An Approach to Understanding Policy Based on Autonomy and Voluntary Cooperation

Mark Burgess

Oslo University College, Norway
mark@iu.hio.no

Abstract. Presently, there is no satisfactory model for dealing with political autonomy of agents in policy based management. A theory of atomic policy units called 'promises' is therefore discussed. Using promises, a global authority is not required to build conventional management abstractions, but work is needed to bind peers into a traditional authoritative structure. The construction of promises is precise, if tedious, but can be simplified graphically to reason about the distributed effect of autonomous policy. Immediate applications include resolving the problem of policy conflicts in autonomous networks.

1 Introduction

One of the problems in discussing policy based management of distributed systems[1, 2] is the assumption that all of the nodes in a network will follow a consistent set of rules. For this to be true, we need either an external authority to impose a consistent policy from a bird's eye view, or a number of independent agents to collaborate in a way that settles on a 'consistent' picture autonomously.

Political autonomy is the key problem that one has to deal with in ad hoc / peer-to-peer networks, and in pervasive computing. When the power to decide policy is delegated to individuals, orders cannot be issued from a governing entity: consistency and concensus must arise purely by *voluntary cooperation*. There is no current model for discussing systems in this situation.

This paper outlines a theory for the latter, and in doing so provides a way to achieve the former. The details of this theory require a far more extensive and technical discussion than may be presented in this short contribution; details must follow elsewhere.

It has been clear to many authors that the way to secure a clear and consistent picture of policy, in complex environments, is through the use of formal methods. But what formalism do we have to express the necessary issues? Previous attempts to discuss the consistency of distributed policy have achieved varying degrees of success, but have ultimately fallen short of being useful tools except in rather limited arenas. For example:

- Modal logics: these require one to formulate hypotheses that can be checked as true/false propositions. This is not the way system administrators work.

– The π-calculus: has attractive features but focuses on issues that are too low-level for management. It describes systems in terms of states and transitions rather than policies (constraints about states)[3].
– Implementations like IPSec[4, 5], Ponder[6] etc. these do not take explicitly into account the autonomy of agents and thus while these implement policies well enough, they are difficult to submit to analysis.

In each of the latter examples, one tends to fight two separate battles: the battle for an optimal mode of expression and the battle for an intuitive interface to an existing system. For example, consider a set of files and directories, which we want to have certain permissions. One has a notion of policy as a specification of the permission attributes of these files. Policy suggests that we should group items by their attributes. The existing system has its own idea of grouping structures: directories. A simple example of this is the following:

```
ACL1:                            ACL2:
 1. READ-WRITE /directory         1. READ-ONLY  /directory/file
 2. READ-ONLY  /directory/file    2. READ-WRITE /directory
```

Without clear semantics (e.g. first rule wins) there is now an ordering ambiguity. The two rules overlap in the specifically named "file", because we have used a description based on overriding the collection of objects implicitly in "directory".

In a real system, a directory grouping is the simplest way to refer to this collection of objects. However, this is not the correct classification of the attributes: there is a conflict of interest. How can we solve this kind of problem?

In the theory of system maintenance[7], one builds up consistent and stable structures by imposing independent, atomic operations, satisfying certain constraints[8, 9]. By making the building blocks primitive and having special properties, we ensure consistency. One would like a similar construction for all kinds of policy in human-computer management, so that stable relationships between different activities can be constructed without excessive ambiguity or analytical effort. This paper justifies such a formalism in a form that can be approached through a number of simplifications. It can be applied to network or host configuration, and it is proposed as a unifying paradigm for autonomous management with cfengine[10].

2 Policy with autonomy

By a policy we mean the ability to assert arbitrary constraints of the behaviour of objects and agents in a system. The most general kind of system one can construct is a collection of objects, each with its own attributes, and each with its own policy. A policy can also be quite general: e.g. policy about behaviour, policy about configuration, or policy about interactions with others.

