Policy Management, Economics, and Risk

Edgar H. Sibley, James B. Michael, and Richard L. Wexelblat

Information Systems and Systems
Engineering Department
George Mason University
4400 University Drive
Fairfax, VA 22101 USA
esibley@gmuvax.gmu.edu
jmichael@gmuvax2.gmu.edu

Computer and Software
Engineering Division
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria, VA 22311 USA
rwex@ida.org

Abstract

Interaction between humans and computers is part and parcel of everyday life. Office systems, home and school computers, military systems, aircraft navigation systems all influence our daily lives whether we consider ourselves computer users or not. The mode of interaction, screen, printer, loudspeaker, microphone, keyboard, buttons, pointing device, or what have you implements a set of limitations on the communication--what can be said and how it is said. The way this interaction works implements a policy set by the designers or implementors of the system and the user is willy-nilly forced to work under the policy. But what happens when an exception is required? What happens when an unanticipated condition arises. Is there a way to tell the automatic teller machine, "Forget the whole thing, just give me my card back?" Can the pilot force the autopilot to give up control? If the policy implemented is not flexible, annoyance (with the bank) or disaster (with the aircraft) can occur.

This article begins to look at the way policy might be specified in a manner sufficiently formal that systems implementors might make use of such a specification. It considers the objects that are relevant to such a specification and what operations might be applied to these objects.

Keywords: Information and Corporate Policy, Information Systems Goals, Formalism in Policy, Meta-modelling, Policy in Environments

1.0 Introduction: Policy, Goals, and Risks

Organizations have policy--this is a given. But it is not easy to agree on what policy *is*. Informally, a policy is a set of rules of action that cover some but not all possible situations. It might be said that policies are rules of behavior but in fact, they are not rules at all. They are really statements of goals or desires. There is value to the organization if they are observed, and often a cost to the organization if they are violated. Thus, in promulgation of a

policy, the potential risks of observing it and of violating it must be taken into account. For example, the policy of a software development organization may state that no software mod is released without proper approvals. But what shall a programmer do when the best customer urgently needs that programmer's bug fix and it is midnight with no member of the approval hierarchy to be found? If, within the culture of the organization, policy has the force of law, then the customer (and the company, perhaps) are out of luck. If the company was wise enough to promote a culture in which good judgement is valued, an exception to policy might be appropriate.

One dictionary defines policy as "a definite course of action as expedient or from other considerations." The definition is actually much longer and the words "prudent" and "wise" appear. Because the policies are (often deliberately) vague, most organizations define procedures that implement the policies. Procedures, intelligently written, will state both the rules and the ways to make exceptions to them: no first-class travel unless approved by the Executive VP, for example.

Hertz [Hertz, 1986] says that policies, "...if they are serious, are intended to influence behavior, which implies that desired action-consequences scenarios... are the basis for policy formulation." If we wish policies to influence behavior of those who use computer systems then it will be necessary to include policy in the programs that run on the computers.

Policies can be stated in general terms with various level of detail:

a. The autonomous land vehicle navigation system shall direct the tank unless the (human) commander explicitly overrides it.

b. Corporate travellers shall minimize costs to the organization at all times.

c. No software modification shall be released before written approval of the Release Control Officer.

Policy A is clear; at least one limit of application is stated and an observer could tell if the policy is observed. Policy B is clearly stated as well, but only a dedicated accountant could tell if it was being complied with. Policy C is also clearly stated there is no exception stated. (There might be an overriding consideration, however. Another policy might state that exceptions to any software release policy may be made if required to satisfy a customer.)

The three examples and their discussion exemplify the problems with trying to treat policy formally. Policies range from precise statements of rules (algorithms) to vague statements of intent. Policy sets leave things unsaid; they over and incompatibly specify; they state goals and leave risks unevaluated and unevaluatable. If we are to increase the intimacy of human-machine interaction--and of course we are: the juggernaut is at full speed and no one dare interfere--then we need to be able to state policy in terms that can be understood by the human and at the same time acted upon by machine. Inspired by many in AI who have worked on planning and many in business and economics who have helped us better to understand organizational behavior, we present here preliminary results on a formalization of policy and its implementation by computer.

The essential parts of such an implementation are:

- should be able to provide expert-level outputs to complex data inputs
- be understandable to the policy makers.
- be usable for the purposes of determining the logical and statistical validity of policy inferences.

Thus the system must be able to handle:

- broad policy problems.
- use information that is not always consistent or complete, and manipulate it with symbolic reasoning without following an algorithmic procedure.
- handle cascading policy decisions (many-dimensioned dynamic decision program with an optimum solution).
- permit the testing of alternative higher level policies

Hirschheim and Klein [Hirschheim and Klein, 1989] make some philosophical assumptions about the alternative paradigms of information systems development (i.e., of different IS development approaches). They feel that the behavior of system analysts and other developers is dependent on the assumptions that are explicit and implicit in the particular paradigm being used. There is therefore a need to document assumptions underlying each paradigm, and then choose between alternatives in a systematic way.

This entails a better understanding of the conceptual foundations of beliefs of the developers (in Mumford's sense [Mumford, 1985], their "value set"). It will result in a set of creative solutions using a mix of the paradigms in addition to further refinement and development of these paradigms. The process is to analyze existing methodologies with respect to the paradigms, and explore the nature of ISD.

The context of any system is the Natural Laws: It must be impossible to transcend the overall laws and conditions of normal living or experience: thus people should not be asked to float around in the air (under normal gravity) without real supports, nor should the laws contravene common-sense or good-practice; they might include: don't spend money before it is available.

