Author Obliged to Submit Paper before 4 July: Policies in an Enterprise Specification

James Cole[1], John Derrick[2], Zoran Milosevic[1], Kerry Raymond[1&3]

1) Cooperative Research Centre for Distributed Systems Technology (DSTC), University of Queensland, 4072, Australia
2) Computing Laboratory, University of Kent at Canterbury, Kent, CT2 7NF, United Kingdom
3) CiTR Pty Ltd, PO Box 1643, Milton, Queensland 4064, Australia

colej@dstc.edu.au, j.derrick@ukc.ac.uk, zoran@dstc.edu.au, kerry@dstc.edu.au

Abstract. Specifying policies doesn't occur in splendid isolation but as part of refining an enterprise specification. The roles, the tasks, and the business processes of an ODP community provide the basic alphabet over which we write our policies. We illustrate this through exploring a conference programme committee case study. We discuss how we might formulate policies and show how policies are refined alongside the refinement of the overall system specification, developing notions of sufficiency and necessity. Policy delegation is also discussed and we categorise different forms of delegating an obligation.

Keywords: ODP enterprise viewpoint, enterprise policies, formalising and refining policies, delegation, case study.

1. Introduction

Enterprise distributed systems are increasingly requiring precise specification of policies that govern individual and collective behaviour of entities in the system. This is needed to support precise reasoning about the systems behaviour - in terms of constraints on their behaviour and also addressing the dynamic effects of changes in policies.

Precise enterprise specification is also needed to reflect the need for the separation of policy implementations from the implementation of the functional part of the system. The separation is required to allow for flexible changes of policies while preserving the main functionality of the system.

Our approach for specifying enterprise policies is based on the definition of policies and other related enterprise concepts as defined in the Open Distributed Processing (ODP) standards [1] [2].

The Reference Model for ODP (RM-ODP) defines an architecture for specifying and implementing distributed systems. Because the complete specification of any non-trivial distributed system involves a large amount of information, the RM-ODP uses multiple viewpoints to help structure the design and provide a basic separation of concerns.

In the enterprise viewpoint one can specify the purpose, scope and policies of a system and its environment separately from other aspects of its implementation.

In the RM-ODP, Foundations [1], a policy is defined as "a set of rules with a particular purpose" where "a rule can be expressed as an obligation, a permission or a prohibition". Further, permission is a prescription that a particular behaviour is allowed to occur. A prohibition is a prescription that a particular behaviour must not occur, and an obligation is a prescription that a particular behaviour is required.

These definitions of obligations, permissions and prohibitions are in style of the Standard Deontic Logic, in that they are impersonal (i.e. do not associate policy statements to the performers of actions) and that they are static (i.e. do not consider the temporal aspects for fulfilling obligations).

The new ODP Enterprise Language standard currently being standardised [3] provides an extension of the description of policies as stated in the RM-ODP to include personal statements, but has yet to address the temporal nature of obligations.

These extensions are introduced in the context of the RM-ODP community concept, as policies are created, changed and enforced by enterprise objects forming a community. The new ODP Enterprise Language standard also elaborates on the notions of policy assignment, including the concepts of delegation and authority. Some of the initial ideas of the new policy framework for ODP enterprise viewpoint can be found in [4].

The ODP Enterprise Language does not prescribe a particular template for the development of enterprise specifications, but rather identifies significant concepts and relationships that form the basis for structuring an enterprise language specification. Neither it does mandate a particular notation - this paper offers one such notation, which expresses policy concepts from the new ODP Enterprise Language standard.

For example, the concept of community is fundamental; it is the main structural element and reflects some grouping of people and resources in the real world. A grouping can be considered a community if it is formed to collectively achieve some objective. Communities are then specified in terms of:

- Roles (both within the community and in the environment of the community). Roles are an abstraction, enabling the community to be specified without regard to the particular entities (people, programs etc) which will fill those roles in some implementation of the specification. The roles within a community are those that are working towards the objective (i.e. the roles share community objective or are committed to some sub-objective), whereas the roles in the environment might not be committed to the community objective. For example, a business and its employees might be seeking to maximise the profit of the business, but the customers are not. Yet it would be impossible to specify the business without also specifying customers that form part of the environment of the business.

