

Policies in Communities: Extending the ODP Enterprise Viewpoint

Peter Linington¹, Zoran Milosevic², Kerry Raymond²

1) *Computing Laboratory, University of Kent at Canterbury, pfl@ukc.ac.uk*

2) *CRC for Distributed Systems Technology (DSTC), {zoran, kerry}@dstc.edu.au*

Abstract

The Reference Model of Open Distributed Processing (RM-ODP) introduces the notion of an enterprise viewpoint and provides a minimum set of concepts for structuring enterprise language specifications. This paper extends the RM-ODP enterprise concepts by exploring how policy can be modelled within and between communities. A model for enterprise behaviour based on physical and social actions is presented.

1. Introduction

The RM-ODP framework is increasingly being used for modelling complex open distributed systems, such as those in the domains of telecommunications, finance, education and defence (e.g. C4I environments). While some of the ODP viewpoint languages, in particular computational and engineering are developed in sufficient details to describe programming and infrastructure artifacts of any distributed system, this is not true of the enterprise language. On the other hand, the availability of maturing distributed infrastructure platforms such as CORBA, DCOM, DCE and Java-RMI increasingly encourages the use of distributed objects for business applications. As a result, the IT community is shifting its interest from platform issues towards enterprise specifications. There is an increasing demand from industry to use enterprise specifications to improve the accuracy of the design of distributed systems, in particular those that cross various administrative and organisational boundaries.

The existing ODP enterprise language, consisting of a limited number of concepts and structuring rules, needs further extensions and refinements in order to be better suited for enterprise modelling of practical open distributed systems. For example, there is a need for rigorous specification of policies governing the behaviour of complex systems and automated sub-systems. These policies need to be made explicit because their monitoring and enforcement will require actions by the system implemented, and the correctness of these actions can only be guaranteed if there is a well defined framework for the description of concepts such as ownership, right, objective, authority, delegation and policy.

This paper seeks to expand on the Enterprise Viewpoint of the Reference Model for Open Distributed Processing, particularly from Part 2 (Foundations) [7] and Part 3 (Architecture) [8] by drawing together recent developments in a number of research areas, and hopes to contribute to the work on the extended enterprise language currently under development within the ISO [9].

Specifically, this paper focuses on a subset of the enterprise concepts, namely on the notions of community and policy (permissions, prohibitions and obligations), and endeavours to provide a more precise framework for reasoning about these fundamental enterprise concepts. It aims to establish a common vocabulary and structure for use by enterprise notations, allowing the unambiguous specification of enterprise requirements.

Section 2 begins by exploring the applicability of the existing ODP enterprise language for describing complex ODP systems which involve the behaviour of people and human organisations. Section 3 follows with a summary of the latest results from deontic logic that are relevant to the concepts of obligations, permissions and prohibitions. Much of the ODP modelling experience is focussed on the computational viewpoint, and Section 4 outlines why the enterprise viewpoint needs a different modelling approach, resulting in the derivation of models of individual enterprise objects (Section 5) and communities of enterprise objects (Section 6). Conclusions are presented in Section 7.

2. The Enterprise Viewpoint

Most specification activities which use the ODP viewpoints begin with the enterprise specification in order to establish the purpose, scope and policies of the ODP system [8]. This must be a very abstract and flexible specification if it is to provide guidance throughout the growth and evolution of the system, and must not over-constrain the other viewpoints. However, it must state the policies to be supported clearly and unambiguously.

Before policies can be expressed, the specification must define the specific vocabulary and structure of the enterprise. It does this by defining a number of *communities* and roles within them, and in stating the ways in which these communities are related.

The ODP concept of a *community* is a “configuration of objects formed to meet an objective ... expressed as a contract which specifies how the objective can be met” [8] where a contract is “an agreement governing part of the collective behaviour of objects” [7]. This collective behaviour is expressed in terms of *roles*, where each role identifies some subset of the overall community behaviour that can be meaningfully performed by a single object within the community.

The concept of a *role* is sufficiently general to specify the behaviour of entities which can be either (parts of) IT systems or people. The specification of a system in terms of the key roles and their relationships enables a stable specification in spite of the facts that the roles may be filled by enterprise objects of a wide variety of natures and that their nature can evolve over time [18]. For instance, a role in a system which was initially allocated to a person can, at a later stage, be filled with an automated entity. Such an entity will provide the same functionality as a person, perhaps with a better performance, but reduced flexibility, and often with lower costs to the overall system. Numerous examples from different domains illustrate this point, e.g. telephone banking services and automatic teller machines in banking, robots in manufacturing, automated pilots in the aircraft industry. In spite of the fact that automated systems increasingly replace people in their traditional roles, the description of the system in terms of roles and their relationships often remains unchanged.

The ODP enterprise language concepts of *obligations*, *permissions* and *prohibitions* can be used to express *policy* statements associated with the roles and the enterprise objects filling them; these policies may be a constraint on, or an extension of, the original behaviour. Policies typically reflect some social norms, such as legal norms from the underlying jurisdiction domain, management norms emanating from the rules of a particular organisation or the norms adopted by various non-formal social groups. As such, policies introduce terms and rules of the corresponding background language, e.g. normative concepts of business contract law and ethical or privacy codes of medical practitioners.