In a network of *autonomous* systems, an agent is only concerned with assertions about its own policy; no external agent can tell it what to do, without its

consent. This is the crucial difference between autonomy and centralized management, and it will be the starting point here (imagine privately owned devices wandering around a shopping mall).

Requirement 1 (Autonomy) *No agent can force any other agent to accept or transmit information, alter its state, or otherwise change its behaviour.*

(An attempt by one agent to change the state of another might be regarded as a definition of an attack.) This scenario is both easier and harder to analyze than the conventional assumption of a system wide policy. It is easier, because it removes many possible causes of conflict and inconsistency. It is harder because one must then put back all of that complexity, by hand, to show how such individual agents can form collaborative structures, free of conflict.

The strategy in this paper is to decompose a system into its autonomous pieces and to describe the interactions fully, so that inconsistencies become explicit. In this way, we discover the emergent policy in the swarm of autonomous agents.

3 Promises

The analysis of 'promises' is naturally motivated by the theory of games and voluntary cooperation[11, 12] and has, to the author's knowledge, only previously been mentioned in a recent context of economics[13].

A promise is a general and abstract unit of intent. Promises, between agents, can deal with things like quality of service, quality of behaviour, specifications of state, etc. Policies of various types have been identified. For instance, in the Ponder model[6], one has authorizations (promises to grant access) and obligations (promises to follow up on a different promise) or dependency, etc. These can all be translated into the notion of promises.

Consider, then, a set of autonomous agents of objects represented as nodes $\mathcal{N} = \{n_1, n_2, \ldots, n_N\}$ in a graph.

Definition 1 (Promise) *A promise is a labelled directed edge (link) that connects two nodes. The promise label represents a specifically intended range of behaviour χ from within a domain of possible behaviours. i.e. $n_1 \xrightarrow{\chi} n_2$. A promise is thus made by a node n_1 to a node n_2. A promise is assumed to be always kept.*

Although it will be important, at a later stage, to discuss whether or not promises are kept, we wish to avoid this issue in the initial discussion; we assume it to be true. Notice also that, in the definition, the agent-nodes, between which promises are made, are kept separate from the constraints between them. This is important for avoiding the kinds of ordering ambiguities alluded to in the introduction.

Example 1 (Service Level Agreement (SLA)). Agent n_1 promises agent n_2 to provide service of type 'database access in time q', τ is the type domain $q \in [0, \infty]$ and the constraint $\chi(q) : 0 < q < 10$ms.

The formulation of a promise, above, has obvious characteristics of a directed graph. It is not a particularly novel construction. It bears a passing resemblance to the theory of capabilities in ref. [14], for instance. Graphs have many desirable properties for defining relationships between entities[15], and there is good reason to retain these properties in describing the relationships between agents. In subsequent work, it will become clear that graphs will prove a useful abstraction of themselves, for management; it is possible to transform graphs and use their spectral properties to discover useful and important properties for management[16–18].

Two special types of promise will be identified below, in order to rebuild conventional structures from these basic atoms.

- A promise to agree to behave like another.
- A promise to utilize the promise of another.

The first of these is essential for defining groups, roles and social structures with concensus behaviour. The latter is crucial for client-server interactions, dependencies and access control. The rest of this paper is about logically combining individual promises into collective and consistent policies that allow cooperation between autonomous agents.

4 What is an inconsistency?

In the extreme case, in which every agent were independent and could only see its own world, there would be no need to speak of inconsistency: unless agents have agreed to be similar, they can do as they please. The only problem that might occur is if an agent promised two contradictory things to a second agent.

Definition 2 (Broken promise) *A promise of χ_1 from agent n_1 to agent n_2 is said to be broken if there exists another promise from n_1 to n_2, of χ_2, in which $\chi_1 \neq \chi_2$.*

This definition is very simple, and becomes most powerful when one identifies promise *types* which is beyond the present scope. It says that an agent can only break its own promises: if an agent promises two different things, it has broken both of its promises. One might feel the need to define 'redundant' promises as being different from broken promises, e.g. if one promise merely extends another then the other is unnecessary; but this opens up an unnecessary subjectivity into the comparison and leads us into trouble straight away. The definition unambiguously identifies a conflict of intention and it can be left up to a human to decide which of the promises is correct, incorrect, redundant etc.