- Processes (a.k.a. business processes). Processes describe some commonly occurring sequence of tasks that contributes to the achievement of the objective (or a sub-objective). Processes are defined as a partial order of steps, each step comprising a task involving one or more of the roles. A task could be the customer placing an order with a salesman (an interaction of roles), or the warehouse delivering the goods. The ordering of steps in a process is generally determined by data flow (outputs must be produced before they can be input elsewhere) or by policies.
- Policies. Policies place additional expectations on the behaviour of roles within the community, and are perhaps the least well-understood aspect of ODP enterprise specifications.

In this paper we focus on the specification and nature of policies in the ODP enterprise viewpoint. We use a programme committee (PC) case study to analyse and refine policies and we are exploiting the policy notation proposed in [5][6] to express policies and their relationships. This notation (language) was developed for the specification of ODP enterprise policies. The notation is a combination of structured English and simple predicate logic, and was built on the top of the formal object oriented specification language Object-Z (to enable formal analysis of the policies to be carried out).

Note that the policies we specify in the enterprise viewpoint represent the policies in force in the enterprise under consideration. Whilst these policies clearly concern the implementation, they are not all necessarily directly representable or implementable in any system. The purpose of such policy specification is not necessarily to implement policies but to be explicit about the constraints within which the system will be constructed.

The purpose of the paper is twofold. Firstly we aim to develop a case study and show how the notation in [5][6] can be applied to a real example. Secondly, by developing the case study a number of issues arise which we subsequently discuss. These include the refinement of policies against a business plan and the notion of delegation.

The structure of the paper is as follows. In Section 2, we introduce our case study. Having establishing our framework for specifying policy in Section 3, we then develop the initial policies for our case study in Section 4. These policies are then further refined in Section 5. We document what we have learned about policies from our case study in Section 6, including a discussion on the ways in which obligation can be delegated. Finally we conclude in Section 7 with some open questions for future study.

2. Motivating Case Study

The case study we consider is that of the policies governing a conference programme committee (PC). This is an interesting example because most programme committees are very physically distributed and most of them use software and networking in some aspects of their work. Space limitations prohibit a complete presentation of the case study in this paper; only pertinent fragments are shown.

The starting point of the case study is the assumption that a programme committee is a community (formed within the larger context of a conference organising committee) with the objective of producing a papers programme and the conference proceedings. Typically, the organising committee will prescribe the general theme of the conference (e.g. DNA sequencing, policies in distributed systems), give some indication of the number of papers required (the PC is not free to develop a programme requiring more days than the venue is booked for etc), and usually set a deadline for providing the programme and the proceedings. For research conferences, there is the expectation that the papers in the programme will be the best available papers, where "best" combines a number of independent criteria: pertinent, original, significant, technically-sound, well-written, etc.

3. Framework for Specifying Policy

We begin by describing the basic set of premises that underlie the policies specified in this paper.

3.1. Policies Exist to Achieve an Objective

We observe that the objectives of a community can usually be expressed as a set of obligations with attendant permissions and prohibitions on how those objectives/obligations can be achieved. If the community arises through refinement of some role/entity in an existing enterprise specification, then its objectives will be the obligations that the initial role/entity were subject to. Such a community will also inherit any permissions and prohibitions to which the initial role/entity was subject to, and all of these policies (or their refinements) apply to the roles and processes of the new community.

This chain of refinement of policies parallels the refinement of an enterprise specification, ensuring that the atomic roles/entities of the system are bound by policies that collectively contribute to the overall objectives of the system. For example, the policy that a paper reviewer cannot review their own paper is a prohibition that contributes to the objective of a conference of high-quality papers.

3.2. Authority Derives from Community

In a lot of work on policies, the existence of a single global source of authority is often assumed. However, we view communities (as collections of objects cooperating towards an objective) as being the basis for all authority. By joining a community, an object undertakes to conform to the policies of that community, whether applied to the community as a whole or to its particular role within the community. Some communities may have specific roles with responsibility for the making and enforcing of policy, but the authority of such roles is delegated from the community itself. As an enterprise specification may include many communities (related to one another in a variety of ways), it follows that there are many authorities within the system and that individual roles/entities may only be subject to the communities of which they are a member.