Several important aspects of law-making and policy setting are investigated here:

- The Organization exists in an Environment that provides Laws. These differ in various places, parts of the world, and areas.
- There is a Risk involved in breaking any Law. This may be treated as an economic cost, depending on the fine exacted and the probability of being caught; this is coupled with the ethical or political cost, not forgetting the personal cost and loss-of-face or reputation in prosecution and subsequent incarceration.
- Although this will not be pursued further here, the risk or cost of breaking a law may be offset by the partial or potential benefits of doing so, seen as short- or long-term profit. Thus the promise of the capture of large amounts of cash or valuables may offset the cost of the risk in being caught and serving a prison sentence. There are times when

there is a large personal or ethical advantage; e.g., when saving the life of a relative or friend against an attacking dog in spite of local laws that normally prohibit the use of force against animals, or attempting hostage release by forcefully attacking terrorists.

As examples:

- International laws, though the highest level are probably the easiest to break, being relatively important only if they are upheld by a major power; in themselves, they are relatively unenforced.
- Group or multi-nation laws, such as those of the EEC, are more important, though sometimes breakable with no loss.
- National, state, county, and local laws and regulations are sometimes hard to enforce, but they are usually prosecuted once detected.

Thus economic aspects generally play an important role (e.g., in trade-off analysis). It is still, however, necessary to address the Multiple Views of the Actors (one will see a different process from another). It is interesting to note that "permission" in an organizational setting may be considered as a deliberate "trap door" or way of allowing a rule or law to be circumvented with impunity.

2.0 An Object-Oriented Approach to Policy Formulation

This section describes some policy objects and the basis for several methods of implementing policy decisions within an organization. It serves to introduce and motivate the problems of embedding policy in procedures and dealing with a multi-tiered policy environment: implications of policy level interactions causing transfer of "rights" either from one part of the organization to another or through procedures that implement policy at different levels. Dobson and McDermid [Dobson and McDermid, 1989a; Dobson and McDermid, 1989b] have laid the groundwork for representing the objects in a system and their relationship to an organization's policies. Their work focuses on security policies.

2.1 Basic Policy Objects

One set of objects that could be important in policy formation and its articulation are found in the triple: {Agent/Action/Data}. However, these are slightly augmented here. The proposed Object types in this field are:

- Policy;
- Player or Agent; and
- Action,

where a Policy Object, is basically a rule, however this may be also be considered as an external or internal law, a goal, or one of the set of objectives of the organization. They are articulated here by some form of inherited properties of objects that control or affect procedure application. In other words, the Policy Objects are a mixture of the External Laws and additional Policies and Procedures, in conjunction with any local Rules. They will ei-

ther be passive, being checked by the actions, or active, initiating or forbidding (i.e., controlling) the actions through triggers on the procedure application.

The Attributes of a Policy Object include either the Rules or Procedures that can be applied to the object, or both. Thus the Operations on the Policy Object may involve:

- Make/Add/Delete Policy
- Change Policy

The Player or Agent Objects are people involved in setting and enforcing policy and in using systems under domination of a policy subset. They play one of the roles of:

- Observer, noting that something is happening;
- Executor, who is starting an action;
- Owner (or administrator), who can destroy an action that is executing;
- Creator, who required the action, but may have given it to an Executor to perform the initiation;
- Customer, who can change the specification of the action.

It should be noted that these may be considered either as attributes of the person or as relationships to other people, and that these may be inherited from "Person Object types." The Player Object types are probably articulated in a semi-hierarchical fashion, as are also the Policy Objects, because, in their various roles, the actors are often in a hierarchy prespecified by the organization. The laws and goals are cumulative, and therefore mainly in a hierarchy. Here, only one level is considered, though there is probably a pure hierarchy of goals to rules.

Action Object instances are mainly initiation of operations on Policy Object instances:

> A set of procedures act only on their prespecified data structures. Actions of the people (in their specified roles) are constrained by the rules. Thus a set of operations or procedures that can be initiated are the operations allowed within a given role and we may consider the process as one in which actions between instances of entities are initiated by sending messages between them.

There are several important two part Relationships between Objects:

- Policy Object to Player or Agent Object. This seems to be the important relationship. Apparently, actors are the only objects that attempt to contravene rules, unless a procedure resulting in a covert channel can be considered a threatening activity with "intent."
- Policy Object to Actions on or of an Object. As in the previous covert activity case, the rules could be contravened by a person who sets up a "false" procedure.
- Action Object to Player or Agent Object. This may be an *inverse* relationship.

We therefore consider policy activities as static or dynamic binding of policy between objects and actions.

The Effect of a Policy is to generate:

A set of constraints in the form of procedures acting only on the specific data structures as initiated by the people (player or agent) roles.

In this paper, we shall not represent the Environment or the Organization as either object types or instances, as they are considered to be the context of the Policy. They observe, or possibly supervise, its actions and are aware of, if not assure, its consequences. Indeed, the causality of the Environment (how it uses products, provides resources, or sets laws) is outside the scope of this research; this must be investigated later.

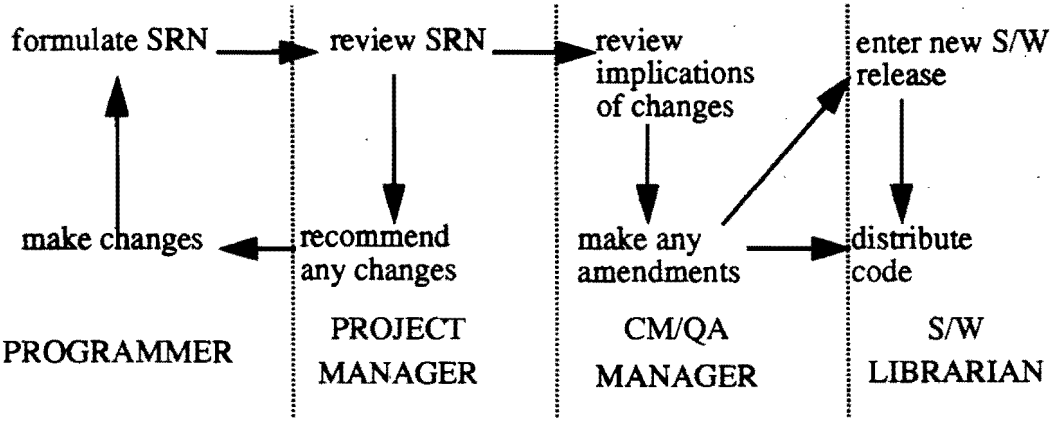
2.2 Motivating Examples

Two examples are given here. The first discusses a multi-tiered policy environment. It introduces the problems of embedding policy in corporate procedures and transferring "rights" from one part of the organization to another. The second deals with a slightly larger case to show the effect of complexity in the process and use of an object-oriented paradigm.

