
Peter F. Linington
University of Kent

Agenda

• What is a policy?
  • Characteristic properties
  • Required elements
• Forms of Policy
• Policy Dynamics
• Specification of Policy
• Multiple Managers
What RM-ODP Says

- The Foundations says (in 11.2.7):
  Policy: A set of rules related to a particular purpose. A rule can be expressed as an obligation, a permission or a prohibition.
  NOTE -- Not every policy is a constraint. Some policies represent an empowerment.
- This is too vague.
  - Could count almost anything as a policy
  - The Enterprise Language (ISO 15414) adds some precision, but still leaves open issues.

Policies

- Take a step back: dictionary definitions
  - courses or principles of action adopted or proposed by an organization or individual
  - prudent or expedient conduct or actions
  - *in Scotland, pleasure-grounds round a mansion.*
- Connotations
  - there is real choice to be exercised
  - the choice is likely to change if circumstances change
  - a policy is a structured collection of rules or preferences
  - may be applied to the set of choices to be made rather than the set of solutions – “*each department shall have a safety policy*”.
Policy Examples

- Registration before the end of May attracts a 10% discount.
- Only gold-class subscribers may access the service if the load is already above 50%.
- If the battery life is below one hour, start migrating modified files to central server.
- If the patient’s temperature is above 40, senior nurses are permitted to administer aspirin; they are obliged to alert the doctor on duty.
- All actions taken must be legal in the State of California.
- If the CEO is unable to perform their duties, the next most senior member of staff on site shall delegate for them.

A Software Engineering View

- A System is specified to satisfy a set of requirements
  - This can be expressed as a contract between consumer and provider of services.
- Policy is about deferring some of the decisions
  - But major design decisions have already been made, and so should be regarded as constraints on behaviour that a policy can require.
- We should be able to verify properties of a design that remain true no matter what policy specifications are subsequently selected.
  - This implies limiting the behaviour a policy can have so as to respect the established design decisions.
Epochs for Specification and Use

• We can distinguish nested epochs of progressive specification and use.

specify

set policy  set policy  set policy

use  use  use  use  use

Policy valid  Policy valid  Policy valid

deployment

Base specification valid

time

Policies and Policy Envelopes

• The term Policy can be used in two ways, denoting
  • a set of rules to be obeyed in some particular situation
  • a point in a larger specification at which such a set of rules needs to be provided. Over a period of time, such a point may be associated with many different specific sets of rules – i.e. policies change.

• This is essentially a type-instance distinction, but since there are many other uses of policy type with different scopes and purposes, we use the distinct term policy envelope to indicate
  • a named point in the specification at which a policy should be defined
  • the constraints on the policies that are acceptable at that point.
ODP Policy Envelope

- The ODP Enterprise Language work had adopted the envelope/value distinction.
- The new ODP definition combines all the elements of policy in the base specification into the policy envelope.
- It contains:
  - Constraints on acceptable values
  - The current policy value
  - Definitions of the behaviour that changes the policy

Policy Definition Concepts

- To define a policy at design time we need to extend the set of concepts provided:
  - Policy identity
  - Policy point of application
    - The elements of behaviour (or state) being controlled
  - Policy envelope
  - Policy value
  - Policy environment
    - Subset of the envelope that identifies visibility of elements of state
    - Policy setting behaviour
Current Policy Work

- The current work on policies covers a spectrum with emphasis on different models, from the business process to much closer to the network infrastructure
  - IETF Policy Framework – DEN – CIM/PIM
    - Uses directories and control components within the network
  - Event/condition/action languages – Ponder – PDL
    - Use application entities and roles, but simple and directly interpretable behaviour (state machine like)
      - Domains in a directory give structure and control
  - ODP Enterprise language – communities and roles
    - Behaviour and constraints on business roles, with emphasis on composition of communities for structure

Ponder Examples

An Authorisation Policy that could be generated from an authorization model.

```
type auth+ video (subject s, Time start, Time end) {
  target videoChannel;
  action start;
  when time.between (start, end); }

inst projectVideo = video (/project/vconf, “1400”, “1900”);
inst reviewVideo = video (/taught/ugrad, “2000”, “2400”);
```
Expressive Power

- One of the attractions of these languages is their flexibility and power
  - Respond to events with any action
    - including injection of higher-level events
  - Filter by source/target domains
  - Guard by expressions including arbitrary system properties

- The Dark Side
  - Conflicts likely and must be handled
  - Emergent behaviour hard to predict
  - Much configuration information outside policy
  - Policy author has complete power

Limits on Expressive Power

- To solve this conflict, we need to place additional constraints.
- We need to limit power; what have other people done?
- Draw parallel with Active Networks
  - Each packet carries a fragment of program
  - This is executed in e.g. routers for flexible distributed QoS management.
  - But this means running code fragments inside critical network components
    - Need to provide a strong sand box
    - Proposals to do so by weakening the power of the language used.
Active Network Languages

- Owen, et al.
  - Modal Classes
    - Declared as containing state elements
    - Methods have guards based on state elements
    - Control setting of state elements by external policies
    - Policies can be replace dynamically.
      - Still trusting author to use sensible guards
- Hicks et al.,
  - PLAN
    - Simple functional language,
    - without general recursion to give resource limits
    - Strong typing
      - Works within limits, but needs structured privileges.