Some communities are sub-communities of another community (e.g. the programme committee is a sub-community of the conference organising committee), and thus are subservient for policy purposes. However, other communities overlap in other ways, and their differing objectives may lead to policies on individual roles that are in conflict. For example, a paper reviewer may have conflicting obligations to review papers and do work for their employer. Also, the author of a paper seeks to achieve publication of their work, while the conference objective is to publish only high-quality papers.

Therefore every policy must include the community (or delegated authority within a community) from which policy is imposed. However, in most specifications, this community is obvious and not explicitly identified.

3.3. Scope of Policy Making

Turning to the policies we firstly note that it is easy to express any form of behaviour as a policy. However, it is not always useful to do so. There are a number of kinds of "non-policy" [4]:

- Imposing policies on roles outside the community. If they are outside the community, they are not committed to the objective of the community and hence are not motivated to conform to the policies of the community. For example, the policy that "Authors are obliged to submit their papers before the due date" is meaningless, as authors will obviously submit papers any time they like. Instead it is the programme committee that must be obligated to react correctly to the author's behaviour.
- Imposing policies on things that are subject to the laws of nature or mechanical operation. There is no point imposing a policy "The author is obliged to write the paper before it is reviewed" since clearly the paper could not have been

reviewed until after it was written. Policy-making only makes sense when the role subject to the policy possesses the potential for some free choice regarding the performance of the action discussed in the policy.

- Imposing policy that could equally well be expressed as rules in some other part of the specification. For example, the format prescribed for submitted papers (e.g. font, word processor etc) is better elaborated in an information specification than in a policy. It may be appropriate to make a policy enabling the PC to reject a paper that is improperly formatted, but it is not the purpose of policy to elaborate the meaning of properly-formatted.

4. Formalising the Policies

With this framework in mind, we begin to formalise our policy statements, which describe prohibitions, permissions and obligations in terms of actions of particular roles in our community (i.e. the PC). To simplify the presentation, we will introduce the actions as we proceed with the formalisation of the policies.

In this section we show how the policy language described in [5] [6] can, with some extensions, be used to formalise the policies in our case study. The formalisation also brings to light a number of issues that we discuss below.

We begin by identifying an action used in formulating the initial policy statements. This is the publish action, and is used with a parameter, allowing us to write “publish(programme)” and “publish(proceedings)”, with the obvious intended meanings.

In addition #programme denotes the number of papers in the conference programme, and format(proceedings, Z) is a boolean condition which is true if and only if the proceedings are in a format acceptable to Z (which could be the publishers or the conference format).

Our initial policy statements can then be given as:

```
[[1]]  A PC is obliged to do publish(programme) before X where
       (30 <= #programme <= 40)
[[2]]  A PC is obliged to do publish(proceedings) before Y where
       format(proceedings, IEEE)
```

Here X and Y are dates such that X < Y and represent the appropriate deadlines.

As we can see this syntax of our policy language is fairly simple (the reader is referred to [5][6] for full details). The general format is:

```
[[label]]A <role> is <modality> to do <action> ...
```

where the modality is one of “permitted”, “prohibited” or “obliged”.

All modalities may refer to the execution of an action. The action denotation consists of an action name followed by optional parameter specifications in brackets, e.g. proceedings is a parameter to the publish action in policy [[2]] above. There are a number of possible clauses after the action denotation, as described below.

We also observe that obligations are often associated with deadlines, either absolute or relative. Often these deadlines are implicit, but we recommend that they be made explicit wherever possible. If there is no deadline associated with an obligation, then it may be difficult to test whether the obligation has been violated ("I'll do it real soon now!").

Obligations thus have a before-clause, which contains a condition upon which the obligation should have been fulfilled, and both policies [[1]] and [[2]] above illustrate this form. Similarly prohibitions can be qualified by an until-clause and permissions by an after-clause.

Where-clauses (e.g. "where 30 <= #programme <= 40") are used to constrain the value of parameters of an action, and conditionality is captured in an obvious fashion by an if-clause.

5. Refinement of Policies

One observation that came about in considering the PC case study is that policy is expressed over some set of roles and some set of actions. Therefore, policy can only be refined/evolved as part of the overall refinement/evolution of the community; it is not something that can be developed independently, in spite of the roles and the actions that constitute the business processes. We observe that in real life, many policies are routinely ignored because of the perception that changing circumstances have made them redundant. For example, in Queensland until recently, cars had to be preceded by a man waving a red flag and ringing a bell to avoid frightening the horses, a law uniformly ignored by motorists and police alike.

