Modeling the "Multipolicy Machine"

D. Elliott Bell*

D. Bell, Ltd.

Abstract

A method of treating several unspecified policies is presented. Precise notions of policy combination, policy conflict, conflict resolution, and policy precedence are introduced. Necessary and sufficient conditions for policies to be combined without conflict are established.

1 Introduction

The work reported here is part of Phase II of the "Multipolicy Machine" project, a Small Business Innovative Research contract addressing multiple-policy interactions. Phase I focused on the identification and initial treatment of multipolicy-machine topics. Phase II includes detailed design of the multipolicy machine; collaboration with a commercial vendor; construction of a formal model of the "multipolicy machine"; and the design and implementation of a prototype illustrating the multipolicy machine solutions. This paper reports on the modeling task of Phase II.

Phase I of the Multipolicy-Machine project identified a wide range of pertinent topics across all levels of abstraction, from organizational considerations of policy matters to implementation design options. (See [HOS92a] and [HOS92b].) Since the modeling task focuses on a conceptual level of discourse appropriate for system evaluation at the higher trust levels of the *Trusted Computing System Evaluation Criteria* [TCSEC85], part of the modeling task was to divide the Phase I results into those that were, and were not, properly addressed in a formal security policy model. An interesting aspect of the effort is that the modeling of the "Multipolicy Machine" addresses the interactions between *unspecified* policies, rather than the conceptualization of a *specific* set of policies. Hence, two initial steps in the modeling task were to determine the topics for consideration within the modeling study and establishing a conceptual framework for addressing unspecified policies.

Section 2 below addresses the Phase I topics that were identified for modeling treatment. Section 3 presents the conceptual framework used for modeling the "Multipolicy Machine"; the specific model artifacts introduced to address the multipolicy-machine topics; and the general results achieved. Section 4 summarizes the work accomplished to date and delineates possible future work.

2 Topics for Inclusion

The first step in picking topics for conceptual modeling was to determine the appropriate level of "policy". Four levels of "policy" were used: organizational, conceptual, abstract-design, and implemented-design.

Organizational policies address the full range of enterprises, people, information, and resources. The policies are narrative, intended for human readers and may be layered. A division of a corporation, for example, is subject to corporate Policies and Procedures, but also impose its own additional (or "implementing") policies. It is also subject to local occupancy codes, state unemployment insurance regulations, and federal occupational safety standards. All these policies are termed "organizational".

A conceptual policy is a representation of part of an organizational policy in abstract, conceptual terms. This policy is typically easy to relate to the portion of the organizational policy it is intended to represent. Careful translation of an organizational policy into conceptual terms is the first essential step in conceptual analysis.

A design that builds on conceptual modeling itself translates an external policy (here, the conceptual policy) into its own terms. Part of the design process is tracing the organizational policy requirements to the

 Research performed under Subcontract DSI-4300-B of SBIR contract F19628-93-C-0022.

conceptual, through the abstract-design policy to a final implemented-design policy.

The sieve that was used to select multipolicy-machine topics for modeling treatment was the limitation to conceptual policy concerns. Topics that addressed organizational or design issues were reserved for direct, engineering examination. The conceptual topics and those that were directly supported by conceptual mechanisms were investigated thoroughly.

The conceptual policy topics selected for treatment are as follows:[1] representing more than one policy; metapolicies; representing the combination of policies; elaborating on the notion of "contradictory" or "conflicting" policies; resolving conflict; precedence and order; and policy evolution.

The multipolicy-machine topics are presented in two groups. The fundamental topics (that is, represention of more than one policy, conceptual-level metapolicies, combinations of policies, conflicting policies, and conflict resolution) are addressed directly in the model development below. The derivative topics (precedence, order, and policy evolution) are addressed narratively at the end of the following section.

3 Modeling Results

The mathematical modeling of multiple policies will proceed as follows: *Conceptual Framework* (section 3.1) treats a mathematical representation of a general computing machine; *Direct Topics* (section 3.2) addresses the representation of combinations of policies and the representation of "policy conflict"; and *Derivative Topics* (section 3.3) addresses the notions of policy precedence, ordered policies, and policy evolution.

3.1 Conceptual Framework

3.1.1 Basic Computing Machine

The basis required for modeling multiple policies is that of a basic computing machine. The development here generalizes that in [BLP73, LPB73, BELL73, BLP75] in two ways. The first is allowing more than a single initial state. The second is the more significant: generalizing from deterministic changes of state to (possibly) non-deterministic transitions.