2.2.1 A Software Engineering Environment

In order to demonstrate the problems in embedding policy in corporate procedures, a set of software configuration management policies for Software Release Notice (SRN) activities are modelled. One way to model SRN policies is to use an activity-role chart [Coordination, 1983], as pictured in Figure 1. In an activity-role chart, each role (e.g., programmer) and activity (e.g., formulate SRN) are defined over time (i.e., ordered). Each role is then instantiated. A person can perform one or more roles simultaneously.

FIGURE 1. Activity-Role Chart.

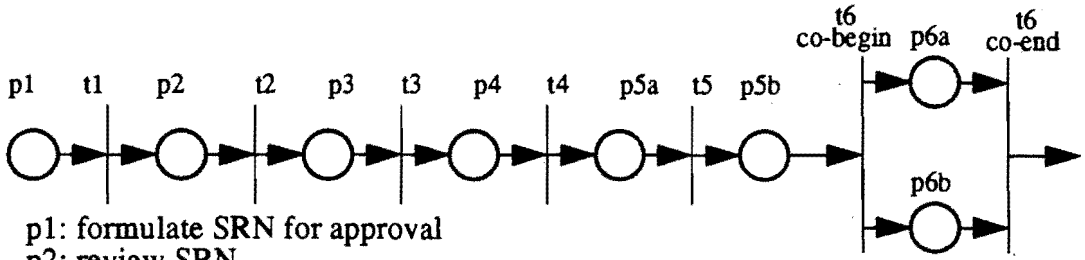


Here, there are two types of policy--explicit and implicit. The *explicit* policies are represented as activities. An example of an *implicit* policy is "before reviewing the implications of software changes, a SRN must be formulated and recommendations for change must be made" (i.e., there are pre-conditions for each activity in the chart that effect an ordering

which is the implied policy, and a set of post-conditions that must be satisfied after the process is executed).

A Petri net [Petri, 1962; Murata, 1989] equivalent of the above activity-role chart is given in Figure 2. The Petri net representation of the SRN process permits the explicit modeling of the "triggering" of activities (e.g., the number of tokens required to move to enable a transition) and concurrent activities. For example, the software librarian activities of entering a new software release and distributing code and manuals must both be completed before moving to the next place in the Petri net. In essence, this branching therefore effects the explicit policy.

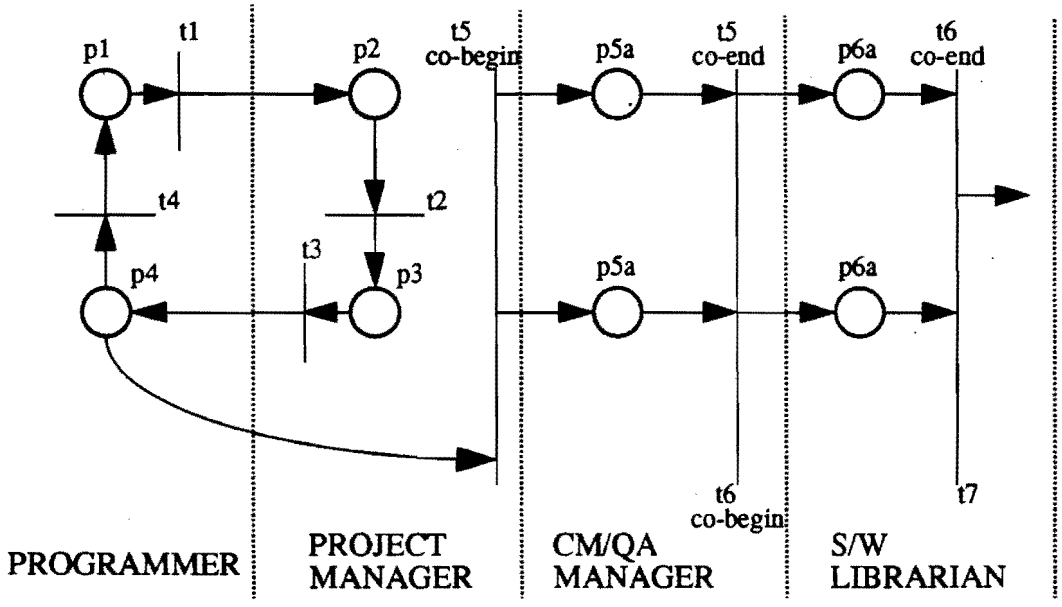
FIGURE 2. Petri Net Representation.



p1: formulate SRN for approval
p2: review SRN
p3: recommend any changes
p4: make changes (if necessary)
p5a: review implication of changes
p5b: make any amendments
p6a: enter new S/W release
p6b: distribute code and manuals