In general, policies apply to either the roles or the enterprise objects that can fill them. They apply to the roles in cases where the roles describe some generic behavioural patterns pertinent to the description of the role itself, such as a CEO role, or a role describing obligations of medical doctors in a specific community. On the other hand, policy statements can apply to enterprise objects which fulfil roles, such as those related to the owners of real-estate properties, driver licence holders and so on. They can also apply to the process of assigning enterprise objects to the roles.

There are certain policy statements that can be fully supported within the underlying infrastructure. Examples are those policies which reflect some of the unambiguous rules of a background community, e.g. business contract laws in most western countries. These policies can be implemented and, to some extent, enforced within an ODP system [12].

There are other kind of policies that should take into account the actions of enterprise objects which model people. Such policies have to be expressed in terms of necessary constraints, leaving a great deal of freedom in determining the actual behaviour of the people concerned. In general, people can introduce elements of hidden internal behaviour or act in an opportunistic way to maximise their rewards based on their personal objectives. These characteristics of people, coupled with imperfect information about them, lead to uncertainty with respect to their actions. Economic agency theory is a class of game-theoretic approaches that can be used to design the optimal strategy of agents in the presence of uncertainty. The primary means for people to increase certainty in an otherwise uncertain world is to enter into contracts and to join communities, as contracts and communities specify an agreed behaviour among a set of enterprise objects. The design of optimal contracts is described in [13], which also presents an infrastructure for contracts and contract enforcement.

Therefore, there are two aspects to behaviour when considered from the enterprise viewpoint. One is an object’s inherent behaviour, reflecting its internal objectives as well as its physical capability to undertake actions and interactions. Another aspect of its behaviour is when the object is considered as part of a community (i.e. the object fills a role within a community), through which the object acquires additional objectives and becomes subject to additional policies. The ODP enterprise concept of *community* is introduced to model how a group of enterprise objects achieve their individual objectives through commonly agreed patterns of interaction. This agreement is institutionalised through the *contract*. If an object wishes to participate in a community, it is obliged to comply with policies set out in the contract of that particular community. This implies a need for an enterprise object to modify its internal behaviour according to the policies of that community. We refer to the resulting object’s behaviour as its social behaviour.

3. Deontic Logic

The concepts of permission, prohibition, and obligation are formally studied in deontic logic. This logic is suitable as a starting point for the specification of any normative system, i.e. a system in which the behaviour of interacting agents (be they people or automated systems), is governed by a set of norms. The concepts of obligations, permissions and prohi-

bitions are often referred to as deontic statements about a system. In the context of an ODP specification, deontic statements express a deontic structure of a system, which should be stable irrespective of the underlying infrastructure. Consider for example, a statement of an enterprise prohibition on a particular group of users from reading some files. Although this enterprise (deontic) statement can be true, the infrastructure may fail to enforce it, e.g. an access control can be imperfect or disabled.

3.1. Problems with using Deontic Logic

RM-ODP [7] bases its definitions of permission, prohibition, and obligation on standard deontic logic. However, standard deontic logic considers only a static picture and does not take into account the interactions of agents, as discussed in [4]. These limitations of the standard deontic logic present significant difficulties for the needs of practical enterprise modelling and lead us to consider some new developments in the fields of deontic logic, agency and normative systems.

3.1.1. Static versus Dynamic

Most of the systems of deontic logic consider a very *static* picture in terms of the relationships (i.e. logical connections) between co-existing obligations and prohibitions [3]. While this can be a good approximation for prohibitions, as they often arise from general ethical principles or standing laws, this is not satisfactory in the case of obligations, which have a more dynamic nature. In fact, most obligations¹ require to be discharged at certain points in time and new obligations can be undertaken or be imposed (through some external authority) at various points in time. These new obligations can also introduce conflicts with the existing obligations and these need to be resolved.

3.1.2. Impersonal versus Personal

In addition to its static nature, standard deontic logic centres around *impersonal* statements such as ‘it ought to be that ...’. These impersonal statements do not capture the distinction between “ought to be” and “ought to do”, which requires the introduction of an explicit concept of an agent carrying out a responsibility. In contrast, personal statements clearly identify the agent which “ought to do”, yielding a more precise set of policies applicable to each enterprise object. A general review of problems with the interpretation of deontic logic (written for computer scientists) is presented in [11]. These are real prob-

¹Apart from the so-called standing obligations which can never be fully discharged [3].

lems, and require a powerful and flexible model to meet the uncertainties in the real world that give rise to the problems.

3.2. A Model for Obligation

Recent results from deontic logic, combined with the logics of action, agency and ability, enable better description of deontic statements that correspond to the real world situations. This section summarises some of these results that we find relevant to enterprise modelling.