The elements of the basic machine are four: states; requests; decisions; and the machine's state-transition relation.

The notation used for the first three elements of the model are as shown in table 1 below.

Elements	Notation
States	$z \in \mathcal{Z}$
Requests	$R \in \mathcal{R}$
Decisions	$D \in \mathcal{D}$
sequence of requests	$\mathcal{R}^{\mathcal{N}};\ x \in \mathcal{R}^{\mathcal{N}};\ x_t$
sequence of decisions	$\mathcal{D}^{\mathcal{N}};\ x \in \mathcal{D}^{\mathcal{N}};\ y_t$
sequence of states	$\mathcal{Z}^{\mathcal{N}};\ z \in \mathcal{Z}^{\mathcal{N}};\ z_t$

Table 1: Basic Elements

The component sets used here are left unspecified. "Requests" and "Decisions" are the inputs to and outputs from the calculating machine, respectively. All the sequences ($\mathcal{R}^{\mathcal{N}}$, $\mathcal{D}^{\mathcal{N}}$, and $\mathcal{Z}^{\mathcal{N}}$) are defined as mappings from the set of natural numbers $\mathcal{N}$ to the sequenced set (that is, to $\mathcal{R}$, $\mathcal{D}$, and $\mathcal{Z}$).

These basic sets express the machine statically. The state transition relation determines the dynamic nature of the machine.

Definition 1 State Transition Relation: *A state transition relation $\mathcal{W}$ is a subset of the Cartesian product* $\mathcal{R} \times \mathcal{Z} \times \mathcal{D} \times \mathcal{Z} : \mathcal{W} \subseteq \mathcal{R} \times \mathcal{Z} \times \mathcal{D} \times \mathcal{Z}$.

$\mathcal{W}$ identifies "allowable changes of state" as follows. If, while in state z_{t-1}, the machine receives the input x_t, then the output of the machine, y_t, and the next state, z_t, must appear with (x_t, z_{t-1}) in $\mathcal{W}$: $(x_t, z_{t-1}, y_t, z_t) \in \mathcal{W}$.

Definition 2 *The basic calculating machine is the system* $\Sigma\Big(\mathcal{R}, \mathcal{D}, \mathcal{Z}, \mathcal{W}, \mathcal{Z}_0\Big)$, *where* $z_0 \in \mathcal{Z}_0$ *and*

$$(x, y, z) \in \Sigma\Big(\mathcal{R}, \mathcal{D}, \mathcal{Z}, \mathcal{W}, \mathcal{Z}_0\Big)$$

$$\iff \forall t \in \mathcal{N}, (x_t, z_{t-1}, y_t, z_t) \in \mathcal{W}.$$

Note that in contrast to [BLP75], the indeterminate initial state is replaced by an indeterminate set of initial states: $\mathcal{Z}_0 \subseteq \mathcal{Z}$. Note further that $\mathcal{W}$ is not required to be a function. More than one result is possible for any $(x_t, z_{t-1}) \in \mathcal{R} \times \mathcal{Z}$.

3.1.2 Representation of "Policy"

A "policy" is defined here as a "value" on a computation of the basic machine. "A policy makes value judgments" is used synonymously with "a policy associates a value".

[1] For additional information about the selection process and results, see [BELL94, pp. 10–17].

Definition 3 *A policy is a function*

$$\Pi : (\mathcal{R}^{\mathcal{N}}, \mathcal{D}^{\mathcal{N}}, \mathcal{Z}^{\mathcal{N}}) \rightarrow \mathcal{V}.$$

The perspective taken herein is slightly different from most security models in that its focus is the entire computation (x, y, z) rather than the individual transitions $z_{t-1} \rightarrow z_t$.[2] Nevertheless, a traditional goal of most security modeling has been to identify local properties of or restrictions on the underlying machine that ensure properties about all possible computations.[3] Hence not only is this perspective appropriate for dealing with very abstract, non-specific conceptual-policies, but it also has sufficient precedent in the literature.

One can view the security modeling of computing resources as a task of determining what additional constraints that one would like to add to an already-defined machine (that doesn't meet one's security needs) in order to emulate a related, more restricted machine that *does* meet one's needs.

3.1.3 Simple-Security Example

The familiar "simple-security property" [BLP75] can be used to illustrate the notion of "policy" as it is defined here. "Simple-security" for the system Σ is defined in terms of states, as follows:

Σ is SS-Secure
iff $\forall (x, y, z) \in \Sigma \; \forall t \in \mathcal{N}$ (z_t is ss-secure.)