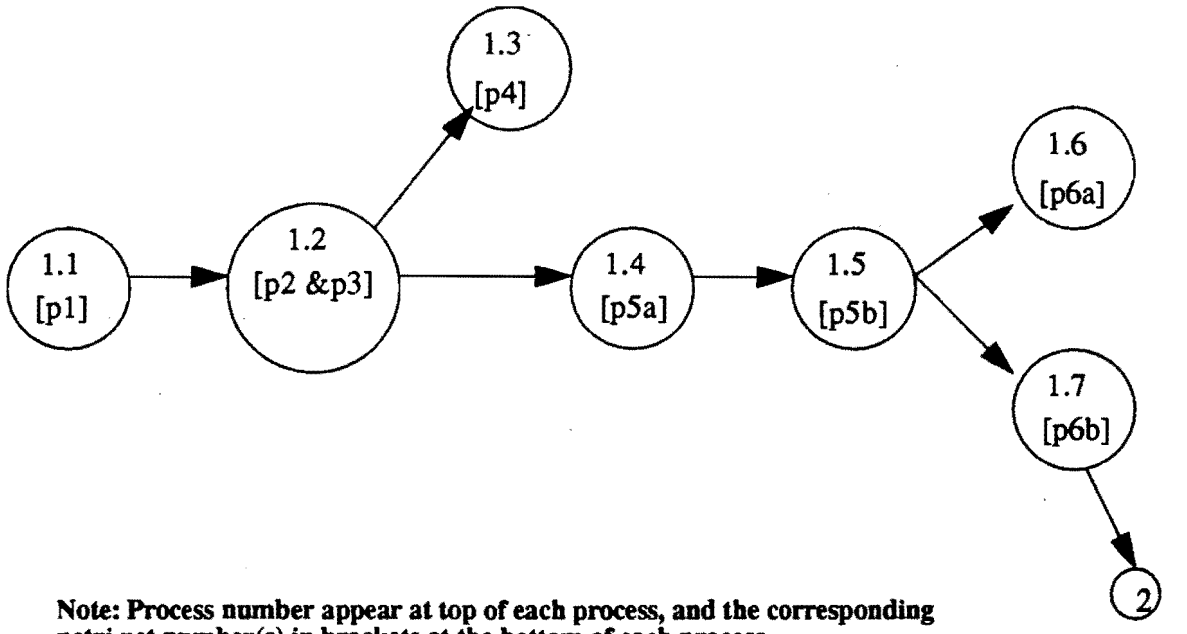
A more expressive graphical notation for representing the SRN policies is that of an activity-role chart with Petri net extensions. This model explicitly captures the pre- and post-conditions of the process.

FIGURE 3. . Activity-Role Chart with Petri Net Extensions.



Another possible representation of the SRN policies is given in Figure 4. This is a two-level data flow diagram(DFD) without data stores or external entities. Each process corresponds to an activity or multiple activities in the activity-role chart (i.e., places in the Petri net diagram). The places these processes represent are given in square brackets.

FIGURE 4. A Two-Level DFD Equivalent



Note: Process number appear at top of each process, and the corresponding petri net number(s) in brackets at the bottom of each process.

The DFD of Figure 5 depicts logical (as opposed to physical) aspects of policies regarding SRNs, such as the data that flows between processes, data stores, and external entities. It can be further extended to reflect control-oriented policies (such as the role or roles that control an activity), resource-oriented activities (e.g., resources required by an activity), and so on. Note that activity-role charts, petri-nets, and data flow diagrams are all object-oriented representations, but each provides a different view of objects. For example, in a DFD, the data flowing from one process to another is equivalent to a message passing between to objects. Only the lowest level DFDs, however, explicitly represent the methods attached to these messages (i.e., process and control specification diagrams).

Our first dilemma is illustrated by asking:

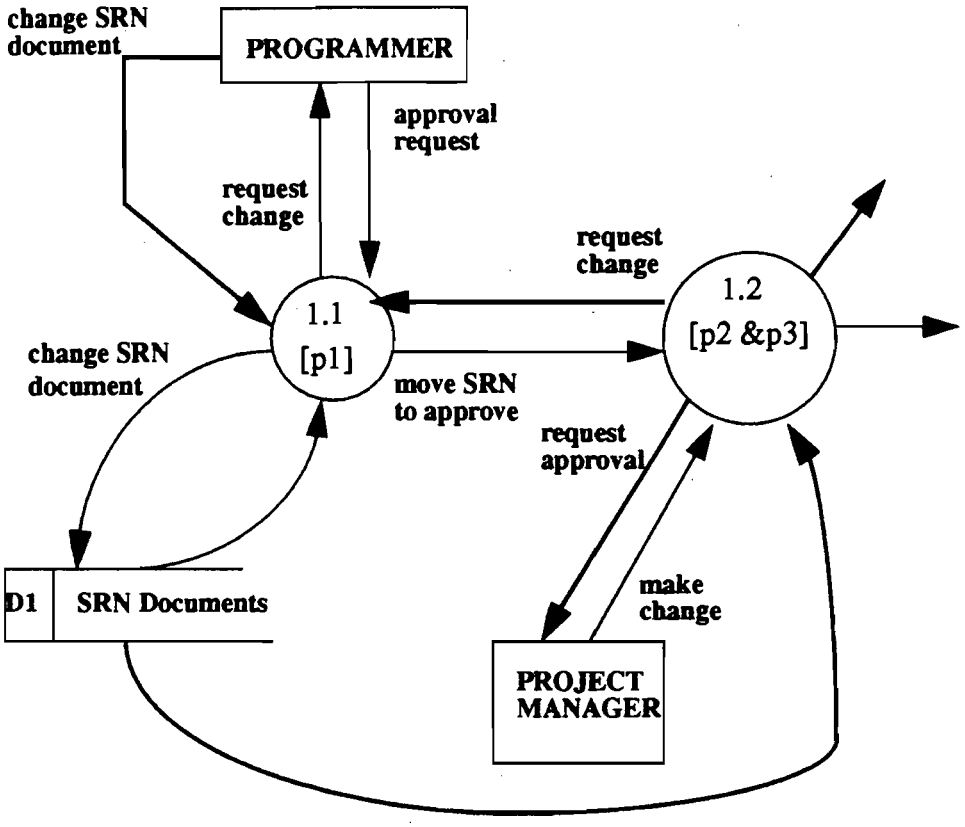
> How do we add the policy requirement that no group or team leader in the SRN process shall manage more than five people?