5 Promise analysis

Logic is a way of analysing the consistency of assumptions. It is based on the truth or falsity of collections of propositions $p_1, p_2, \ldots$. One must formulate these

propositions in advance and then use a set of assumptions to determine their status. The advantage of logic is that is admits the concept of a proof.

Is there a logic that is suitable for analyzing promises? Modal logic has been considered as one possibility, and some authors have made progress in using modal logics in restricted models[19, 20]. However, there are basic problems with modal logics that limit their usefulness[21].

More pragmatically, logic alone does not usually get us far in engineering. We do not usually want to say things like "it is true that $1 + 1 = 2$"? Rather we want a system, giving true answers, which allows us to compute the value of $1 + 1$, because we do not know it in advance. Ultimately we would like such a calculational framework for combining the effects of multiple promises. Nevertheless, let us set aside such practical considerations for now, and consider the limitations of modal logical formalism in the presence of autonomy.

5.1 Modal Logic and Kripke Semantics

Why have formalisms for finding inconsistent policies proven to be so difficult? A clue to what is going wrong lies in the many worlds interpretation of the modal logics[22]. In the modal logics, one makes propositions p, q etc., which are either true or false, under certain interpretations. One then introduces modal operators that ascribe certain properties to those propositions, and one seeks a consistent language of such strings.

Modal operators are written in a variety of notations, most often with $\Box$ or $\diamond$. Thus one can say $\Box\, p$, meaning "it is necessary that p be true", and variations on this theme:

$\Box\, p$	$\diamond p = \neg\Box\, \neg p$
It is necessary that p	It is possible that p
It is obligatory that p	It is allowed that p
It is always true that p	It sometimes true that p

A system in which one classifies propositions into "obligatory", "allowed" and "forbidden" could easily seem to be a way to codify policy, and this notion has been explored[19, 20, 23, 21].

Well known difficulties in interpreting modal logics are dealt with using Kripke semantics[24]. Kripke introduced a 'validity function' $v(p, w) \in \{T, F\}$, in which a proposition p is classified as being either true of false in a specific 'world' w. Worlds are usually collections of observers or agents in a system.

Consider the formulation of a logic of promises, starting with the idea of a 'promise' operator.

- $\Box\, p =$ it is promised that p be true.
- $\diamond p = \neg\Box\, \neg p =$ it is unspecified whether p is true.
- $\Box\, \neg p =$ it is promised that p will not be true.

and a validity function $v(\cdot, \cdot)$.

5.2 Single promises

A promise is something that is shared between a sender and a recipient. It is not a property of agents, as in usual modal logics, but of a pair of agents. Logic says nothing about this topology of a promise (indeed, we would like to keep this separate, for reasons that become clearer in section 5.7), so one attempts to build this into the semantics.

Consider the example of the Service Level Agreement, above, and let p mean "Will provide data in less than 10ms". How shall we express the idea that a node n_1 promises a node n_2 this proposition? Consider the following statement:

$$\Box\, p, \quad v(p, n_1) = T. \tag{1}$$

This means that it is true that p is promised at node n_1, i.e. node 1 promises to provide data in less than 10ms – but to whom? Clearly, we must also provide a recipient. Suppose, we try to include the recipient in the same world as the sender? i.e.

$$\Box\, p, \quad v(p, \{n_1, n_2\}) = T. \tag{2}$$

However, this means that both nodes n_1 and n_2 promise to deliver data in less than 10ms. This is not what we need; a recipient is still unspecified. Clearly what we want is to define promises on a different set of worlds: the set of possible links or *edges* between nodes. There are $N(N-1)$ such directed links. Thus, we may write:

$$\Box\, p, \quad v(p, n_1 \to n_2) = T. \tag{3}$$

This is now a unique one-way assertion about a promise from one agent to another. A promise becomes a tuple $\langle \tau, p, \ell \rangle$, where τ is a theme or promise-type (e.g. Web service), p is a proposition (e.g.deliver data in less than 10ms) about how behaviour is to be constrained, and ℓ is a link or edge over which the promise is to be kept. All policies can be written this way, by inventing fictitious services. Also, since every autonomous promise will have this form, the modal/semantic content is trivial and a simplified notation could be used.

