise, a set of three aggregations would be required.
2. This enables a single class (SUPAPolicyTermsInStmtDetail; see section 5.8.2.2) to be used as a superclass to define which one of its subclasses participates in this relationship. The advantage of this design is that as more SUPAPolicyTerm subclasses are added in the future, the SUPAPolicyStatement object is not affected.

6.7. The Abstract Class"SUPAEvent"

THIS WILL BE DEFINED IN THE NEXT VERSION OF THIS DOCUMENT.

6.8. The Abstract Class"SUPACondition"

THIS WILL BE DEFINED IN THE NEXT VERSION OF THIS DOCUMENT.

6.9. The Abstract Class"SUPAAction"

THIS WILL BE DEFINED IN THE NEXT VERSION OF THIS DOCUMENT.

7. SUPA Logic Statement Information Model

This section defines the classes, attributes, and relationships of the SUPA Logic Statement Information Model (SLSIM).

7.1. Overview

A Goal policy rule (also called a declarative policy rule, or an intent-based policy rule) is a declarative statement that defines what the policy should do, but not how to implement the policy. In this draft, such rules are called SUPA Logic Statements.

This Section, and the following Sections, will be finished in the next version of this document.

7.2. Constructing a SUPAPLStatement

7.3. Working With SUPAPLStatements

7.4. The Abstract Class "SUPALogicClause"

7.5. The Abstract Class "SUPAPLStatement"

7.5.1. SUPAPLStatement Attributes

7.5.2. SUPAPLStatement Relationships

7.5.3. SUPAPLStatement Subclasses

7.5.3.1. The Concrete Class "SUPAArgument"

7.5.3.2. The Concrete Class "SUPAPLPremise"

7.5.3.3. The Concrete Class "SUPAPLConclusion"

This will be defined in the next version of this document.

This document has no actions for IANA.

11. Acknowledgments

This document has benefited from reviews, suggestions, comments and proposed text provided by the following members, listed in alphabetical order: Bob Natale, Liu (Will) Shucheng, Marie-Jose Montpetit.

12. References

This section defines normative and informative references for this document.

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3060] Moore, B., Ellesson, E., Strassner, J., Westerinen, A., "Policy Core Information Model -- Version 1 Specification", RFC 3060, February 2001

[RFC3198] Westerinen, A., Schnizlein, J., Strassner, J., Scherling, M., Quinn, B., Herzog, S., Huynh, A., Carlson, M., Perry, J., Waldbusser, S., "Terminology for Policy-Based Management", RFC 3198, November, 2001

[RFC3460] Moore, B., ed., "Policy Core Information Model (PCIM) Extensions, RFC 3460, January 2003

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

[RFC6021] Schoenwaelder, J., "Common YANG Data Types", RFC 6021, October 2010.

12.2. Informative References

[1] Strassner, J., "Policy-Based Network Management", Morgan Kaufman, ISBN 978-1558608597, Sep 2003

[2] Strassner, J., ed., "The DEN-ng Information Model", add stable URI

[3] Riehle, D., "Composite Design Patterns", Proceedings of the 1997 Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA '97). ACM Press, 1997. Page 218-228

[4] DMTF, CIM Schema, v2.43, http://dmtf.org/standards/cim/cim_schema_v2440

[5] Strassner, J., ed., "ZOOM Policy Architecture and Information Model Snapshot", TR245, part of the TM Forum ZOOM project, October 26, 2014

[6] TM Forum, "Information Framework (SID), GB922 and associated Addenda, v14.5, https://www.tmforum.org/information-framework-sid/

Authors' Addresses

John Strassner
Huawei Technologies
2330 Central Expressway
Santa Clara, CA 95138 USA
Email: john.sc.strassner@huawei.com