In order to satisfy the policies [[1]] and [[2]] above, we refine them further. There are often many ways to refine a community into a set of roles, processes and policies to meet the objective. The choice of roles and processes made determines the exact structure of policies introduced. In the PC example, we have chosen to establish a process that issues a public Call for Papers (as opposed to soliciting papers from a closed set of individuals). Hence we must develop a set of policies based around the issuing of the Call for Papers. This set uses the following base actions:

- publishCFP(CFP, widely) - the call for papers is to be sent out to a wide audience.
- submit(paper) - submit a paper to the conference.
- receive(paper) - receive the paper
- refuse(paper) - refuse the paper (as distinct from reject the paper, see below), the opposite of submit
- notify(author, paper,outcome) - notify an author about whether the paper has been accepted.
- revise(paper) - revise the paper.
- reject(paper) - reject the paper from the conference.

The policies below introduce further deadlines D1 < D2 < D3 < X and D3 < D4 < Y, where:

- D1 - last viable date for issuing a call for papers
- D2 - advertised deadline for submitting papers
- D3 - notification date of paper acceptance
- D4 - date that the proceedings goes to the printers
- X - date of publication of programme
- Y - date of publication of proceedings

The following [[a]] to [[g]] are imposed by the PC community (the authority), whereas the initial policies [[1]] and [[2]] were imposed by the conference organising committee.

[[a]] A PC is obliged to do publishCFP(CFP, widely) before D1

Notice how the subjective notion of advertising the CFP widely has been pushed into the action description to avoid the policy statements being subjective, e.g. writing it as "A PC is obliged to do publishCFP (CFP) widely before D1". That is, "widely" has been used as action parameter rather than a qualification on the policy. Indeed, as we see here, it is possible to refine some or all parts of a policy into the selection of specific actions (or processes), thus effectively "programming" the policy.

[[b]] A PC is obliged to do receive(paper) before D2 where format(paper,conf)

[[c]] A PC is permitted to do refuse(paper) after D2

It is interesting to note that [[c]] was not expressed as "A PC is prohibited from receive(paper) after D2". Refusal of a paper is a real action that occurs (e.g. an e-mail is sent to the author explaining the deadline has past) whereas the prohibition would simply ignore the author's attempt to submit the paper.

Similarly, [[c]] was not expressed as "A PC is obliged to refuse(paper) after D2" because the intention is to permit the PC to accept papers after D2 if they wish. This is expressed in the following policy:

[[d]] A PC is permitted to do receive(paper) after D2

Space prohibits the elaboration of the business processes of the PC but it should be noted that the actions of receive(paper) and refuse(paper) are alternative actions in response to a submit(paper). Remember that there are no policies directly on submitting, as the author is outside of the PC community.

However, no conference can continue to receive papers indefinitely. Given the business processes of the PC, it is not appropriate to receive a paper after notification has been sent to the authors:

```
[[e]]  A PC is prohibited to do receive(paper) after
       time(notify(author,paper,outcome))
```

where time returns the time at which a specified action occurs. As discussed further below, this illustrates how this policy language supports the specification of dynamic policies (see Section 6.2).

```
[[f]]  A PC is obliged to do notify(author, paper,outcome) before
       D3
```

In order to deal with violation of an obligation we introduce an otherwise clause. This clause details the policies applicable if the initial obligation is not met. Otherwise clauses can be nested, an example of which we will see below. First here is a simple example:

```
[[g]]  An AcceptedAuthor is obliged to do submit(revise(paper))
       before D4 otherwise a PC is permitted to do reject(paper)
```

Notice how the action of notifying an author that their paper has been accepted has a side-effect of introducing that author into the PC community as a new role within community called AcceptedAuthor, and hence subject to the policies of the community. (Note that the author's membership of the PC community does not imply that the author is a "PC member" which is a distinct role within the PC community.)

We can now talk about the relationship between the policy statements [[a-g]] and the original statements [[1]] and [[2]], and in particular whether the former are *necessary* or *sufficient* with respect to [[1]] and [[2]]. In fact we can only talk of necessity against a particular business plan, thus with respect to the plan of receiving and reviewing papers (i.e. the normal conference plan) the policies [[a-g]] are necessary. With the additions of actions to actually print proceedings these policies would also be sufficient, i.e. if [[a-g]] were met then so too would be [[1-2]].