5.3 Regional or collective promises from Kripke semantics?

Kripke structures suggest ways of defining regions over which promises might be consistently defined, and hence a way of making uniform policies. For example, a way of unifying two agents n_1, n_2 with a common policy, would be for them both to make the same promise to a third party n_3:

$$\Box\, p, \quad v(p, \{n_1 \to n_3, n_2 \to n_3\}) = T. \tag{4}$$

However, there is a fundamental flaw in this thinking. The existence of such a function that unifies links, originating from more than a single agent-node, is contrary to the fundamental assumption of autonomy. There is no authority in

this picture that has the ability to assert this uniformity of policy. Thus, while it might occur by fortuitous coincidence that p is true over a collection of links, we are not permitted to *specify* it or demand it. Each source-node has to make up its own mind. The logic verifies, but it is not a tool for understanding construction.

What is required is a rule-based construction that allows independent agents to come together and form structures that span several nodes, by *voluntary cooperation*. Such an agreement has to be made between every pair of nodes involved in the cooperative structure. We summarize this with the following:

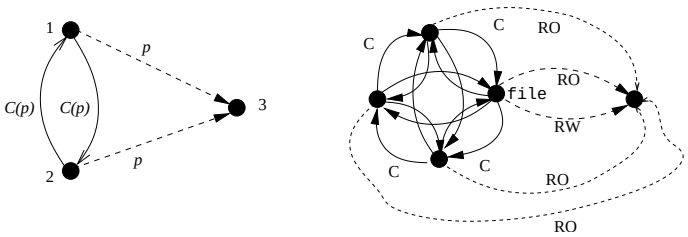

Fig. 1. (Left) Cooperation and the use of third parties to measure the equivalence of agent-nodes in a region. Agents form groups and roles by agreeing to cooperate about policy. (Right) This is how the overlapping file-in-directory rule problem appears in terms of promises to an external agent. An explicit broken promise is asserted by file, in spite of agreements to form a cooperative structure.

Requirement 2 (Cooperative promise rule) *For two agents to guarantee the same promise, one requires a special type of promise: the promise to cooperate with neighbouring agent-nodes, about basic promise themes.*

A complete structure looks like this:

- n_1 promises p to n_3.
- n_2 promises n_1 to collaborate about p (denote this as a promise $C(p)$).
- n_1 promises n_2 to collaborate about p (denote this as a promise $C(p)$).
- n_2 promises p to n_3

By measuring p from both n_1 and n_2, n_3 acts as a judge of their compliance with the mutual agreements between them (see fig. 1). This allows the basis of a theory of measurement, by third party monitors, in collaborative networks. It also shows how to properly define structures in the file-directory example (see fig 1).

5.4 Dependencies and handshakes

Even networks of autonomous agents have to collaborate and delegate tasks, depending on one another to fulfill promised services. We must find a way of

expressing dependency relationships without violating the primary assumption of autonomy.

Consider three agents n_1, n_2, n_3, a database server, a web server and a client. We imagine that the client obtains a web service from the web server, which, in turn, gets its data from a database. Define propositions and validities:

- $p_1 =$ "will send database data in less than 5ms", $v(p_1, n_1 \to n_2) = T$.
- $p_2 =$ "will send web data in less than 10 ms", $v(p_2, n_2 \to n_3) = T$.

These two promises might, at first, appear to define a collaboration between the two servers to provide a promise of service to the client, but they do not.

