Position paper for
Policy Workshop 1999, 15-17 November 1999
HP- Laboratories, Bristol, UK

Requirements and Policies

Jonathan D. Moffett
Department of Computer Science, University of York
Heslington, York YOl0 5DD, UK
jdm@cs.york.ac.uk

1. Introduction

This position paper explores the relationship between requirements and policies and makes some suggestions, based on this, for broadening the scope of high level policies.

2. Requirements Engineering

Requirements engineering [1] is a discipline which has emerged over the last few years in response to the realisation that too many systems were being built which worked, in some sense, but did not do what their users wanted. It was realised that it was essential to distinguish the specification of requirements – the "what" of a system – from the "how" of its implementation.

One definition of a requirement is: "Something called for or demanded, a condition which must be complied with", and it will immediately be recognised that this definition is not far removed from working definitions of policies.

Requirements specifications are typically used across contractual boundaries. An important aspect of them is that a requirements specification should provide implementers with everything they need to know to satisfy the relevant stakeholders, but nothing more. In other words the implementer should have freedom to satisfy the requirements in whatever way is most appropriate.

It should be noted that Requirements Engineering is normally conducted by modelling, which has two components: static structures and dynamic behaviour.

3. High Level Policies

We have not thought it necessary in this forum to give an exposition of Distributed Systems Management Policies. However, they are typically regarded as objects, and the Requirements activity is normally carried out on systems. But, high level policies are typically rather static, and in any case one of the major concerns of Requirements Engineering is the tracking of changes and tracing their consequences. It would therefore seem appropriate to consider treating high level policies as requirements. It also seems reasonable to treat static system policies and changeable policy objects in the same framework; in particular, that is the only way in which to detect conflicts which span both kinds of policy.

Using [2] as a basis, we put forward a number of aspects of high level policies on which the requirements approach could throw light.

High and Low Level Policies

There are high level policies, such as

```
O+ Manager M must protect Dept D files from loss due to fire
   or media failure
```

and low level goals, derived from it, such as:

```
A+ Domain BackupSW{read, write} Domain D files
```

This low level policy is one of several in a set which implements the high level one by means of regular back up.

In [3] we envisaged a relationship between policies at different levels of a policy hierarchy, in which partitioning, refinement and delegation of responsibility could be shown to achieve the high level policy. However, it now seems to us to be more useful to regard high level policies as requirements, and the low level ones as an implementation of them. The standard techniques of a system life-cycle process can then be used to derive the implementation from the requirements.

Regarding high level policies as requirements could have the advantage of ensuring that other solutions to the problem, which are not necessarily regarded so naturally as low level policies, are considered. For example, improving the fire-proofing of the building and using more reliable media might be an acceptable implementation of this high level policy.

Policy Modes

Another aspect of policies which should be considered is whether the structuring of high level policies into the +/- Obligation/Authorisation paradigm may be inappropriate, particularly if our requirement is that something should not happen. Suppose we have a requirement such as:

- The standby manager shall[1] not perform any control actions

This enables two different possible implementations by means of low level policies: negative obligation; or negative authorisation. It is a design and implementation issue which of these mechanisms should be used, and will depend on factors such as the trustworthiness and reliability of the manager, the need for adequate authority in emergencies, etc.

Policies in Distributed Systems

It will be particularly useful to distinguish between requirements and implementation in distributed systems with autonomous participants. In order to give them maximum autonomy, the principle of providing implementers with everything they need to know to satisfy the relevant stakeholders, but nothing more, should be respected.

Static Structures

One aspect of policies that has not been given much attention, is the static structures upon which they operate. Let us re-examine the example from above:

```
A+ Domain BackupSW{read, write} Domain D files
```

[1] "Shall" is the conventional verb for expressing a requirement.

There is an implied structure here. Presumably `BackupSW` and `D files` are well-defined in some domain hierarchy. However, that structure is not part of the policy specification, but is given as part of a particular context. This may be satisfactory for specific implementation, but it is possible to envisage problems arising in a wider context:

- In a distributed systems context, it assumes that all users use the domain hierarchy notation, i.e. it forces an implementation on them rather than stating a requirement;
- The domain hierarchy notation is very powerful, but can it actually satisfy all reasonable demands for data structuring? It is considerably less rich in relationships than structuring notations such as Entity-Relationship diagrams.
- For high level policies, a less implementation-oriented notation may be useful for expressing requirements.

Behaviour

The current policy notation appears to be adequate for describing when low-level actions should or should not occur. However, it may not be powerful enough for some purposes. To take examples from two quite different areas:

- Does concurrency have a part to play?
- How should we express mandatory security policies such as the Chinese Wall Security policies?

These are both aspects of policies which need to be able to be expressed at the requirements level. where behavioural notations, such as some of the components of UML [4], already exist for this purpose.

4. Conclusions

Policy notations have made a real contribution to enabling policies to be expressed and changed. However, the practical use of them has been at a rather low level. It will be useful to consider how far high level policies should be regarded as requirements, with two main benefits:

- The rich notations of Requirements Engineering will become available;
- The specification of policies can be integrated into the general process of specification of requirements for a system.

References

[1] G. Kotonya and I. Sommerville, *Requirements Engineering - Processes and Techniques*: John Wiley, 1998.

[2] M. S. Sloman, "Policy Driven Management for Distributed Systems," *Journal of Network and Systems Management*, vol. 2, 1994.

[3] J. D. Moffett and M. S. Sloman, "Policy Hierarchies for Distributed Systems Management," *IEEE Journal on Selected Areas in Communication*, vol. 11, pp. 1404-1414, 1993.

[4] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*: Addison-Wesley, 1998.