Of course, in a typical organization today, it is likely that such a statement of procedures or rules has been generalized and is not only applicable to the SRN activity. Thus it may have been stipulated at a higher level in the organization, possibly as a statement that: "No group or team leader in any organizational process shall manage more than fifteen people." This may also (explicitly or implicitly) be understood to have the added clause: "in some cases or locales the number may be further restricted." Then at the SRN or some prior level, there would be articulation to the value "five."

Such an inter-level policy interaction, which leads us to ask:

How may we construct a policy enforcement procedure that it is not embedded in the operating procedures, and hence hard-coded in the programs and algorithms of the MIS?

FIGURE 5. Details for Processes 1.1 and 1.2.



This suggests a management solution similar to "delegation of responsibility." The question: "Who enforces what policy?" is answered by allowing intermediate managers to pass their responsibility down to a operational level; the action associated with the policy is most naturally carried out at this (lower) level. The problem with this "manual" approach is that the managers generally need a "book of procedures and practices" to record the current policy and its level of delegation. It also allows new or replacement managers to find the answers to policy issues (and their rights or required actions) without unduly disturbing their manager (or staff aids).

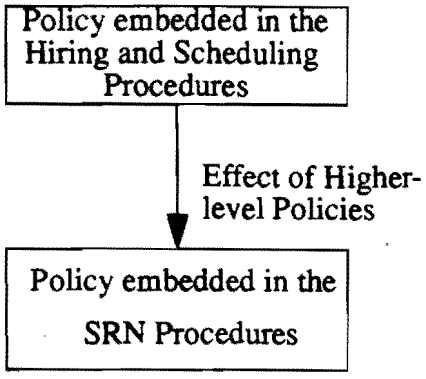
The problem with such a manual procedure is the task of rewriting many parts of the manual when any major or top-level policy changes; this is also true of national laws, etc. It may not be surprising that there are great similarities between these problems and the maintenance of large and complex computer programs.

In the past, the solution has been *ad hoc*, primarily because there was little or no practical automated method: all manual methods tend to be clumsy, slow, and expensive. However, today we have some powerful techniques that introduce formalism and allow theorem proving or goal setting as ways of their resolution. Maybe the solution is not yet elegant,

but it is certainly becoming available.

Thus one solution is the addition of a higher level activity is: the allocation of resources to the SRN process. Consider the assignment of people to project in either the personnel department or SRN management domain, the procedure may be represented through implied inheritance of policies through the procedures using an object-oriented paradigm, as illustrated in Figure 6.

FIGURE 6. Hierarchical Effects of Higher-level Policy



It will be obvious that these actions are a form of "Procedure Enforced Policy" PEP (see 3.1) with a simple parameterization of policies that depend on the context or value of the data instance in the object.

2.2.2 An Engineering Release Policy Framework

This scenario discussed here is given *only* from an Engineering Designer's Perspective

> An Engineering Designer works on the design new version of some part of a product item. The job was initiated by the manager who was given an engineering change order (by some unit manager). The designer works for weeks, reporting each week on work expended and progress. As the design progresses, it is tested by the designer and then by a separate validation team member. It is not until the approval of this product by the validator that the new version is approved.

The external view is that of an Organization existing in an Environment that has a set of relevant Laws of operation and provides Requests (for products) to the Organization and absorbs its Products. The Organization, acting through its Agents, adds to the Laws in a set of Policies, and these constrain the operation by forming a compatible set of rules that cause restrictions on the Actions that take the external Requests and transform them into Products.

In doing this, the organization will consume Resources and obtain other external resources (e.g., Income and Services).

The External Objects in this representation must therefore be defined to be:

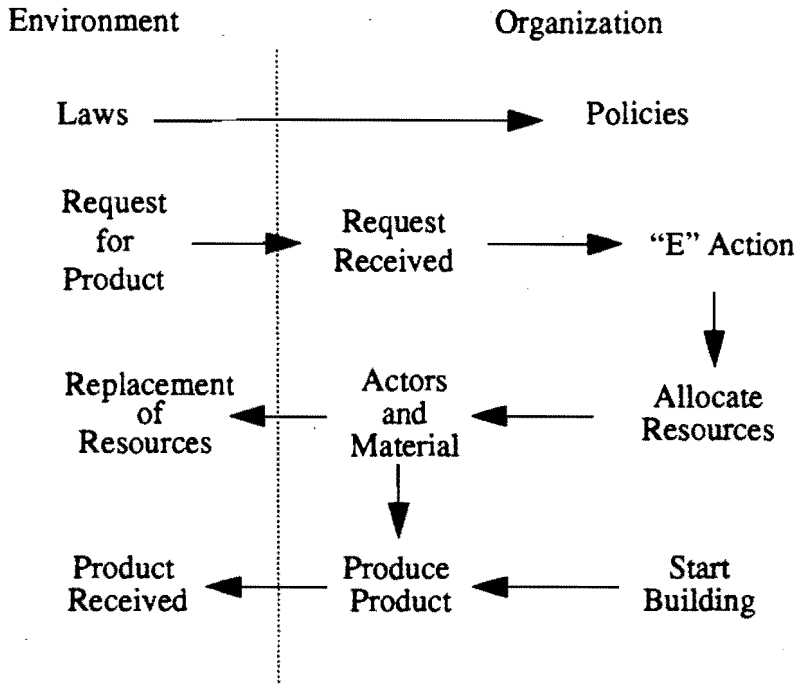
> Requests for Products, orders via the sales force,

Resources, such as raw material and people, etc.

Mechanisms to absorb Products delivered

Means to produce and deliver Orders for resources.