The policies [[a]] through [[g]] above came about primarily through refinement of the community's objectives relative to its business process. Let us now consider policy refinements relative to the partitioning of the behaviour of a community into a set of roles. Let us look at the policy [[a]]. To refine this we consider roles PCChair and PCMember within the PC. In order to publish the CFP we need policies concerning the drafting and approval process. To do so we consider the following additional actions:

- draft(CFP) - the action to make an initial draft of the CFP
- approve(CFP) - the action to approve the CFP
- amend(CFP) - the action to amend the text of the CFP
- distribute(CFP) - the action to distribute the CFP, perhaps repeatedly,

and an additional condition:

- approve(PC, CFP) - a condition true if a majority of the PC have approves the CFP

and a new deadline:

- D0 - the deadline for drafting the CFP.

We then refine [[a]] as follows.

[[a1]] A PCChair is obliged to do draft(CFP) before D0

[[a2]] A PCChair is prohibited to do publishCFP(CFP) until approve(PC, CFP)

[[a3]] A PCMember is obliged to do approve(CFP) otherwise a PCMember is obliged to do amend(CFP) otherwise PCChair is permitted to do satisfy(approve(PC, CFP))

The action "satisfy" asserts the truth of a condition.

[[a4]] A PCChair is obliged to do publishCFP(CFP) before D1

[[a5]] A PCMember is obliged to do distribute(CFP) before D2

[[a6]] A PCMember is prohibited to do distribute(CFP) until a PCChair does a publishCFP(CFP)

6. Observations

In this section, we discuss some of our observations arising from our case study.

6.1. Default Policies

When one considers policies [[a5]] and [[a6]], it begs the question of whether a PCMember who is obliged to distribute the CFP actually has the permission to do so. Clearly, there are circumstances when this is a prohibited behaviour. Therefore, should the set of policies include "A PCMember is permitted to do distribute(CFP)" or similar? If we take this view, then we would also require permission for the PCChair to prepare a draft CFP, for PCMembers to approve the draft, etc. Clearly this would be tedious for both the reader and the writer of specifications. This suggests the need to establish some sensible default policy. There are a number of ways this could be done:

- When introducing the actions in the specification, indicate whether the default policy is permission or prohibition, based on the “sensitivity” of the specific action
- Through inheritance of default or explicit policies in outer communities
- Through the assumption that role-action pairs forming part in the business process are permitted
- Through the tagging of the community as being fundamentally permissive or restrictive

Further work is needed to clarify the best approach.

6.2. Dynamic Policies

Another observation that we found in the case study is that some policies seemed to alter other policies already in force. Consider policy [[e]]:

```
[[e]]  A PC is prohibited to do receive(paper) after
       time(notify(author,paper))
```

This policy illustrates how policies can come into force based on occurrence (or non-occurrence) of run-time events. Therefore, it may not be possible to determine in advance precisely which policies will be concurrently in force, and hence not be feasible to determine in advance if policies are in conflict.

There are other kinds of dynamic policies. For example, the programme committee is obliged to receive papers if submitted by the due date D2, permitted to receive papers if submitted after the due date, and prohibited from receiving papers after notification of the authors. However, the programme committee chair is permitted to postpone the due date D2 if insufficient papers have been received:

```
[[p]]  A PCChair is permitted to do extend(D2) if (#papers < 30
       and D2 < now[1] < D3)
```

Postponing the due date D2 impacts on all of the policies relating to the acceptance of papers relative to D2, changing the set of policies in force.

It also creates some interesting scenarios if some papers were not accepted under the policies embodying the original due date, but which must be accepted if assessed under the policies using the revised due date. In other scenarios, the reverse situation might apply; actions were permitted under the original policies that would be prohibited under the new policies. In either case, should there be some attempt to “roll back” the actions that have or have not occurred due to the old policies and then replay them under the new policies?

---

1. Notice how some policies are established within a time range, and to model this we use “now” which stands for the current time. It is typically used within an if-clause as this policy illustrates.