The dynamic nature of obligations can be analysed through the notion of forward branching time, that models indeterminism of the future and determinism of the past [6]. The moments in time are ordered into a tree like structure with forward branching representing the indeterminism of the future and the absence of backward branching representing the determinacy of the past, as illustrated in Figure 1. In other words, this tree represents a set of

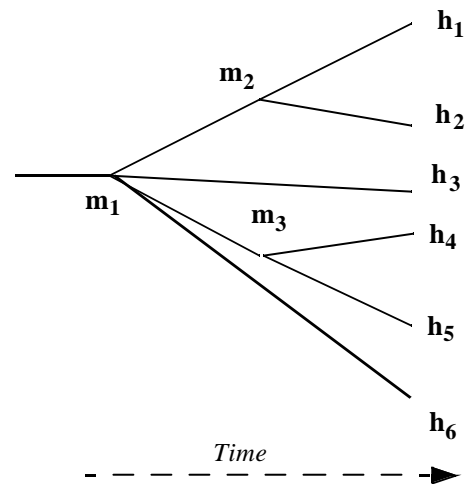


Figure 1. Forward-branching time

possible worlds in future and also a complete temporal evolution of the world. In such a tree, a history h represents a maximal set of linearly ordered sets of moments, a single complete branch of the tree. It is shown that in evaluating formulas (e.g. about an action A of an agent) against the branching temporal frames, one needs to take into account both a moment together with a history through that moment. The branching tree defined in this way is suitable for expressing dynamic but impersonal deontic statements (i.e. the statements which do not involve agents).

When supplemented with the model of actions of agents, the tree structure can be exploited to model their action choices, based on von Neumann’s theory of games [14]. These choices reflect a set of possible alternatives for actions of agents, thus enabling expressions of personal deontic statements. This allows modelling of influence that agents are able to

exercise upon the course of history, i.e. it allows modelling of their actions or choices at a given moment. Formally, by acting at moment m , an agent a can select one of its choice sets, S_i of the histories through a moment m . For example, at m_1 , an agent a can choose one of the choice sets S_1 , S_2 or S_3 , as illustrated in Figure 2. If agent a chooses S_1 , it can

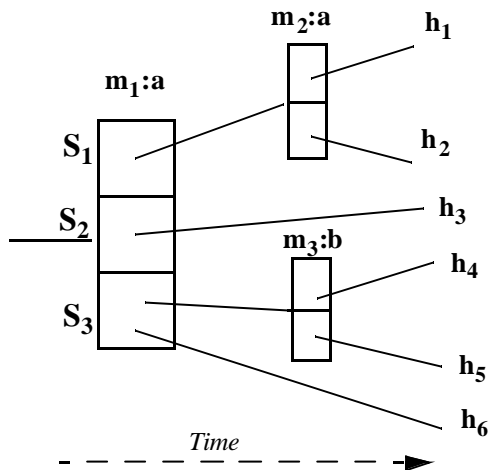


Figure 2. Action choices

then face another choice of actions at m_2 , leading to the realisation of one of the histories, h_1 or h_2 . Further, if any of two histories are undivided at a moment, they must lie within the same choice set. Although histories h_4 and h_5 divide at m_3 , agent a has no influence on the choice, which is performed by agent b . In some circumstances, choices are made by the environment (nature), and none of the agents can determine the outcome.

With this model as the foundation, a deontic formula for a permission, prohibition or obligation partitions the paths through the tree into paths that satisfy the deontic statements and those that do not.

Finally, this tree structure can be further enriched to model priorities or weights that agents assign to their different choices. For this, a value (or utility) function can be associated with each of the histories and, if there is probabilistic information related to various outcomes of selecting an action, the agents should select a strategy which maximise their expected utility function. This function is the sum of the values of the various histories of a choice set, weighted by the probability assigned to each history [6]. If these probabilities are not known (which is often the case in real life situations), a dominance ordering, based on a game theoretic approach for dealing with choices under uncertainty can be applied [6]. This interpretation of expected behaviour in terms of perceived value can be related to the 18th and 19th century concept of utilitarianism [17]. Note that the value function reflects both the desirability of the ultimate outcomes and the current set of obligations.

It is important to note that the above model is more suitable for modelling normative systems than the standard deontic logic systems. It supports the dynamic nature of deontic statements and also allows personal deontic statements, those that are more closely related to the actions of agents. The framework was initially proposed by Belnap and Perloff [2] and further contributions were made by Horty [6] and Brown [3]. This framework provides a foundation for the terms and grammar of the so called *stit* sentences, namely the sentences of the form ‘see to it that’. It is argued in [2] (and the series of papers based on this) that the statements of the form $[a \textit{ stit}: A]$, where a is an agent and A is an action statement, are the most suitable linguistic construct for describing alternatives and choices which assigns an action to an agent. This formulation is a development from classical speech act theory [1] and, which by making agency more explicit, makes the structure of obligations clearer. A computational technique for making agency explicit is given in [10] in which policy statements are expressed in terms of both subject and target. Further description of the *stit* semantics and grammar is beyond the scope of this paper.

Therefore, while standard deontic logic may be satisfactory for impersonal oughts, it is not sufficient for the descriptions of situations in which actions are attributed to agents. These situations need to be described when reasoning about the agents doing actions in a social context and are needed to specify enterprise objects filling roles in a community.

The model of temporal frames, augmented with the notions of choice sets and utility functions, provides a foundation for deriving a framework for a more elaborate semantic model of permissions, prohibitions and obligations than is currently included in ODP standards. Section 5 provides such a framework which can serve as a reference model for interpretation of personal deontic statements in the ODP enterprise language.

4. Enterprise Modelling is Different to Computational Modelling

All of the above suggests that the enterprise modelling of concrete ODP systems involves issues which are quite different from computational modelling, which is well-understood within the ODP community. The existing ODP enterprise concepts and structuring rules related to the concept of community [8] need to be extended and refined to provide a framework for modelling policies and their relationship with the behaviour of enterprise objects and communities.