The promise to serve data from $n_1 \to n_2$ is in no way connected to the promise to deliver data from $n_2 \to n_3$:

- n_2 has no obligation to use the data promised by n_1.
- n_2 promises its web service regardless of what n_1 promises.
- Neither n_1 nor n_3 can force n_2 to act as a conduit for database and client.

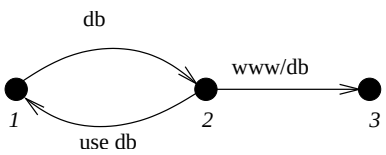

Fig. 2. Turning a conditional dependency into a real promise. The necessary structure is shown in graphical form.

We have already established that it would not help to extend the validity function to try to group the three nodes into a Kripke 'world'. Rather, what is needed is a structure that complete the backwards promises to *utilize* promised services – promises that completes a *handshake* between the autonomous agents. We require:

- A promise to uphold p_1 from $n_1 \to n_2$.
- An acceptance promise, to use the promised data from $n_2 \to n_1$.
- A conditional promise from $n_2 \to n_3$ to uphold p_2 iff p_1 is both present and accepted.

Thus, three components are required to make a dependent promise (see fig. 2). This requirement cannot be derived logically; rather, we must specify it as part of the semantics of autonomy.

Requirement 3 (Acceptance/usage promise rule) *Autonomy requires an agent to explicitly accept a promise that has been made, when it will be used to derive a dependent promise.*

One thus identifies a second special type of promise: the "usage" or "acceptance" promise.

5.5 Autonomous, voluntary cooperation

What use is this construction? First, it advances the manifesto of making all policy decisions explicit. In the example in fig. 2, it shows explicitly the roles and responsibilities of each of the agents in the diagram. Furthermore, the graphical representation of these promises is quite intuitive and easy to understand. The construction has two implications:

1. The component atoms (promises) are all visible, so the inconsistencies of a larger policy can be determined by the presence or absence of a specific link in the labelled graph of promises, according to the rules.
2. One can provide basic recipes (handshakes etc.) for building concensus and agent "societies", without hiding assumptions. This is important in pervasive computing, where agents truly are politically autonomous and every promise must be explicit.

The one issue that we have not discussed is the question of how cooperative agreements are arrived at. This is a question that has been discussed in the context of cooperative game theory[25, 11], and will be elaborated on in a future paper[26]. Once again, it has to do with the human aspect of collaboration. The reader can excerise imagination in introducing fictitious, intermediate agents to deal with issues such as shared memory and resources.

5.6 Causality and graph logic

As an addendum to this discussion, consider *temporal logic*: this is a branch of modal logic, in which an agent evolves from one Kripke world into another, according to a causal sequence, which normally represents time. In temporal logic, each new time-step is a new Kripke world, and the truth or falsity of propositions can span sequences of worlds, forming graph-like structures. Although time is not important in *declaring* policy, it is worth asking whether a logic based on a graph of worlds could be used to discuss the collaborative aspects of policy. Indeed, some authors have proposed using temporal logic and derivative formalisms to discuss the behaviour of policy, and modelling the evolution systems in interesting ways[27–29].

The basic objection to thinking in these terms is, once again, autonomy. In temporal logic, one must basically know the way in which the propositions will evolve with time, i.e. across the entire ordered graph. That presupposes that such a structure can be written down by an authority for the every world; it supposes the existence of a global evolution operator, or master plan for the agents in a network. No such structure exists, *a priori*. It remains an open question whether causality is relevant to policy specification.

5.7 Interlopers: transference of responsibility

One of the difficult problems of policy consistency is in transferring responsibilities from one agent to another: when an agent acts as as a conduit or interloper

for another. Consider agents a, b and c, and suppose that b has a resource B which it can promise to others. How might b express to a: "You may have access to B, but do not pass it on to c"?

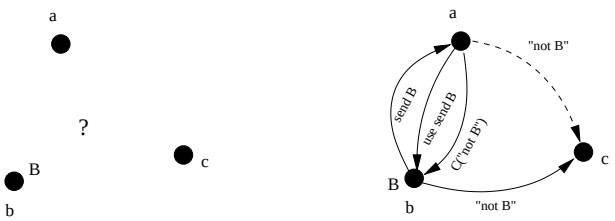

Fig. 3. Transference of responsibility.