Or should all policy decisions be judged solely by the policies that applied at the time? Or should the roles involved in the policy be allowed to choose which policies (as a set or individually) apply to them "give them the benefit of the doubt". There is no hard-and-fast rule in real-life, and indeed, for resolution, one must look to the objectives from which the policies (original and revised) were refined for guidance. In the case of the PC, a paper which was initially refused due to its late submission would probably be included by applying the new rules (presumably with the knowledge and consent of the author) since the original objective was to select the best N papers, and so clearly more papers to chose from increases the overall likelihood of a high-quality conference.

There would seem to be scope for applying action-reversal or compensatory actions (where actions cannot be reversed) that are sometimes seen in transaction management systems.

6.3. Delegation of Obligation

Obligations are often delegated. In the PC example, a PCMember is obliged to review some papers.

[[q]] A PCMember is obliged to do review(paper) before D3.

However, the PCMember may wish to delegate this duty to a colleague. One way that this delegation might be formalised is to introduce the following 2 policies:

[[r]] A PCMember is permitted to do introduce (Reviewer, Colleague).

[[s]] A Reviewer is permitted to do review(paper) where not conflict-of-interest(Reviewer, paper).

The action introduce creates a new Reviewer (a role within the PC community), filled by the colleague. However this begs the questions:

- Is the Reviewer obliged to review the paper?
- Is the PCMember still obliged to review the paper?
- Does the PC need to consent to the delegation?

In exploring the nature of delegation of obligation, we found four different forms of delegations, which we named transfer, sharing, splitting and outsourcing. To understand the differences, we need to make explicit the authority imposing the policy. Figure 1 illustrates authority X obliging role Y to do action A (block arrow shows the 'direction' of this obligation, while the thin arrow depicts the role performing the action).

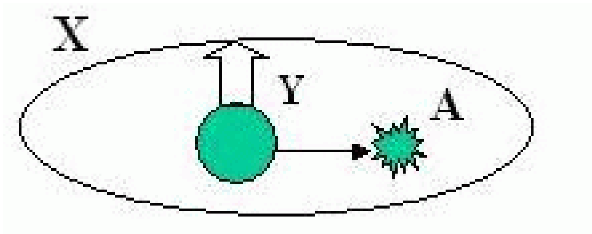

Figure 1. Original Obligation

6.3.1. Transfer of Obligation. A transfer of an obligation occurs if the original obligation is fulfilled by the creation of the new obligation. If original obligation is "X obliges Y to do A", then the transferred obligation is "X obliges Z to do A". Since X is the judge of the fulfilment of an obligation, it follows that X must consent to the transfer. Following the transfer, Y is no longer obligated. X will not regard Y as having failed if A does not occur. Figure 2 illustrates the consequences of transferring the obligation from Y to Z.

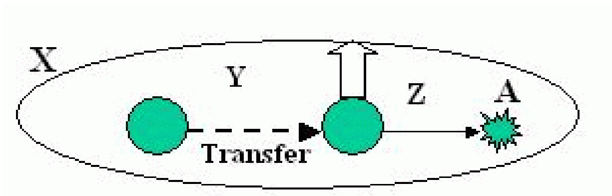

Figure 2. Transfer of Obligation

6.3.2. Shared Obligation. A shared obligation occurs when the obligation is extended to oblige an additional participant. That is, "X obliges Y to do A" becomes "X obliges Y and Z to do A". This implies that Y and Z collectively have the obligation to do A, the breakdown of responsibility between Y and Z is not specified. However, if A does not occur, X will consider that both Y and Z have failed (even if one of them carried out their allocated part of A). X's consent is required to share an obligation. Figure 3 illustrates sharing an obligation.

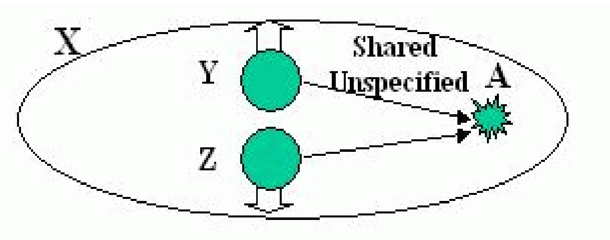

Figure 3. Sharing an Obligation

6.3.3. Split Obligation. A split obligation occurs when the action required is split into two (or more) parts. That is, "X obliges Y to do A" becomes "X obliges Y to do A1 and X obliges Z to do A2" (where A1 and A2 are some partitioning of A). If A1 occurs and A2 does not, then only Z is deemed to have failed by X (and vice versa). However, it should be noted that if A1 contains dependencies on A2, then Z's failure to complete to A2 may cause Y to fail to complete A2. Splitting requires the consent of X. Figure 4 illustrates splitting an obligation.