Equivalently, one can define Π_{SS} in the following fashion. Let π_3 denote the projection onto the third factor and ran, the range of a function. For any function "op" mapping $V_{ss}^{\mathcal{N}}$ to $\mathcal{V}$, define

$$\Pi_{SS} \triangleq op \circ ran \circ \sigma_{ss}^{\mathcal{N}} \circ \pi_3.$$

With the stated-based policy $\sigma_{ss} : \mathcal{Z} \rightarrow temp$ and the value sets $val = \{may, cannot\}$ and $VAL = \{\text{BAD}, \text{GOOD}\}$, define the function η as follows:

$$\eta(k) \quad = \quad \begin{cases} \text{GOOD} & if \quad K = \{may\} \\ \text{BAD} & if \quad cannot \in K. \end{cases}$$

Simple-security can then be pictured as in figure 1.

The simple-security example generalizes in a direct fashion to discretionary-security and *-property-security found in [BLP75].[4]

[2]Non-interference models are an exception to this generalization.

[3]The Basic Security Theorem [BLP73] and several unwinding theorems (such as [GOME84]) are examples.

[4]Although ss-security is state-based (as are ds-security and *-property-security), not all policies are. Non-interference relies on "purging" all test computations to establish "interference". This sort of policy is a direct $\Sigma\Big(\mathcal{R}, \mathcal{D}, \mathcal{Z}, \mathcal{W}, \mathcal{Z}_0\Big) \rightarrow \mathcal{V}$ function that cannot be factored through $\mathcal{Z}$.

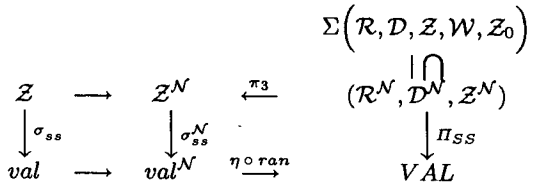

Figure 1: Simple-Security Example

3.2 Direct Topics

This section addresses some of the multipolicy-machine topics directly within the modeling framework. The topics addressed are combinations of policies (using policy combiners) and policy conflict (using policy attenuations, non-conflicting policies, and policy combiners that resolve conflict).

3.2.1 Policy Combinations

With a policy defined as a mapping that associates a value with every computation, it is obvious that a multiple policy situation is exactly the existence of several value judgments on a single machine. The interest lies in constructing descriptive mechanisms to address putting several policy judgments together. At the conceptual level that question becomes:

> given a set of policies $\{\Pi_{A_1}, \ldots, \Pi_{A_n}\}$, how does one determine the joint "value" judgment of the entire set?

The basic notion is that of a "policy combination" or "policy combiner". A policy combiner for policies Π_A and Π_B is a mapping from the combination of value sets V_A and V_B to a single value set $\mathcal{V}$.

Definition 4 *For Π_A and Π_B, the combination of Π_A and Π_B is a function $C_{A,B}$ that maps $V_A \times V_B$ to $\mathcal{V}$.*

A policy combiner associates with every calculation in $\Sigma\Big(\mathcal{R}, \mathcal{D}, \mathcal{Z}, \mathcal{W}, \mathcal{Z}_0\Big)$ a value judgment in $\mathcal{V}$ by combining the values assigned by Π_A and Π_B using the mapping $C_{A,B}$ as shown in figure 2.

The composition $C_{A,B} \circ (\Pi_A \times \Pi_B)$ will denoted by $C_{A,B}^{\circ} : \Sigma\Big(\mathcal{R}, \mathcal{D}, \mathcal{Z}, \mathcal{W}, \mathcal{Z}_0\Big)^2 \rightarrow (V_A \times V_B) \rightarrow \mathcal{V}$.

Definition 5 *For policies $\{\Pi_{A_1} \ldots \Pi_{A_n}\}$, their combination is a function*

$$C_{A_1, \ldots, A_n} : V_{A_1} \times \cdots \times V_{A_n} \rightarrow \mathcal{V}.$$

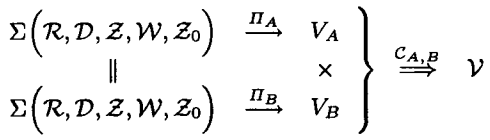

Figure 2: Two-Policy Combiner

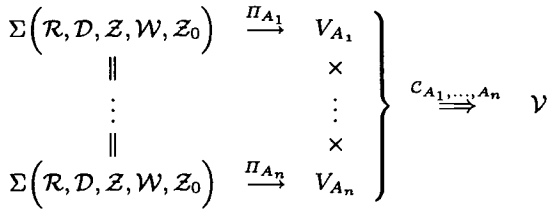

Figure 3: N-Policy Combiner

A general policy combiner associates a value as in figure 3. $C^{\circ}_{A_1,\ldots,A_n}$ will be used to denote the composition $C_{A_1,\ldots,A_n} \circ (\Pi_{A_1} \times \cdots \times \Pi_{A_n})$.