4.1. Enterprise Behaviour

When considering enterprise behaviour (especially when human behaviour is involved), we need to adopt a more sophisticated model of behaviour than that used for computational behaviour. In theory, human behaviour is completely unpredictable. A person can choose to do (or not do) anything at almost any time. Attempting to express enterprise behaviour in a computational manner tends to result in a chaotic specification:

```
EnterpriseBehaviour ::=
  [any action I am capable of];
EnterpriseBehaviour
```

Yet enterprise behaviour is rarely chaotic in practice, because the entity being modelled *chooses* to constrain its behaviour in order to achieve its own objectives or to conform to the expectations of others. This ability to choose to constrain its behaviour in some way reflects the concepts of “free will” or “intention” of enterprise entities.

Thus the policy framework proposed here assumes that enterprise behaviour can be described at two levels:

- **physical behaviour** (a.k.a. inherent behaviour) which describes the widest (most permissive) view of the object’s behaviour, assuming it can perform any action it is capable of, constrained only by any internal programmed sequencing or physical realities. E.g. a person is capable of loading a gun and a person is capable of firing a gun, but the gun cannot be fired until it has been loaded.
- **social behaviour** which describes a subset of the physical behaviour, which is constrained by the decision of the object to meet its objectives or to conform to a community. E.g. a person might constrain themselves to load a gun only immediately before firing, either because of their own concerns or the concerns of their community about the dangers of leaving loaded guns around.

When considering type matching of enterprise behaviour, it must be recognised that only type matching of physical behaviour can be tested a priori. Physical behaviour matches a role without regard to the constraints placed on that role by the contract, just as, in the computational specification, type matching depends only on the types of the interfaces concerned, not on the broader view of the configuration in which they occur. If an enterprise object possesses appropriate physical behaviour, it can satisfy any social behaviour based on that physical behaviour *if it chooses to do so*. So, an enterprise object can *assert* that it can satisfy a social behaviour; this implies it will undertake the obligation (a social constraint) to conform to that social behaviour.

Thus an enterprise object can take on an obligation to “see to it that” the necessary restrictions in behaviour are performed.

Obviously, an enterprise object can undertake many social constraints (obligations to conform to some social behaviour). Indeed, it might undertake too many and find itself with conflicting or inconsistent obligations. There is little point in constructing a policy framework that prevents conflicting/inconsistent obligations, since they occur constantly in the real world. Instead, this policy framework assumes that an enterprise object assigns priority to some obligations over others, and that these priorities may change over time (as discussed in Section 3.2).

Both the physical and social behaviour of an enterprise object are dynamic. An enterprise object might learn new skills (enlarging the alphabet of physical behaviour) or forget them (reducing the alphabet of physical behaviour). An enterprise object might join a community, leading to new obligations constraining its social behaviour, or it might leave a community, relaxing its social behaviour.

4.2. Enterprise Actions

The “doing of something” is an action. Concepts like activity and behaviour are related in the normal RM-ODP manner [7]. An enterprise action:

- can involve one or more players, each playing a labelled **participant-role** in the action template. Interactions have more than one participant-role, although all participant-roles could be carried out by the same enterprise object.
- happens if:
 - the action is *ready* (i.e. the precondition of the action is satisfied), and
 - all of the players of the action are:
 - *willing* to perform that participant-role based on their social behaviour (i.e. it is appropriate given their objectives and obligations)
 - *able* to perform their participant-role in terms of having the ability to play that participant-role, satisfying the predicate associated with the exercising of that ability, and satisfying any physical behavioural constraints

When considering the readiness of a multi-player action, the question arises about who is responsible for detecting the readiness and causing the action to begin. The answer depends on the level of abstraction. At a high level of abstraction, the precondition will be presented as a simple predicate without regard to how it would be evaluated. At a lower level of abstraction, this multi-party action would be refined into sub-actions, some of which would be the

detection of the readiness to start the high-level action and the consequent notification of other players to be involved in that high-level action.

Enterprise actions can be either physical actions or social actions (see Section 4.3) or both. Driving a car fast is, for the car, a physical action. Having an accident is a physical action. Choosing to drive fast, or (after the accident) choosing to drive slowly is a social action. The car still includes driving fast in its behaviour, but the driver applies a social constraint.

4.3. Social Actions

Physical behaviour of an enterprise object has associated physical state (e.g. which cars are in my driveway). Social behaviour has associated social state, which consists at least of:

- the objectives of the enterprise object
- the roles filled by the enterprise object
- the permissions, prohibitions, and obligations of the enterprise object
- the relative weightings of objectives and obligations

So, enterprise objects have social actions corresponding to:

- acquiring, relinquishing, and re-prioritising objectives
- joining/leaving a community (filling/unfilling a role) and the corresponding acquisition, relinquishing, and reprioritising of objectives

People regularly undertake a role with an obligation to do X, despite intending not to do X, when it conflicts with some other objective/obligation. That is, people do take on roles in the presence of conflicting obligations/objectives, but handle that conflict by assigning relative priorities to the conflicting obligations/objectives. They may make that assignment at the time of taking on the new obligations, or as late as when the conflict “occurs” (i.e. when there is no actual behaviour available that does not violate one or other of the obligations).