FIGURE 7. The Example as a Block Diagram



The Internal Objects then are:

Policies (that relate to the Laws in a somewhat hierarchical fashion)

Agents to make policy and initiate actions (they may be a resource).

Actions that transform requests into products.

Resources that are used-up in the actions:

Actors to make the product,

Material Resources (that are used-up),

Requests for new or replacement resources, and

Products of the actions.

These are illustrated in Figure 7.

The hierarchic, lattice, inheritance, or specialization of these objects is:

Authority for Sign-Off Agent

Project Manager Agent

Project team member Agent

Validator Agent

Unit manager Agent

Project/Project part Product

Effort Report Product

Test Results Product

Design Product

Version Product

Engineering Change Order Request (internal)

Report on progress and expenditures Action

Request Test Validation Action

Apply Test Validation Action

Request Approval to make new Version Action

Make New Version Action

Give Approval to make new Version Action

In order to understand the application of the policy, a time line may be generated showing interactions. The following are a few sample policy instances:

- 1. Requests are handled by O objects (of class A). If someone else, Error 0.
- 2. Actions to schedule people and hold material for manufacturing are made by
- E object instances (of class A) If not possible to create new instance, Error 1. If not enough material, Error 2. If no people available, Error 3.

Note:

This depends where the assignment is made - maybe it is done in the scheduling or personnel assignment and thus not a possible error here. However, it could also be a new type of real-world problem, like illness, etc. at the time they are needed.

- 3. NRR instances may only be created by E object instances.

In this case, a Time Line may be checked to be valid, as follows:

- 1. A new order is received (by an instance of the Observer Actor (O)).
  By Policy 1, OK
- 2. This O instance now makes a new instance of "Request Received" (RR)
  By Policy 2, error condition 1 may occur
- 3. Some Executor Actor (E) instance now looks at the total set of RR and works out schedules by looking at Actor (A) and Material (M) instances and deciding who works on the RR, updating QOH and Q held accordingly. At that time, stock depletion to reorder point will be checked (by E) and if needed a New Resource Request (NRR) instance is created.
  By Policy 2, error conditions 2 and 3 may apply
  Or Policy 2, error conditions 4 and 5 may apply (if there is no supplier?)

- 4. By magic, the day blooms when the product is to be made. Some E instance tells the actor to find the material and put it together. A new Product instance (P) is produced, after some time, and the schedule and material Q held adjusted (when completed) for A and M.

  By Policy 2, error conditions may apply (not in warehouse or person ill, or just left the company).

If this were continued, it would become obvious that policies are missing. One could be:

- 100. When a policy is to be changed, an agent will initiate it (from whatever place makes sense in the cycle).

It is also necessary to look at higher level policy for possible variance and to see how it fits (hierarchically) with lower level policy. Some may be:

- 1. We shall not be more than 2 weeks late on delivery
- 2. Cost overruns shall not exceed 10%
- 3. A decision to act on a schedule needs sign off from the actor's manager.

However, this involves some form of theorem proving technique (or equivalent) and this represents a substantial complexity. It will not be discussed further in this paper, but is currently under investigation by the authors.

Some exemplary attributes of the objects are shown in Appendix 1.

3.0 Policy Enforcement: A Meta Model Approach

There are at least three ways of enforcing policy through computational techniques:

a. Using something akin to automated "Policies and Procedures Manuals"

b. Checking that all initial, intermediate, and final states of the system are operating within policy limits (i.e., assuming that the system has not already violated the rules--akin to the condition of Bell-LaPadula for security), and stopping an action (e.g., deferring output and rolling any changes back) if a violation is detected.

c. Checking that all actions are "safe" (i.e., that they cannot violate policy). This implies a form of trusted code and players similar to that of most multidimensional security systems today.

Here we propose using an object-oriented approach to Policy using a technique of embedding controls within an Active and Extensible Dictionary System [Sibley, 1986]. It should be noted that such an approach will also allow a "generator" style of implementation, and that this may be one of the most effective ways of utilizing the definition.

The key to this is to define the instances or "actual" objects at the database level or level 0 (as reproduced in Table 1); the article also discusses the application of inter-level con-

straints. Indeed, the same proposed technique can be used in all three types of policy enforcement representations. A preliminary investigation of each is now discussed.

TABLE 1. The DBMS/IRDS Levels.

Level	Type of Data Name in DBMS	Name in IRDS
0	Data Database	--
1	Meta-Data Schema	IRDS Database
2	Meta-Meta-Data Schema	IRDS Schema
3	Meta-Meta-Meta-Data Schema	IRDS Meta-Schema

3.1 Procedure Enforced Policy

Actions may be stored as meta data (in a manner similar to that used in object-oriented paradigms); these are invoked to enforce policy (or roll back any attempted change) before starting or after completing an operation. The actions may be implemented as constraints to be satisfied before an action is allowed or after it has completed (i.e., possibly as triggers initiated by or after actions); in some cases, both may be required (e.g., this may be required because the entire processes may be recursively unsolvable [Graham and Denning, 1972; Landwehr, 1981]). These will be termed "procedure enforced policies - PEP".

The problem here is in the difficulty of finding and changing procedures after a policy has been altered; a similar difficulty occurs when a procedure has been found to be incorrect (i.e., it is found to be implementing policy that is either ambiguous or wrong--conflicting or illegal).

In an Active IRDS Environment, probably the simplest way to implement policy, the actions that overtly or covertly enforce the top levels and articulated policy are "hard-coded" -- thus they are embedded in the allowed operations that are "part" of an object. The users of the system are therefore constrained only to operate on those objects. This is therefore the typical well-designed object-oriented approach; only predefined operations are available to the user/applications programmer.