The n-policy combiner provides the descriptive mechanism to combine policies. The composition $C_{A_1,\ldots,A_n}$ captures the combining aspect while $C^{\circ}_{A_1,\ldots,A_n}$ captures the full mapping from the elements of the system to the set of values $\mathcal{V}$.[5]

For the remainder of this development, it will be assumed that $V_A \equiv \mathcal{V}$ for every policy Π_A and that $C_{A,B}$ is of a very simple form, such as $\vee$ ("join" or "OR") or $\wedge$ ("meet" or "AND").

3.2.2 Conflict

The notion of conflict between policies (at the organizational level) leads to the modeling concepts of policy attenuation and conflict-resolving policy combiners.

A policy Π_A assigns a single value to every computation in the system. A policy attenuation for Π_A specifies a set of value judgments that are acceptable to Π_A.[6] The value $\Pi_A(x,y,z)$ is always included within the policy attenuation set $\alpha_A(\Pi_A(x,y,z))$. In the usual case, all the value sets are the same ordered set $(\mathcal{V}, \geq)$ and attenuations are of the form "anything below the policy value" or "anything above the policy value". An "anything below" attenuation is termed a "min-attenuation"; an "anything above" attenuation, a "max-attenuation".

Definition 6 *A* policy attenuation *is a function*

$$\alpha : \mathcal{V} \rightarrow \mathcal{P}(\mathcal{V}),$$

where $v \in \alpha(v) \quad \forall v \in \mathcal{V}$.

For a policy Π_A, an associated policy attenuation will be denoted α_A. The set of values $\alpha(v)$ represent "non-conflicting" policy values with respect to the value v.

For example, consider the simple-security- and discretionary-security-policies, Π_{SS} and Π_{DS}, with policy values $V_{SS} = V_{DS} = \mathcal{V}$, where $\mathcal{V} = \{ \mathit{must}, \mathit{may}, \mathit{cannot} \}$ and the value set is transitively ordered by $\mathit{must} > \mathit{may} > \mathit{cannot}$. Both Π_{SS} and Π_{DS} assign the value *cannot* for an unacceptable calculation and *may* for a calculation that is not unacceptable. The implicit understanding is that if Π_{SS} (respectively, Π_{DS}) judges a calculation *may*, a combined judgment of *cannot* would not be considered seditious. Hence, a value judgment of *may* carries the connotation that *may* is preferred, but that *cannot* would be tolerated. Similarly, a value judgment of *cannot* not only prefers *cannot*, it will not tolerate any value above *cannot*. Thus the policy attenuations α_{SS} and α_{DS} are represented as follows:

$$\alpha_{SS}(v) = \begin{cases} \{ \mathit{may}, \mathit{cannot} \} & \text{if } v = \mathit{may} \\ \{ \mathit{cannot} \} & \text{if } v = \mathit{cannot}. \end{cases}$$

and

$$\alpha_{DS}(v) \equiv \alpha_{SS}.$$

As a second example, consider a public health policy Π_{PH} and a privacy policy Π_{PRIV}, both with the same value set $\{ \mathit{must}, \mathit{may}, \mathit{cannot} \}$. The policy Π_{PRIV} assigns values *may* and *cannot*, in the same way as Π_{SS} and Π_{DS} do. The public health policy Π_{PH}, on the other hand, assigns the values *must* and *may*. The value judgment *must* relates to information that is vital to countering severe community-threat hazards such as a tuberculosis epidemic. The value judgment *must* therefore connotes a policy opinion that certain information (or certain calculations or certain actions) cannot be held away from public scrutiny. Thus, the policy attenuation α_{PRIV} is equivalent to α_{SS} and α_{DS}, while the policy attenuation α_{PH} is represented as follows:

$$\alpha_{PH}(v) = \begin{cases} \{ \mathit{must}, \mathit{may} \} & \text{if } v = \mathit{may} \\ \{ \mathit{must} \} & \text{if } v = \mathit{must}. \end{cases}$$

One policy accepts values at or above its preferred judgment (the policy Π_{PH}), while the other accepts values at or below its preferred judgment (the policy Π_{PRIV}).