At a high level of abstraction, the social actions relating to objectives and participation in communities are sufficient. However, at more detailed level, these social actions involve a complex web of planning and decision-making, often based on imperfect knowledge about other enterprise objects. Planning and decision-making are social actions for devising future behaviours that lead to the achievement of obligations/objectives. Many enterprise objects (especially people) have significant social state and corresponding social actions relating to planning and decision-making.

As most enterprise objects engage in interactions with others, it becomes more difficult to make detailed long-term plans, due to the time required to analyse each of the combinatorial explosion of future scenarios, as illustrated in Section 3.2. (A social

activity need not have any physical embodiment, but it still can take time and “brain space”). Therefore, plans tend to be more detailed in the short term, while the long-term plans tend to be less detailed. By being more abstract, long-term plans are more stable in the presence of change, and can be refined later to determine the necessary details.

Since plans are significantly impacted by the behaviour of others, enterprise objects that plan tend to have social state relating to the behaviour of others, in terms of their objectives, roles, obligations, and weightings. This knowledge is used to remove unlikely scenarios from the planning activity, enabling time spent planning to focus on the analysis of likely scenarios in detail. Unfortunately, knowledge about other enterprise objects can be imperfect for many reasons:

- the information released about another enterprise object might be deliberately incomplete or inaccurate (this might be consistent with the objectives/obligations of that enterprise object)
- correct information may be released but it might not have been circulated sufficiently far or sufficiently fast or sufficiently reliably

So, information about other enterprise objects will tend to be associated with some “reliability estimate” reflecting the quality of the source of the information or the means by which the information was obtained.

Even in the presence of accurate knowledge of an enterprise object, we cannot rule out that the enterprise object will suddenly not conform to the behaviour expected by its objectives/obligations. So, this enterprise object will tend to keep “reputation estimates” of other enterprise objects, reflecting its expectation that their behaviour will be conformant. It is entirely possible for a well-behaved enterprise object X to have a bad reputation with another enterprise object Y, due to X’s incorrect knowledge of the conformant behaviour expected of X. These estimates of reliability of knowledge and reputation of other enterprise objects are all factors considered in deciding which future scenarios are sufficiently probable to merit detailed analysis.

All of this additional social state has associated social actions to create, update, and exchange social state. To manifest these social actions may require physical actions, e.g. communication between enterprise objects. However, such physical actions will usually occur at a lower level of abstraction and hence are not shown as distinct in the enterprise specification.

Research in economics and distributed artificial intelligence investigates the issues pertinent to social actions, and these will be used to model many aspects of social behaviour.

5. Enterprise Objects

Enterprise objects can either be “atomic” entities, or can be refined into a community of objects (see Section 6). Enterprise objects have abilities, objectives, and policies (permissions, prohibitions, obligations).

5.1. Abilities

An **ability** represents the action-playing that an enterprise object is capable of in terms of its physical behaviour.

An ability is described by:

- an action template/type
- a participant-role within that action
- an associated predicate, expressed in terms of any parameterisation of the behaviour and the players within the action

If an enterprise object has an ability, then that object is *physically* capable of playing that participant-role in that action, provided the predicate is satisfied. Having the ability does not imply that the enterprise object will play that participant-role; it might have undertaken obligations not to do so. The collection of abilities possessed by an enterprise object together with the physical (or programmed) constraints on those abilities determines an enterprise object’s physical behaviour.

5.2. Objectives

Enterprise objects have objectives; these objectives guide any internal choice in the enterprise object’s behaviour (see Section 3.2). For some enterprise objects (e.g. manufactured artifacts), the pursuit of the objective is pre-programmed. However, sentient enterprise objects (e.g. people) could be described as under an obligation to achieve their objective. Thus, for most modelling purposes, objectives can be regarded as pre-existing (background) long-term obligations.

To work toward an objective, enterprise objects may decide to join (or leave) a community.

5.3. Policies

5.3.1. Permission

RM-ODP defines permission as “*a prescription that a particular behaviour is allowed to occur. A permission is equivalent to there being no obligation for the behaviour not to occur.*”

A permission is defined by:

- an action
- a participant-role in that action
- a predicate on social behaviour
- a community-role
- an authority which grants the permission

If an enterprise object has this permission, then, when the predicate is true, the enterprise object can play the participant-role in the action when filling the community-role, by order of the authority.

There are two ways of looking at the concept of permission. The ODP text follows standard deontic logic in describing the way the world is, in terms of what actions may occur; this corresponds to the idea of “having permission”. However, there is another common usage, associated with “granting permission”, in which there is an implicit or explicit agency, and there are consequential obligations on the authority as a result of granting a permission. The authority should not normally simultaneously grant a permission and prevent the permitted action from taking place. Thus, the second statement in the ODP definition is weaker than the first, because it fails to recognise that permission can be “empowering”. If X is permitted to do Y, then X has the luxury of *choosing* whether or not to do Y, but the authority has no such choice; it is obligated to allow X to do Y.

A real-world analogy is “I permit you to use my car today”. You might or might not choose to use the car, but I have obligated myself to not refuse you the keys. However, permission does not require the authority to take all possible steps to facilitate X doing Y. For example, giving permission to use my car does not obligate me to fill the car with petrol, arrange for you to get a drivers licence, etc.