Although design aids can be provided for mode of implementation, they are restricted in their capability, unless a formal (rather than a procedural or programmed definition) specification of the policy and local rules is available. This is unlikely as it represents a mix of formal and informal techniques; these are expensive to produce, difficult to maintain, and generally not cost effective in the overall system life cycle (i.e., they are lacking in sufficient benefit or value for the high cost involved). Due to the lack of a formalism or its tie-in to the implementation, this produces a system that is potentially very expensive to maintain. This, coupled with the difficulty of changing procedures when the policy is altered, make it relatively unattractive, though the commonest approach today in general. Thus it will not be further investigated here.

3.2 State Condition Enforced Policy

A second method of using an active dictionary approach is to use a pre- and post-condition on all actions or operations, as in the Vienna Development Method (VDM) [Jones, 1986], to ensure that the policy is not violated. Here, this is termed a "State Condition Enforced Policy"--SCEP"; this is an analog of one technique for implementing multilevel secure systems for the DoD (e.g., in the Bell-LaPadula Model criteria [Bell, 1976; Bell, 1988], the system must always be in a "secure," "safe," or "compliant" state before and after each action).

One example is:

```
procedure x in a,b,c out p,q
precondition within-policy
....
postcondition within-policy
```

where the boolean procedure: "within-policy" is true if no policy condition has been violated. Of course, we must still consider whether there may be some "intermediate" non-compliant condition, though this may not mean that the result is then an non-policy compliant "overall" system, provided that there is independence between the operations and the dictionary/database features that they are using. Indeed, an intermediate state of a database during update may be in violation of a rule or condition, but the final data system state remains valid after completion of the set of procedures: this occurs in many updates involving a referential integrity condition between distinct parts of the database.

Thus SCEP in an Active IRDS Environment may be implemented by using pre- and post-conditions to initiate policy related actions. Such a system may have its policy sets tailored to the level at which an operation is being initiated. Thus the policy would probably best placed in a "natural hierarchy" (i.e., to reduce processing difficulties) and the parts initiated within operations using the object-oriented approach. Naturally, more restrictions are placed on operations through inheritance as the policy is articulated down the organization.

3.3 Procedure-Restriction-and-Checking Enforced Policy

As a third approach, the one we are primarily pursuing at this time, we tie-in actions to policy whenever a procedure is performing some task that could subvert policy. This we term: "Procedure-Restriction-and-Checking Enforced Policy"-- PRCEP. It is similar to the policy application of several MLS (Multilevel Secure Systems) today, where the original state is secure and checking is made by the kernel (hopefully a relatively small piece of the system, such as a few thousand lines of code) to see that only the correct people initiate allowable and allowed operations on the proper data. There is usually also a "trusted" part (not too large) that is not as closely checked or proved: this acts as the well-understood and proven mechanism for retaining the secure state of the system.

In a similar vein, the set of objects may have "hard policy" constraints embedded in their portfolio of allowable operations on and between Object types. Examples of these expressed in the objects and interactions of the previous system are for the:

- Manager Object type having operations (possibly inherited from a higher level) that control
  - Authority for Sign-Off an Agent instance
  - Giving Approval to make a new Version of an Action
- Designer Object type providing controls of
  - Project team member Agent instances
  - Report instances stating progress and expenditures on an Action instance
  - Request instances for Test Validation Action instances
  - Request instances for Approval to make new Version Action instances
  - Making New Version Action instances
- Testing Object type giving permission for
  - Validator Agent instance to operate
  - Application of Test Validation Action instances

PRCEP in an Active IRDS Environment therefore is best implemented by the policy being "precompiled" into a set of invocable checkers which are stored with the operations on the specific object types that they impact. However, because there may be two or more places (object types) which may be affected by the same policy or its articulation, the "operational constraints" may have to be called from a library and applied with inheritance as pre- and post-conditions.

4.0 Policy Formalization

In order to formalize a model of any system, certain types of tools and representations must be defined. This would involve:

- A rule maintenance and consistency checker: needed because policies are very likely to be dynamic (i.e., change over time).
- A means of formal representation of the policies in the system. For example, if an automated theorem prover is to be used to check the consistency of a set of policies, they must be represented as a set of well-formed formulas (axioms).

The type of axioms representing the policies in an SRN system includes some or all of the following categories:

- Existence:

  Defining the existence of roles and activities, classes of roles, and activities, etc.

- Activities that are the to be conducted by each role:

  "A Programmer is responsible for submitting a written SRN for any change to a module of a system, and must respond to a Program Manager's response to an SRN within five working days."

- Interaction between roles/activities:

  "All interaction between a Programer and the [Configuration/Quality Assurance Manager and Librarian] must go through Project Manager."

- Timing (ordering) of activities:

  Pre- and post-conditions.

  "An SRN can be entered into the SRN database if and only if the SRN has been approved by the Project Manager and Configuration/Quality Assurance Manager."

- Interpretation of each policy:

  Beliefs.

  Instantiation of roles and activities. In an object-oriented model, objects and classes are equivalent to templates.

- Permission to circumvent policy:

  The set of conditions and specific permissions.

  "A Programmer may submit an SRN request directly to a Configuration/Quality Assurance Manager if the Project Manager cannot be contacted for two consecutive business days."

- Meta-policies:

  "A Configuration/Quality Assurance Manager may never receive permission to circumvent a rule."

- What to do if no applicable policy is found:

  Default rules.

- Inconsistency:

  Resolving conflicts between rules.

These represent some of the classes that are currently being investigated in our research efforts. Attempts are under way to apply formalism to this and similar problems.

5.0 Conclusions

This paper has given some selected definitions of the term "policy" and provided a discussion of some alternative ways of enforcing it, both "manually" (by management procedures and practices) and automatically. When a computer is used in the process of policy enforcement, there are several alternative methods that can be applied, from the program (computer procedural) method that ensures no side effects other than those that are allowed in the "translated" statements of the organization's "Policy and Procedures Manual" to a rule-based (artificial intelligence motivated) mechanism that attempts to enforce all the policy expressed as the set of equivalent rules or "company laws," etc.