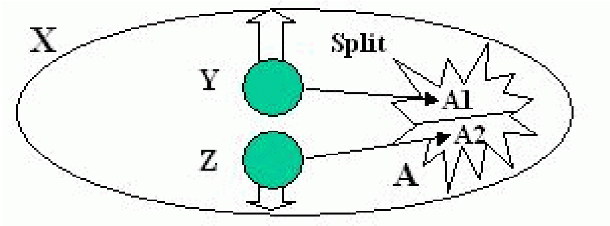

Figure 4. Splitting an Obligation

6.3.4. Outsourced Obligation. An outsourced obligation occurs when the obligee[2] obliges another to carry out all or part of the required behaviour. That is, "X obligates Y to do A" becomes "X obligates Y to do A and Y obligates Z to do part/all-of-A". If A does not occur, then X deems Y to have failed. If A did not occur because of Z's failure to carry out its actions, then Y may deem Z to have failed, but X will not make any judgement regarding Z, as X had no relationship with Z. As far as X is concerned, the full responsibility rested with Y. Outsourcing does not require X's consent. Indeed, outsourcing does not usually require the new obligee to be within the community. Figure 5 illustrates outsourcing of an obligation.

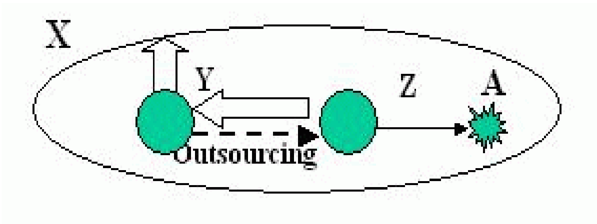

Figure 5. Outsourcing an Obligation

Note that the outsourcing of obligation is not meaningful if obligation on Y to do A implies that Y itself must do action A. In discussing outsourcing, we are making the assumption that the obligation to do A is in fact an obligation to ensure that A is done.

2. Yes, "obligee" *is* a real word; it means "one who is obliged".

In real life, the notion of obligation appears to include both interpretations, e.g. someone else may pay a traffic-offence fine for you, but may not go to jail for you - the fine can be outsourced, the jail term cannot.

6.4. Delegation of Permission and Prohibition

If obligations can be delegated, it invites the question of whether all policy can be delegated.

If the authority X is the only source of permissions to do A, then Y's permission to do A would appear to be distinct from Z's permission to do A. Also, there would appear to be little motivation for delegation, since, unlike obligation, permission is not personally onerous.

Similarly, there appears to be no meaningful interpretation of delegation of a prohibition. If X prohibits Y from doing A, then Z is free to decide to not do A without the involvement of X or Y.

In real life, permissions and prohibitions are often associated with obligations, often enabling (or preventing) specific sub-actions associated with the obligation. For example, the PC Chair is obliged to distribute the Call for Papers, and is permitted to use the Internet to do so but prohibited from printing paper brochures for that purpose. In such situations, the delegation of the obligation to another would seem to require that the permission and prohibition be similarly delegated.

6.5. Actions, Alternative Actions, and Inactions

As noted in Section 5, some actions are mutually-exclusive. When an author submits a paper, the PC may choose to receive it for reviewing or refuse it but not both! Therefore, the prohibition on one of these actions (in some circumstances) becomes an effective obligation to do the other. Similarly, an obligation to do one action is an effective prohibition on the other. Is it necessary to state both policies, or is it reasonable to allow one to be derived from the other? Or is this simply a matter of style?

The other alternative to an action may be inaction, and indeed, in real life, deliberate inaction is often used as a response to an awkward situation. Should inaction be the subject of policy-making too? Could the PC be permitted to do no-action in response to a paper submitted after the conference (say)? Arguably, "inaction" is effectively an action, since it involves someone performing the action of deciding to "do nothing" at some level of refinement.