However, “I permit you to use Bill Clinton’s car” is not a valid permission, as I cannot undertake the obligation to make Bill Clinton’s car available. Of course, I can undertake the obligation, but I cannot fulfill it, unless Bill Clinton’s car (or the enterprise objects that control access to it) recognises my authority over it. This is discussed further in Section 6.3.

However, analysis of obligations implied by the granting of permissions is clearly an area where the branching trees of consequences from our various actions rapidly become intertwined, and some level of conflict or inconsistency in the obligations that result is inevitable.

5.3.2. Prohibition

RM-ODP [7] defines prohibition as “*a prescription that particular behaviour must not occur. A prohibition is equivalent to there being an obligation for the behaviour not to occur.*”

Like a permission, a prohibition is defined by:

- an action
- a participant-role in that action
- a predicate on social behaviour
- a community-role
- an authority which imposes the prohibition

If an enterprise object has this prohibition, then, when the predicate is true, the enterprise object cannot play the participant-role in the action when filling the community-role, by order of the authority.

5.3.3. Obligation

RM-ODP [7] defines obligation as “*a prescription that particular behaviour is required. An obligation is fulfilled by the occurrence of the prescribed behaviour*”.

Obligations are different from permissions and prohibitions. They are generally more primitive, because the obligation to obey the policy-making of the community is the basis on which communities grant permissions and impose prohibitions. Thus, a community may create a context in which the acceptance of permissions, prohibitions or further obligations is itself obligatory, based on prior agreement by the members of the community.

Obligations are not granted or imposed, but rather are agreed as part of a joining a community (entering into a contract).

An obligation influences the choices made by the enterprise object in order to pursue a favourable “future world” (i.e. a path through the tree of possible future behaviours) consistent with the obligation. Since an enterprise object will usually have many obligations, there must be some relative weights associated with these goals to enable the optimal strategy to be determined, as was discussed in 3.2. Note that it is not generally possible to determine whether a path will lead to the fulfillment of an obligation, and so choices are often made using probabilistic analysis based on possibly incomplete or incorrect information. That is, the enterprise object makes what it *believes* to be the best choice.

An enterprise object will typically have objectives at instantiation. It is these initial objectives which may cause the enterprise object to undertake roles in communities in the belief that the activities of the community will contribute to its objectives. In doing so, the enterprise object acquires objectives and obligations related to the community and its role within it.

The weighting of different objectives and obligations is not fixed, but may vary throughout the lifetime of the enterprise object. Some actions (whether performed by this enterprise object or others) may have the effect of altering the weightings of these goals.

Obligations can be expressed as:

- enabling (trigger) conditions, which makes the obligation “active”. This may be expressed either as:
 - a predicate that holds while the obligation is “active”, e.g. “when it is dark, you will watch my house”
 - a pair of activating and deactivating conditions which toggle the obligations into active and inactive modes, e.g. “when the sun sets, you must start watching my house and continue to do so until the sun rises again”.
- satisfaction condition, which signifies the obligation has been satisfied, e.g. “you must pay me \$10”
- violation condition, which signifies the obligation is unachievable, e.g. a deadline has passed

All of the above might be expressed in terms of predicates on states, or the occurrence of some behaviour.

Standing obligations can never be satisfied, so these must be defined by a violation condition.

To achieve an objective, an enterprise object may decide to join (or leave) a community.

6. Community

A community (as defined in Part 3) is described by (at least):

- the behaviour of the community (can include both physical and social behaviour)
- the roles of the community, which describe the subset of the community behaviour performed by the enterprise object that fills each role. This can include both:
 - a subset of the community’s physical behaviour and its corresponding social behaviour
 - preconditions on the enterprise object that fills the role, which can be both physical (possessing the abilities needed to perform the role behaviour) and social (e.g. possessing necessary permissions to perform the role behaviour)
 - the policies that are applicable to the role, which the enterprise object filling the role must agree to accept
- the combination of participant-roles and community-roles over which this community claims authority to grant/impose policy. The community may claim the authority in its own right, or it may claim the authority by delegation from a superior authority (e.g. an outer community).

In filling a role in a community, the enterprise object is obligated to accept the (direct or delegated) authority of the community over the corresponding participant-roles, and constrain their physical behaviour accordingly. This is the fundamental community contract, from which all specific community contracts are derived.

Communities can be nested and communities can overlap (have one or more participants in common). For nested communities, the inner community is bound by the policies of the outer community. For overlapping communities, the enterprise objects in the intersection of the two communities are bound by the policies of both communities.

Every community has a contract, and every contract defines a community. In the real world, some contracts are not usually regarded as forming a community due to their ephemeral nature, but nonetheless there *is* a community, albeit short-lived, e.g. buying at a garage sale. Indeed, communities can be formed to carry out a single action. The definition of enterprise action in Section 4.2 hints at the possibility that an action with its participant-roles could be refined into a community with corresponding roles.

6.1. Policy Making in Communities

In general, there are no default rules for policy-making in communities. That is, in the absence of either permission or prohibition, it is not defined whether or not an enterprise object is *willing* (as defined in Section 4.2) to play a participant-role. The default rules must be specified by the community.