At both ends of this policy enforcement implementation spectrum, there are problems. At one end, the policy is embedded in computer programs and is therefore potentially intractable to change. At the other end, the rules are generally a very complex and interrelated

set that may prove incompatible (or ambiguous) -- thus the enforcement mechanism may need to have a complete theorem prover as its basis (either used before and after applications of rules - as in some pre and post condition enforcing mechanisms, or precompiled as a set of non-ambiguous rules - which may be arduous initially and difficult to maintain).

We are currently taking a somewhat middle of the line approach: the application of an object-oriented paradigm, with inheritance to allow augmented rule application at the different lower levels of the organization and a meta-schema approach that should reduce conflict between the rules. Naturally, this will require theorem proving as a basic part of the system. From an ISD perspective, an object-oriented approach to system analysis and design provides a rich mechanism for modeling a system on real-world policies and entities. It also, however, introduces additional complexity into the requirements for a rule enforcement mechanism (e.g., the inference engine [rule enforcer/theorem prover] must be capable of applying and maintaining the various methods associated with each message).

6.0 References

[Bell and LaPadula, 1976]

Bell, D. E., and LaPadula, L. J. Secure Computer System: Unified Exposition and Multics Interpretation, MTR-2997, MITRE, March 1976.

[Bell, 1988]

Bell, D. E. "Concerning 'Modelling' of Computer Security." Proceedings 1988 Symposium on Security and Privacy, April 1988.

[Coordination, 1983]

"Coordination Technology as the Basis for a Programming Environment." Electrical Communication, Vol. 77, No. 4, 1983, pp. 307-313.

[Dobson and McDermid, 1989a]

Dobson, J. E., and McDermid, J. A. "A Framework for Expressing Models of Security Policy." Proceedings 1989 IEEE Computer Society Symposium on Security and Privacy, IEEE, Oakland, CA, May 1989, pp. 229-239.

[Dobson and McDermid, 1989b]

Dobson, J. E., and McDermid, J. A. Security Models and Enterprise Models, Database Security: Status and Prospects II, C. E. Landwehr (Ed.), Elsevier Science Publishers, Amsterdam, 1989.

[Graham and Denning, 1972]

Graham, G. S., and Denning, P. J. "Protection--Principles and Practice." Proceedings of the AFIPS Spring Joint Computer Conference, 1972, pp. 417-29.

[Hertz, 1986]

Hertz, D. B. Expert Systems for the Analysis and Synthesis of Strategic Policy. Proceedings of the IFAC International Conference on Economics and Artificial Intelligence (Aix-en-Provence, France, September 2-4, 1986), pp. 43-48.

[Hirschheim and Klein, 1989]

Hirschheim, R., and Klein, H. K. Four Paradigms of Information Systems Development. Communications of the ACM, Vol. 32 No. 10, October 1989, pp. 1199 - 1216.

[Jones, 1986]

Jones, C. B. Systematic Software Development Using VDM. London, Prentice-Hall, 1986.

[Landwehr,1981]

Landwehr, C. E. A Survey of Formal Models for Computer Security, Technical report 8489, Naval Research Laboratory, Washington, D.C., September 30, 1981.

[Mumford, 1985]

Mumford, E. Values, Technology and Work. Information Systems. London, Martin Nijhoff Publisher, 1981.

[Murata, 1989]

Murata, T. "Petri Nets: Properties, Analysis and Applications." Proceedings of the IEEE, vol. 77, no. 4, 1989.

[Petri, 1962]

Petri, C. A. "Kommunikation mit Automaten." Bonn: Institut fur Instrumentelle Mathematik, Schriften des IIM Nr.3, 1962.

[Sibley, 1986]

Sibley, E. H. "An Expert Database System Architecture Based on an Active and Extensible Dictionary System." Proceedings First International Workshop on Expert Databases. Menlo Park, Benjamin Cummings, 1985, pp.

Appendix: Exemplary Attributes of the Objects

The major attributes of the objects are:

- 1. For the Actor Object
  - Identifier (name)
  - Schedule
- 2. For the Material Object
  - Identifier (name)
  - Quantity on Hand (unexpended)
  - Quantity held for current Orders
  - Reorder point (or equivalent)
- 3. For the New Resource Request Object
  - Identifier (name)
  - Reorder Quantity
  - Supplier ID
- 4. For the Request Object
  - Identifier (name)
  - Quantity
  - Customer
  - Date needed
- 5. For the Product Object
  - Identifier (name)
  - Quantity
  - Customer

Error Conditions of Objects are:

E1. For an Actor Object

- 1. Problem with name
- 2. Overlapping schedule found (more than one thing to do at the same time)

E2. For the Material Object

- 1. Problem with name
- 2. Impossible QOH (or wrong by inspection)
- 3. Impossible Quantity Held (or wrong by inspection)
- etc.

Important Operations associated with the Objects are:

- 1. For an Actor Object

  add new employee

make/change Schedule

- 2. For the Material Object
  Change QOH
  Change Quantity held for Current Orders
  Reorder
- 3. For the New Resource Request Object
  Add new reorder instance
  Received reordered item
- 4. For the Request Object
  Log request received
  Remove request/completed
- 5. For the Product Object
  Initiate Shipping and billing
  Remove instance of Request now shipped

Some operations that are "started" elsewhere but affect the Object are:

- 1. For an Actor Object
  Using systems under the domination of a policy
  Scheduling by Request Object action
- 2. For the Material Object
  Scheduling by Request Object action
- 3. For the New Resource Request Object
  Out-stocking and reorder by Material Object
- 4. For the Request Object
  Removal when shipped by Product object
- 5. For the Product Object
  Completion of make from Actor/Material objects