6.6. One Action or Many Actions?

We also observe that cardinality of actions is an issue in policies. While prohibitions prescribe an absence of action, it is generally assumed that permission to do an action is a permission to do the action many times. With obligation, the expectation is that the action is done once, unless it is a standing obligation, in which the action is expected to be repeated continuously (possibly in response to some stimuli). For example, in policy [[a5]] we might wish that the 'distribute' action is repeatedly executed, whereas in policy [[a4]], we clearly want the 'publishCFP' action to occur just once. This is an issue that was not elaborated in our case study.

6.7. Policy on Actions or Policy on State?

The discussion of delegation of policy (see Sections 6.3 and 6.4) opens up the issue of whether policy should be made only about the performance of actions or whether policy should also be made about achieving a certain state of affairs, either through their own actions or the actions of others (equally their own inactions and the inactions of others). For example, obliging the PC to remain within budget is a significant constraint on their behaviour but not one that requires any specific kind of action to be performed.

6.8. Conflicts and Completeness

Policies are often in conflict. Although we do not have space to explore this issue here, one of the purposes of using a formal language for specifying policies is that it opens a way for conflict and consistency analysis to be applied. [5][6] discuss this issue briefly in the context of the translation of the policy language into the specification notation Object-Z.

Tools also have a role here, both in terms of conflict analysis and also in terms of change management in the presence of large scale policy specifications. The interested reader in this topic is referred to [7].

It is also difficult to judge whether a set of policy statements is complete, suggesting a need for policy simulation tools to be used for practical observation of a set of policies in action.

7. Conclusions and Open Questions

Specifying policies in a real-world case study turned out to be harder than we expected. Our experiences have lead us to a better understanding of the framework in which we write policies. Specifically, we have learned:

- that policies must be refined with respect to a business plan
- that using a policy notation is very useful in uncovering a number of subtleties in the policies themselves
- that default policies are necessary to avoid tedious and repetitious specifications
- that there are various forms of delegation of obligation

However, our real-world case study has left us with a number of open questions:

- how we should best specify default policies
- what linguistic support is needed to reflect the dynamic interaction of policies
- the interpretation of delegation of permissions and prohibitions

In many ways, our case study has asked more questions about policies in enterprise specifications than it has answered. However, it demonstrates the real need for extensive research into these issues by communities such as this workshop.

Acknowledgements

John Derrick gratefully acknowledges receipt of a University Of Queensland Visiting Scholars Travel Grant.

The work reported in this paper has been funded in part by the Co-operative Research Centre for Enterprise Distributed Systems Technology (DSTC) through the Federal Government's AusIndustry CRC Programme (Department of Industry, Science & Resources).

References

[1] ISO/IEC JTC1/SC21, "Basic reference model of open distributed processing, part 2: Descriptive model," ITU-T X.902 - ISO/IEC 10746-2, Aug. 1994.

[2] ISO/IEC JTC1/SC21, "Basic reference model of open distributed processing, part 3: Architecture," ITU-T X.903 - ISO/IEC 10746-3, 1995.

[3] ISO/IEC JTC1/SC7/WG17N0106, "Information Technology - Open Distributed Processing - Reference Model - Enterprise Language," ISO ISO/IEC 15414 | ITU-T Recommendation X.911, Jan. 2000.

[4] P. Linington, Z. Milosevic and K. Raymond, "Policies in Communities: Extending the ODP Enterprise Viewpoint". Proc. 2nd IEEE Enterprise Distributed Object Computing Workshop, San-Diego, Nov.1998.

[5] Formalising ODP Enterprise Policies. M.W.A. Steen and J. Derrick. In 3rd International Enterprise Distributed Object Computing Conference (EDOC '99), University of Mannheim, Germany, September 1999. IEEE Publishing.

[6] ODP Enterprise Viewpoint Specification, M.W.A. Steen and J. Derrick. In Computer Standards and Interfaces (to appear).

[7] E. Lupu and M. Sloman, "Conflicts in Policy-Based Distributed Systems Management". In IEEE Transactions on Software Engineering, 25(6): 852-869, Nov. 1999.

[8] E. Lupu, M. Sloman, N. Dulay, N. Daminaou, "Ponder: Realizing Enterprise Viewpoint Concepts", Proceedings, Fourth International Enterprise Distributed Object Computing Conference, Japan, 2000.