The difficulty in this promise is that the promise itself refers to a third party, and this mixes link-worlds with constraints. As a single promise, this desire is not implementable in the proposed scheme:

- It refers to B, which a has no access to, or prior knowledge of.
- If refers to a potential promise from a to c, which is unspecified.
- It preempts a promise from a to b to never give B along $a \rightarrow c$.

There is a straightforward resolution that maintains the autonomy of the nodes, the principle of separation between nodes and constraints, and which makes the roles of the three parties explicit. We note that node b cannot order node a to do anything. Rather, the agents must set up an agreement about their wishes. This also reveals that fact that the original promise is vague and inconsistent, in the first instance, since b never promises that it will not give B to c itself. The solution requires a cooperative agreement (see fig. 3).

- First we must give a access to B by setting up the handshake promises: i) from $b \rightarrow a$, "send B", ii) from $a \rightarrow b$, accept/use "send B".
- Then b must make a consistent promise not to send B from $b \rightarrow c$, by promising "not B" along this link.
- Finally, a promises b to cooperate with b's promises about "not B", by promising to cooperate with "not B" along $a \rightarrow b$. This implies the dotted line in the figure that it will obey an equivalent promise "not B" from $a \rightarrow c$, which could also be made explicit.

At first glance, this might seem like a lot of work for express a simple sentence. The benefit of the construction, however, it that is preserves the basic principles of make every promise explicit, and separating agents-nodes from their intentions. This will be crucial to avoiding the contradictions and ambiguities of other schemes.

6 Conclusions

A graphical scheme for analysing autonomous promises has been outlined in a stripped-down form. Cooperative behaviour requires the presence of mutual agreements between nodes. The value of the promise idiom is to make difficult algebraic constraints into a simple graphical technique that is intuitive for management. A number of theorems can be proved about promises (elsewhere). The promise paradigm forces one to confront the fundamental issues in cooperative behaviour, and can be used to build up systems from scratch, seeing the inconsistencies that arise visually. This also opens the way to make analysis tools and incorporate a wider range of policies in cfengine[10].

In such a short paper, it is not possible to expand on the detailed definitions, proofs or numerous applications of this idea. Some applications include, the analysis of management conflicts (especially in autonomous agencies e.g. in BGP), identification of important and vulnerable nodes, by spectral analysis, and providing a language for a general theory of pervasive, autonomous computing. These will be discussed in future work.

Acknowledgement: I am grateful to Jan Bergstra, Siri Fagernes and Lars Kristiansen for helpful discussions.

References

1. M.S. Sloman and J. Moffet. Policy hierarchies for distributed systems management. *Journal of Network and System Management*, 11(9):1404, 1993.
2. E.C. Lupu and M. Sloman. Towards a role based framework for distributed systems management. *Journal of Network and Systems Management*, **5**, 1996.
3. J. Parrow. *An Introduction to the π-Calculus, in The Handbook of Process Algebra*, page 479. Elsevier, Amsterdam, 2001.
4. Z. Fu and S.F. Wu. Automatic generation of ipsec/vpn security policies in an intra-domain environment. *Proceedings of the 12th internation workshop on Distributed System Operation and Management (IFIP/IEEE).*, INRIA Press:279, 2001.
5. R. Sailer, A. Acharya, M. Beigi, R. Jennings, and D. Verma. Ipsecvalidate - a tool to validate ipsec configurations. *Proceedings of the Fifteenth Systems Administration Conference (LISA XV) (USENIX Association: Berkeley, CA)*, page 19, 2001.
6. N. Damianou, N. Dulay, E.C. Lupu, and M. Sloman. Ponder: a language for specifying security and management policies for distributed systems. *Imperial College Research Report DoC 2000/1*, 2000.
7. M. Burgess. On the theory of system administration. *Science of Computer Programming*, 49:1, 2003.
8. A. Couch and N. Daniels. The maelstrom: Network service debugging via "ineffective procedures". *Proceedings of the Fifteenth Systems Administration Conference (LISA XV) (USENIX Association: Berkeley, CA)*, page 63, 2001.
9. M. Burgess. Cfengine's immunity model of evolving configuration management. *Science of Computer Programming*, 51:197, 2004.
10. M. Burgess. A site configuration engine. *Computing systems (MIT Press: Cambridge MA)*, 8:309, 1995.
11. R. Axelrod. *The Complexity of Cooperation: Agent-based Models of Competition and Collaboration.* Princeton Studies in Complexity, Princeton, 1997.