Policy attenuations make possible the definition of policy conflict. The concept of conflict is approached double

[5] Note that $C^{\circ}_{A_1,\ldots,A_n}$ satisfies the definition of "policy".

[6] An "attenuation" is a "weakening". A policy attenuation is a set of acceptable "weaker" value judgments associated with the pure policy value judgment Π_A.

negatively. Policy Π_B is said to non-conflict with policy Π_A provided its value judgments are always in the set of acceptable weakened values delineated by the policy attenuation α_A. That is, policy Π_B non-conflicts with Π_A provided that the value B would assign is acceptable to policy A. Policies conflict if neither one is non-conflicting with the other.

Definition 7 *Let policies Π_A and Π_B have attenuations α_A and α_B, respectively. Policy Π_B is said to be* non-conflicting *with Π_A provided*

$$\forall(x,y,z) \in \Sigma\Big(\mathcal{R},\mathcal{D},\mathcal{Z},\mathcal{W},\mathcal{Z}_0\Big),$$
$$\Pi_B(x,y,z) \in \alpha_A\big(\Pi_A(x,y,z)\big)$$

Otherwise, Π_B conflicts *with Π_A.*

If Π_B conflicts with Π_A, there is an (x,y,z) such that

$$\Pi_B(x,y,z) \notin \alpha_A\big(\Pi_A(x,y,z)\big).$$

We call such an (x,y,z) an illustrative conflict of Π_B with Π_A. An element

$$(x,y,z) \in \Sigma\Big(\mathcal{R},\mathcal{D},\mathcal{Z},\mathcal{W},\mathcal{Z}_0\Big)$$

is a conflict between Π_A and Π_B if it is either an illustrative conflict of Π_A with Π_B or an illustrative conflict of Π_B with Π_A. Call Π_A and Π_B mutually non-conflicting if there is no conflict (x,y,z) between Π_A and Π_B.

More generally, we say that (x,y,z) is a conflict among the set of policies

$$\{A_1,\ldots,A_n\}$$

if there are $1 \le i,j \le n$ such that (x,y,z) is a conflict between Π_{A_i} and Π_{A_j}. The set $\{A_1,\ldots,A_n\}$ is called mutually non-conflicting if there is no conflict among the set of policies.

Definition 8 *For an ordered set of values $(\mathcal{V},\preceq)$, a* min-attenuation *is policy attenuation α provided*

$$\alpha(v) = \{v' : v \succeq v', v' \in \mathcal{V}\}.$$

Dually, α is a max-attenuation *provided*

$$\alpha(v) = \{v' : v \preceq v', v' \in \mathcal{V}\}.$$

If $(V,\succeq)$ is a boolean lattice $(V,\vee,\wedge)$, then the elements "less than" an element v are the interval from O to v, and the elements "greater than" an element v are the interval from v to I. In this case, min-attenuation will be called MEET-ATTENUATION, and max-attenuation, JOIN-ATTENUATION. The lattice $(V,\geq)$, a linearly ordered set, is a common case.[7]

A policy combiner $C_{A,B}$ is said to "resolve the conflicts" between policies Π_A and Π_B provided it is a policy combiner that non-conflicts with both Π_A and Π_B.

[7]S. Ovchinnikov has noted the similarity between these lattice value-sets and Goguen's L-fuzzy sets [GOGU67].

Definition 9 *The policy combiner $C_{A,B}$* RESOLVES conflicts *between Π_A and Π_B (relative to α_A and α_B) provided $C^\circ_{A,B}$ non-conflicts with both Π_A and Π_B. $C^\circ_{A_1,\ldots,A_n}$ resolves conflicts amongst the policies $\{\Pi_{A_1},\ldots,\Pi_{A_n}\}$ provided it non-conflicts with Π_{A_i} for $1 \le i \le n$.*

Theorem 1 *For policies $\{\Pi_{A_1},\ldots,\Pi_{A_n}\}$ with associated min-attenuations $\{\alpha_{A_1},\ldots,\alpha_{A_n}\}$, there exists a policy combiner $C_{A_1,\ldots,A_n}$ that is non-conflicting with every Π_{A_i}.*

Proof: Let

$$\hat{C}_{A_1,\ldots,A_n}(x,y,z) \triangleq \min\{\Pi_{A_i}(x,y,z) : 1 \le i \le n\}.$$

Because α_{A_i} is a min-attenuation and

$$\hat{C}^\circ_{A_1,\ldots,A_n}(x,y,z) \le \Pi_{A_i}(x,y,z),$$

$\hat{C}^\circ_{A_1,\ldots,A_n}$ non-conflicts with Π_{A_i}.[8]

The dual result holds for max-attenuations. Similarly for meet-attenuation and join-attenuations (substituting $\{\wedge,\preceq\}$ and $\{\vee,\succeq\}$, respectively, for the pair $\{inf,\leq\}$).