What should a policy look like?

- Enumerations?
  - No problem, as long all elements acceptable. Weak.
- Declaration of an invariant?
  - Over what set of terms?
  - Constraint-oriented specification
- Evaluation of expressions?
  - Over what set of terms? What result type?
  - OK for simple types, but major problems with references.
- Execution of program
  - Over what set of terms?
  - Side effects are a major problem for constraining what can be done!
The controlled behaviour

- Should it be possible to navigate from a policy to all the behaviours it affects?
  - Not at execution time – too burdensome.
  - But the envelope might define patterns for situations where it is applicable.

```java
if (!XyzPolicyEnvelope.getCurrentValue().checkPermA())
    throw new InvalidAException();
```

- Horrid! At least give some syntactic support?
- An opportunity for MDE support for incorporating the checks at all the right places.

* disclaimer about syntax!

A Preliminary Hypothesis

- Policies should be
  - Defined over a restrictive view of the basic specification
  - Declarative or pure functional (with no side effects)
  - Yield a result that is strongly typed
  - They should not have side effects.
Restricted Views

• Are needed, but with what level of discrimination?
  • Need something better than just an explicit parameter list in the policy envelope.
  • Could base view on modular structure of basic specification, but is this enough?

• Needs some experiments to see how good designers are at capturing/predicting requirements!

Return Types

• The problem with evaluation to a reference is that the type you need to give a precise enough constraint may not exist.
  • Consider the policy that says who should delegate for the CEO when he has an accident
    • If the envelope constraint is more complex than “any staff above grade 6”, don’t want to clutter the organogram with the type you need.
  • But in ODP, a type is a predicate.
  • So why not allow constraint in terms of a local type expression as shorthand?

```plaintext
findDelegate(CEO) : (Staff s | s.grade>6 ∧ s.service>2)
```
Changing Policies

Policy Footprint

• Most of the policy languages under active discussion have a strong emphasis on a event-condition-action styles of modelling.
  • Their use of filtering and guards depends on state
• Can talk about policies and policy envelopes in terms of coverage and reachability of regions of state space (or, if you like, phase space, folding in event trace as part of state)
Coverage of State Space

- Consider the footprints in state space
  - the envelope has largest coverage;
  - individual policies are subsets of this;
  - They probably all share some initial state.

![Envelopes and Policies](image)

Changing Policy

- What happens if we want to change policy and the system is in a state that would no longer be valid?
- Transition rules; do you make the change:
  - if at rest?
  - if in overlap region?
  - when next in overlap?
    - consider concurrency – does it matter if introduction of related changes is gradual?
    - guarantee change within a bounded period?
    - need transactional change of group of policies?
    - do you define matching sections?
  - Evolution and planned change – ramping down, then up?
Timing, Versions and the Tool Chain

- The Enterprise Specification is maintained throughout the lifetime of the system. If policies change, then these changes need to work through to the running system, often on precise time-scales.
  - a pricing policy, or a tax structure may need to be introduced on a specific date;
  - other development and maintenance will need to be carried out in parallel;
  - recovery and back tracking need to be supported;
  - need preparatory changes and dummy running;
  - implies need for flexible build process and version control.

Policy Specification
Linking Specifications

- The definer of the policy envelope may need to:
  - Provide mechanisms for identifying or selecting specific policies
  - Control the working set of policies available for selection
  - Provide suitable default policies for *out of the box* running
  - This is against the static nature of UML class structure.

Policy Envelope Revisited

- Remove stuff not constraining value definition
- Envelope elements
  - Pattern for application
  - Set of behaviours influenced
    - Actions in simple cases
  - View of state used to assess policy
  - Restrictive types for results
- Ideally, syntactic sugar for use of current value by reference to policy.
The Shrink-Wrapping Problem

- In building a system, components, including collections of policies, may be locally specified, reused or purchased in a shrink-wrapped form, on their own or with a subsystem.
- How do we ensure consistency and predictability?
- How do we link the different parts of the specification, and manage changes to them?

Enterprise Constraints on Hooks
Linking Models – UML

- Struggle to find a neat way of integrating UML models
  - Need to manage references between models with minimum of deployment effort, separate evolution.

Deployment/Configuration Models

- How do other people do it?
  - Look at UML profile for Framework Architectures – links & decoration with configuration and refinement constraints.
    - Can do almost anything with tags and stereotypes
    - But need strong support for tool vendors to take them seriously
    - Specifics of that proposal too low level – assumes you are defining implementation structure
  - EJB configuration tools have similar integration problems
    - Effectively steer the process by introducing an external level of deployment specification.
  - Could make it a preparatory transformation; use MDE!
Conclusions: Where Next?

• Designers of policy languages should consider both
  • Expressive power
  • Ability to define and limit a policy envelope
    • Need to experiment and see if something more controllable than state/event/action can be made sufficiently easy to use.
• More work is needed on the dynamics of policies, particularly on the interaction of policy change and concurrency.
• Policies are a powerful and successful tools, but they are only one aspect of system specification, and we must not lose sight of the need to fit them into the bigger picture.