12. R. Axelrod. *The Evolution of Co-operation*. Penguin Books, 1990 (1984).
13. J.D. Carrillo and M. Dewatripont. Promises, promises. Technical Report 172782000000000058, UCLA Department of Economics, Levines's Bibliography.
14. L. Snyder. Formal models of capability-based protection systems. *IEEE Transactions on Computers*, 30:172, 1981.
15. M. Burgess. *Analytical Network and System Administration — Managing Human-Computer Systems*. J. Wiley & Sons, Chichester, 2004.
16. Tuva Hassel Stang, Fahimeh Pourbayat, Mark Burgess, Geoffrey Canright, Kenth Engø, and Åsmund Weltzien. Archipelago: A network security analysis tool. In *Proceedings of The 17th Annual Large Installation Systems Administration Conference (LISA 2003)*, San Diego, California, USA, October 2003.
17. G. Canright and K. Engø-Monsen. A natural definition of clusters and roles in undirected graphs. *Science of Computer Programming*, 53:195, 2004.
18. M. Burgess, G. Canright, and K. Engø. A graph theoretical model of computer security: from file access to social engineering. *International Journal of Information Security*, 3:70–85, 2004.
19. R. Ortalo. A flexible method for information system security policy specifications. *Lecture Notes on Computer Science*, 1485:67–85, 1998.
20. J. Glasgow, G. MacEwan, and P. Panagaden. A logic for reasoning about security. *ACM Transactions on Computer Systems*, 10:226–264, 1992.
21. E. Lupu and M. Sloman. Conflict analysis for management policies. In *Proceedings of the Vth International Symposium on Integrated Network Management IM'97*, pages 1–14. Chapman & Hall, May 1997.
22. B.F. Chellas. *Modal Logic: An Introduction*. Cambridge University Press, 1980.
23. H. Prakken and M. Sergot. Dyadic deontic logic and contrary-to-duty obligations. In *Defeasible Deontic logic: Essays in Nonmonotonic Normative Reasoning*, volume 263 of *Synthese library*. Kluwer Academic Publisher, 1997.
24. S.A. Kripke. Semantical considerations in modal logic. *Acta Philosophica Fenica*, 16:83–94, 1963.
25. S. Fagernes and M. Burgess. The effects of 'tit for tat' policy for rejecting 'spam' or denial of service floods. In *Proceedings of the 4th System Administration and Network Engineering Conference (SANE 2004)*, 2004.
26. M. Burgess and S. Fagernes. Pervasive computing management ii: Voluntary co-operation. *IEEE eTransactions on Network and Service Management*, page (submitted).
27. A.K. Bandara, E.C. Lupu, J. Moffett, and A. Russo. A goal-based approach to policy refinement. In *Proceedings of the 5th IEEE Workshop on Policies for Distributed Systems and Networks*, 2004.
28. A.K. Bandara, E.C. Lupu, J. Moffett, and A. Russo. Using event calculus to formalise policy specification and analysis. In *Proceedings of the 4th IEEE Workshop on Policies for Distributed Systems and Networks*, 2003.
29. A.L. Lafuente and U. Montanari. Quantitative mu-calculus and ctl defined over constraint semirings. *Electronic Notes on Theoretical Computing Systems QAPL*, pages 1–30, 2005.