Theorem 2 *Let Π_A and Π_B be policies. Let α_A be a min-attenuation and α_B, a max-attenuation. There is a policy combiner $C_{A,B}$ that resolves conflicts between Π_A and Π_B iff $\Pi_A \ge \Pi_B$.*

Proof: If $\Pi_A \ge \Pi_B$, $C_{A,B} \equiv \pi_A$ (the projection onto Π_A) is such a policy combiner. If $C_{A,B}$ resolves conflicts between Π_A and Π_B, then $\Pi_A \ge C^\circ_{A,B} \ge \Pi_B$, so that $\Pi_A \ge \Pi_B$ by transitivity.

Corollary 3 *Let $\mathcal{V}$ be linearly ordered and let policy Π_B have a max-attenuation α_B. There is no policy combiner $C_{A,B}$ that resolves conflicts between Π_A and Π_B if*

$$ran\,\Pi_A - ran\,\Pi_B \neq \emptyset.$$

Proof: Let $v' = min\big(ran\,\Pi_A - ran\,\Pi_B\big)$. Now,

$$v'' \in ran\,\Pi_B \quad \text{and } v'' \le v' \qquad \Rightarrow \qquad v' \in \Pi_B$$

because α_B is a max-attenuation. Since $v' \notin ran\,\Pi_B$,

$$v'' > v'\,\forall v'' \in ran\,\Pi_B.$$

Hence,

$$\forall(x',y',z') \in \Pi_A^{-1}(v')$$
$$\Pi_A(x',y',z') = v' < \Pi_B(x',y',z').$$

$\Pi_A \not\geq \Pi_B$ and the result follows from Theorem 2.

Corollary 4 *Let $\mathcal{V}$ be linearly ordered and let policy Π_A have a min-attenuation α_A. There is no policy combiner $C_{A,B}$ that resolves conflicts between Π_A and Π_B if*

$$ran\,\Pi_B - ran\,\Pi_A \neq \emptyset.$$

Proof: This corollary is dual to corollary 3.

[8]Note that any policy combiner less than $\hat{C}_{A_1,\ldots,A_n}$ also resolves conflicts among $\{\Pi_{A_1},\ldots,\Pi_{A_n}\}$.

3.3 Derivative Topics

The remaining multipolicy-machine modeling topics are addressed in this section: policy precedence, policy order, and policy evolution.

3.3.1 Policy Precedence

The notional idea of "precedence" is captured via a modeling function that is called a "policy precedence". A policy precedence associates a value in an ordered set T with every computation in the system $\Sigma\Big(\mathcal{R}, \mathcal{D}, \mathcal{Z}, \mathcal{W}, \mathcal{Z}_0\Big)$.

Definition 10 *A (policy)* precedence *is a function*

$$\varrho : \Sigma\Big(\mathcal{R}, \mathcal{D}, \mathcal{Z}, \mathcal{W}, \mathcal{Z}_0\Big) \rightarrow T,$$

where T is a set ordered by $\geq$. The precedence associated with policy Π_A is denoted ϱ_A.

Two distinct forms of precedence-respecting policy combiners are introduced. The absolute-precedence policy combiner totally ignores the lower-precedence-level policy.[9] The second type of precedence is the conflict-precedence policy combiner.

An absolute-precedence policy combiner assigns values not-in-conflict with the high-precedence policy, totally disregarding the low-precedence policy. A conflict-precedence policy combiner combines two policies without precedence when possible. When there *is* conflict between the policies, it resolves the conflict in favor of the higher-precedence policy.

Definition 11 *An* absolute-precedence policy combiner *is a policy combiner $C_{A,B}$ that satisfies the following symmetric conditions:*

$\varrho_A(x,y,z) < \varrho_B(x,y,z)$
$\Rightarrow (x,y,z)$, not a conflict between $\Pi_B \,\&\, C^{\circ}_{A,B}$

$\varrho_B(x,y,z) < \varrho_A(x,y,z)$
$\Rightarrow (x,y,z)$, not a conflict between $\Pi_A \,\&\, C^{\circ}_{A,B}$

Conflict-precedence is defined in a fashion similar to that of absolute-precedence, with the addition of a conflict condition.