An enterprise object (especially one filling roles in multiple communities) can acquire a collection of permissions and prohibitions with overlapping scope (e.g. pertaining to the same participant-role). A superior authority must provide a resolution mechanism to determine whether a given collection of permissions and prohibitions does or doesn't make the enterprise object willing to perform the participant-role. A resolution mechanism of a delegated authority may be constrained by the nature of its delegation from the superior authority.

An authority is free to impose any system of resolution it wishes (unless it is constrained by obligations, e.g. to an outer community). However, the following principles are commonly applied and underpin the legal frameworks in many countries [5]:

- *lex specialis legi generali derogat* - the specific overrides the general
- *lex superior legi inferiori derogat* - higher authorities overrule lower authorities
- *lex posterior legi anteriori derogat* - new law overrides old law.

6.2. Why do Communities create Policy Frameworks?

In what circumstances would a community create a policy framework for some X? There are two common reasons:

- The community believes that controlling X would contribute to its objectives.
- The community is obligated (by some outer community) to control X.

In the absence of a policy framework of an outer community, the existence of a policy framework for X in this community implies that X is prohibited unless it conforms to this community's policies. Note that this default prohibition might not be respected by non-members of the community, and other mechanisms might be needed to impose the compliance of non-members (if so desired). For example, an environmentally-conscious group might decide in which circumstances members are permitted to drive their cars. While group members may be bound by these policies as a condition of membership of the group, this community is unable to constrain the (ab)use of cars by non-members through policy alone.

In the presence of a policy framework of an outer community, the policy framework of the inner community is bounded by the framework of the outer community. The inner community cannot permit what is prohibited by the outer community, unless it has a delegated authority to do so. The inner community may be able to prohibit or obligate its members, provided the outer community does not prohibit the inner community from imposing such policies. For example, the Driving Licence Commission (DLC) might be permitted to decide the rules for the issuing of drivers licences (e.g. requiring some test of driving competency), but the government might obligate the DLC to issue drivers licences only to those applicants over 18 years old. That is, the DLC cannot give a drivers licence to a child, even if the child can pass the DLC's competency test.

The way that non-members of the community are constrained will, in general, be derived from some outer level community which does involve all the parties concerned. Therefore, policies made by the inner community are respected in the outer community, only when the authority of the inner community is obtained by delegation from the outer community.

6.3. What can be the Subject of Policy?

When a community establishes a policy framework over X, there is usually the expectation that Xs recognise the community as appropriate authority. Therefore, the scope for establishing an effective policy framework is limited to the enterprise objects choosing to fill roles in the community and anything

over which they have effective control (this control might arise through participation in other communities, or through ownership).

Policy *can* reference enterprise objects that don't meet the criteria above, but it must be made clear which enterprise objects are actually obligated. There is a significant difference between "The members of this bushwalking club shall not cut down the trees in the forest" and "Nobody shall cut down the trees in the forest", where the trees in the forest are not under the control of the bushwalking club. The first one obligates the members of the community (this is valid); the second one obligates people outside the community (and is not valid). A more appropriate/realistic expression of the latter obligation would be "Members of this community will try to prevent the cutting down of the trees in the forest"; this puts the obligation back onto the enterprise objects who can be constrained by the policies of this community.

Of course, if an outer community gave the bushwalking club the right to make policy about the trees in the forest, then members of the outer community would be constrained by the policies of the bushwalking club.

In general, this concept of an outer community needs to be brought into play whenever the problem of conflict between the policies of autonomous communities arises. In this way, policies can be defined with regard to some form of community, albeit sometimes a rather loose one. Where there is no acknowledged outer community, it may not be possible to resolve conflicts in a civilized way, and disputes may be ongoing, or resolved by possession or by force.

6.4. Enforcing Policy

A community can choose to enforce its policies by optimistic or pessimistic means.

Pessimistic enforcement is essentially preventative and involves on-going checking. Mechanisms are devised to ensure that the right things are done and the wrong things are not. Real world examples include locking a car to prevent access except by those with keys, checking of a security pass on entry to a building, etc. Passwords and access control lists are used in computer systems. Note that all of these examples are primarily concerned with preventing the prohibited actions. It is more difficult to devise mechanisms to force required things to happen. However, there are some examples, e.g. an alarm clock can be set to ensure that the person wakes up on time. Generally pessimistic enforcement of obligations tends to take the form of nagging, "You still haven't done X", i.e. constant reminders of the obligation.

Pessimistic enforcement tends to be used:

- when trust is low, i.e. when the community has the belief (rightly or wrongly) that non-compliance is rife
- when the damage potentially caused by non-compliance is high
- when viable preventative mechanisms can be created
- when some effective sanction can be applied post-hoc to those who do not comply

An optimistic enforcement does not involve preventative measures, but relies on detecting non-compliance and reporting/correcting them. This is widely used in real life. It tends to be used when:

- when trust is high
- when the potential damage due to non-compliance is low
- when viable preventative mechanisms do not exist

The availability of viable preventative mechanisms has to be assessed against the objectives of the community. In real life, cheap convenient preventative mechanisms often exist but their use is prohibited by concerns about civil liberties. Or, to put it more simply, a community must weigh up the relative risk of non-compliance against the costs of enforcing compliance and the risks inherent in the compliance mechanism.