Definition 12 *A* conflict-precedence policy combiner *is a policy combiner $C_{A,B}$ that satisfies the following symmetric conditions:*

(x,y,z), a conflict between $\Pi_A \,\&\, \Pi_B$
$\&\ \varrho_B(x,y,z) < \varrho_A(x,y,z)$
$\Rightarrow (x,y,z)$, not a conflict between $\Pi_A \,\&\, C^{\circ}_{A,B}$

(x,y,z), a conflict between $\Pi_A \,\&\, \Pi_B$
$\&\ \varrho_A(x,y,z) < \varrho_B(x,y,z)$
$\Rightarrow (x,y,z)$, not a conflict between $\Pi_B \,\&\, C^{\circ}_{A,B}$

Ss-security and ds-security neatly illustrate the difference between conflict- and absolute-precedence. By theorem 1, there is a policy combiner $C^{\circ}_{A,B}$ for Π_{SS} and Π_{DS} that resolves their conflicts. For any precedences ϱ_{SS} and ϱ_{DS}, $C^{\circ}_{A,B}$ is a conflict-precedence policy combiner.

On the other hand, for $\varrho_{SS} < \varrho_{DS}$ (or the reverse), any absolute precedence policy combiner will ignore either the values of Π_{SS} or those of Π_{DS}. Moreover, the fact that absolute-precedence policy combiners ignore some policy value judgments allows policy variation through changing either the policy combiner or the precedence functions ϱ_{A_i}.

3.3.2 Order

The idea of "order of execution" of policies has also been raised as an aspect of metapolicy. This notion is similar to a common implementation of access control lists (ACLs) that calculates the effect of a set of ACL entries by finding the "first" ACL entry that addresses a pending access request. To illustrate, suppose Jones requests *read* access to file *foo* where the ACL for *foo* is

UserID	GroupID	Modes
Smith		rw
	Verwirrt	null
Wesson		rw
Crisco		r
	Darwin	rew
Jones		rew

Jones will be granted *read* access, unless Jones is a member of the group *Verwirrt: Verwirrt*'s *null* access would be calculated for relevancy before Jones's *rew* access is calculated. The result is no access at all. Suppose, on the other hand, *foo*'s ACL were rearranged as follows:

UserID	GroupID	Modes
Smith		rw
Jones		rew
	Verwirrt	null
Wesson		rw
Crisco		r
	Darwin	rew

Then the ACL entry granting Jones *rew* access would be evaluated before the *null* access ACL entry for *Verwirrt* was reached.[10]

It has been noted frequently that an order-based determination of a joint ACL decision is fraught with confusion

[9] This discussion is phrased in terms of two policies; generalization to n policies is straightforward.

[10] While this example is absolute-precedence, the point of the example is the order, not the type of precedence.

for the user of the system who is trying to grant or to revoke access privilege. One is required to decide not only on the ACL entry to make, but also to check the entire ACL for unintended side-effects.

Fortunately, "order" is just a special case of precedence. Specifically, prepending the ordinal to each ACL entry that represents its place in the list allows a sequential process to be represented as a function. The initial ACL above would then become

Ord	UserID	GroupID	Modes
1	Smith		rw
2		Verwirrt	null
3	Wesson		rw
4	Crisco		r
5		Darwin	rew
6	Jones		rew

Rearrangement of these ACL entries would not, in this implementation, alter the result of the calculation:

Ord	UserID	GroupID	Modes
4	Crisco		r
1	Smith		rw
6	Jones		rew
3	Wesson		rw
5		Darwin	rew
2		Verwirrt	null

Both ordered ACLs produce the same joint ACL decision.

Similarly, an "order" on policies is equivalent to having a precedence function identical to the order. Consider an ordered list of policies, $(\Pi_{A_1}, \ldots, \Pi_{A_n})$. Since the order is itself injective onto the set of policies, the inverse of the order is a mapping from Π_{A_i} to the set of integers $\{1, 2, \ldots, n\}$. Defining

$$\varrho_{A_i}(x, y, z) \triangleq i,$$

yields precedence functions for all the policies. Hence, an ordering of policies Π_{A_i} is a special case of precedence, and order reduces to a previously solved problem, namely precedence.