One of the difficulties with pessimistic enforcement is that the mechanisms might enforce policy on a wider range of enterprise objects than the community actually has the authority to do. For example, the bushwalking club might build a high fence around the forest to prevent non-members from chopping down the trees. In the absence of any prohibition regarding denying access to the forest, the bushwalking club would be contributing positively towards their obligations by building the fence.

7. Conclusions

The Enterprise Viewpoint of RM-ODP has been less developed than most of the other ODP viewpoints (e.g. computational viewpoint). This paper identifies a number of substantial bodies of work in traditionally independent research areas, such as deontic logic, agency theory, and speech act theory, which might be usefully combined as a basis for a normative ODP Enterprise Language. However, much of this research is still in progress, and the future work on the Enterprise Viewpoint must incorporate new advances in those disciplines.

This paper primarily addresses the relationship between policies and communities. In particular, it has been necessary to refine how policies can be imposed by a community and how policies in different communities interact. To do this, it has been necessary to refine enterprise behaviour. The refinement of enterprise behaviour reveals that traditional

computational modelling techniques can be applied to the physical behaviour of enterprise objects, but that enterprise behaviour has an additional dimension of social behaviour to express how enterprise objects can strive towards objectives, choose to undertake obligations, and make plans correspondingly. Having refined enterprise behaviour, policy statements (permissions, prohibitions, and obligations) can be associated with the roles, actions, and authorities of communities.

The next step in this research programme is to perform mappings from the enterprise concepts described in this paper to specific information and computational languages, e.g. UML [15] and CORBA [16].

Acknowledgements

The work reported here was undertaken during a sabbatical visit by Peter Linington to the Cooperative Research Centre for Distributed Systems Technology. The work reported in this paper has been funded in part by the Co-operative Research Centre Program through the Department of Industry, Science & Tourism of the Commonwealth Government of Australia.

References

- [1] Austin, J.L. 1962, *How to do things with words*, Cambridge, Harvard University Press
- [2] N. Belnap and M. Perloff, *Seeing to it that: a canonical form for agentives*. *Theoria*, 54: 175-199, 1988.
- [3] M. Brown, *Doing As We Ought: Towards a Logic of Simply Dischargeable Obligations*, in *Deontic Logic, Agency and Normative Systems*, eds. M. Brown and J. Carmo, Proceedings of DEON'96, 3rd Int. Workshop on Deontic Logic in Computer Science, January 1996.
- [4] M. Brown, *Agents with Changing and Conflicting Commitments: A Preliminary Study*, In Proc. 4th Int. Workshop on Deontic Logic in Computer Science (DEON'98), January 1998.
- [5] N. den Haan, "Investigations into the Application of Deontic Logic" in *Executable Modal and Temporal Logics*, Proc. of the IJCAI-93 Workshop, M. Fisher and R. Owens eds, Springer, pp 157-178.
- [6] J. Horty, *Combining Agency with Obligation (Preliminary Version)*, in *Deontic Logic, Agency and Normative Systems*, eds. M. Brown and J. Carmo, Proceedings of DEON'98, 3rd Int. Workshop on Deontic Logic in Computer Science, January 1996.
- [7] ISO/IEC IS 10746-2. *International Standard 10746-2, ITU-T Recommendation X.902: Open Distributed Processing - Reference Model - Part 2: Foundations*, January 1995.
- [8] ISO/IEC IS 10746-3. *International Standard 10746-3, ITU-T Recommendation X.903: Open Distributed Processing - Reference Model - Part 3: Architecture*, January 1995.
- [9] ISO/IEC WD 15414. *Open Distributed Processing - Reference Model - Enterprise Viewpoint*, January 1998.
- [10] E. Lupu, M. Sloman, *A Policy Based Role Object Model*, in Proc. 1st International Workshop on Enterprise Distributed Object Computing (EDOC'97), pp 36-47, Gold Coast, Australia, October 1997.
- [11] J-J Ch Meyer, R.J. Wieringa and F.P.M Dignum, *The role of Deontic Logic in the Specification of Information Systems*, chapter 2 in "Logics for Databases and Information Systems", Eds J Chomicki and G Saake, Kluwer, 1998.
- [12] Z. Milosevic, A. Berry, A. Bond, K. Raymond. *Supporting Business Contracts in Open Distributed Systems*, In Proceedings of the 2nd International Workshop on Services in Distributed and Networked Environments (SDNE'95), pages 60-67, Whistler, Canada, IEEE Computer Society Press, June 1995.
- [13] Z. Milosevic, *Enterprise aspects of open distributed systems*, Ph.D. Thesis, The University of Queensland, 1995.
- [14] J.von Neumann, O. Morgenstern, *Theory of Games and Economic Behavior*, Princeton: Princeton University Press, 1944.
- [15] Object Management Group, *Unified Modelling Language*, OMG ad/97-08-{02-09}, August 1997.
- [16] Object Management Group, *CORBA/IIOP 2.2 Specification*, OMG formal/98-02-01
- [17] Henry Sidgwick, *The Methods of Ethics*, 7th Ed. London, 1907.
- [18] S. Tyndale-Biscoe, B. Wood "Machine Responsibility - How to deal with it" in Proc. 1st International Workshop on Enterprise Distributed Object Computing (EDOC'97), pp 36-47, Gold Coast, Australia, October 1997.