3.3.3 Policy Evolution

Policy evolution involves the idea of a policy or a set of policies changing. One kind of change would be the total replacement of policy Π_A by a second policy Π_B. A second kind of change would be a slight alteration of policy Π_A. A third change would be the addition or deletion of a policy.

At the conceptual level, a "slight alteration of policy" is not really distinct from the replacement of one policy by another: swapping out policy Π_A for either a "slight alteration" or a radically different policy is a change of policies in any case. Thus, there remains the replacement of a policy, the addition of a policy, and the deletion of a policy.

At the conceptual level, an alteration of precedence could cause an absolute-precedence policy combiner to alter, add or delete a policy. Consider the three policies Π_A, Π_B, and Π_C with associated precedence functions $\{\varrho_A, \varrho_B, \varrho_C\}$. Assume that ϱ_A and ϱ_B assign the value 1 to all calculations and that ϱ_C assigns 0 to all calculations. An absolute-precedence policy combiner on $\{\Pi_A, \Pi_B, \Pi_C\}$ will ignore Π_C. Thus, that policy combiner makes it appear that there are only two policies in force. Altering ϱ_A to be identically 0 makes the policy combiner ignore Π_A also. That alteration was effectively a policy deletion. Similarly, altering ϱ_C to be identically 1 puts Π_C into force. That action "adds" policy Π_C. The combination of those two changes is effectively the "replacement" of Π_A by Π_C.[11]

"Policy evolution" can thus be viewed as a higher-level perspective on changes to policy precedence values. The ability to add and to delete policies produces the ability to "change" policies, either slightly different policies or significantly different policies. Policy evolution is a special case of varying policy precedence.

4 Summary and Future Work

This modeling task has resulted in capturing the appropriate Multipolicy-Machine topics in a conceptual framework. In addition, two general results have been established, one that a set of min-attenuated (resp. max-attenuated) policies have a policy combiner that resolves conflicts and the other that establishes necessary and sufficient conditions for a min-attenuated policy Π_A and max-attenuated policy Π_B to have a non-conflicting policy combiner.

This essential step in the conceptual analysis of the Multipolicy-Machine has produced descriptive machinery that generalizes the traditional security-policy-modeling perspective in allowing non-determinacy, a set of initial starting states, and sets of unspecified policies. Further, an initial set of general results has been established. General utility of these results will be able to be assessed when these preliminary tools are put to use in the derivation of specific solutions for a multipolicy-machine implementation.

References

[BELL94] D. E. Bell, "Multipolicy Machine Model", DBL Item 94–019, D. Bell, Ltd., Reston, VA, 6 February 1994.

[11] Clearly the same result could have been achieved by altering the policy combiner itself. There is, however, an elegance about leaving the policy combiner unchanged and altering the (instantaneous) precedence functions.

[BLP73] D. E. Bell and L. J. La Padula, "Secure Computer Systems: Mathematical Foundations", MTR–2547, Vol. I, The MITRE Corporation, Bedford, MA, 1 March 1973. (ESD–TR–73–278–I)

[BELL73] D. E. Bell, "Secure Computer Systems: A Refinement of the Mathematical Model", MTR–2547, Vol. III, The MITRE Corporation, Bedford, MA, December 1973. (ESD–TR–73–278–III)

[BLP75] D. E. Bell and L. J. La Padula, "Secure Computer Systems: Unified Exposition and Multics Interpretation", MTR–2997, The MITRE Corporation, Bedford, MA, July 1975. (ESD–TR–75–306)

[GOGU67] J. A. Goguen, "L-Fuzzy Sets", *J. Math. Anal. Appl.* **18** 145–174.

[GOME84] J. A. Goguen and J. Meseguer, "Unwinding and Inference Control", *Proc.* 1984 IEEE Symp. on Security and Privacy, Oakland, CA, April 29–May 2, 1984, 75–86.

[HOS92a] H. H. Hosmer, "Metapolicies II", *Proc.* 15th NCSC, Baltimore, MD, October 13–16, 1992, 369–379.

[HOS92b] H. H. Hosmer, "The Multipolicy Paradigm", *Proc.* 15th NCSC, Baltimore, MD, October 13–16, 1992, 409–422.

[LPB73] L. J. La Padula and D. Elliott Bell, "Secure Computer Systems: A Mathematical Model", MTR–2547, Vol. II, The MITRE Corporation, Bedford, MA, 31 May 1973. (ESD–TR–73–278–II)

[TCSEC85] *Department of Defense Trusted Computer System Evaluation Criteria,* DoD 5200.28-STD, December 1985.