Goal Directed Conflict Resolution and Policy Refinement

Mukta S. Aphale *, Timothy J. Norman, and Murat Şensoy

Dept. of Computing Science,
University of Aberdeen,
Aberdeen, UK.
{m.aphale,t.j.norman,m.sensoy}@abdn.ac.uk

Abstract. A policy (or norm) is a guideline stating what is allowed, what is forbidden and what is obligated for an entity, in a certain situation, so that an acceptable outcome is achieved. Policies occur in many types of scenarios, whether they are loose social networks of individuals or highly structured institutions. It is important for policies to be consistent and to support the goals of an organisation. This requires a thorough understanding of the implications of introducing specific policies and how they interact. It is difficult, even for experts, to write consistent, unambiguous and accurate policies, and conflicts are practically unavoidable. In this paper we address the challenge of providing automated support for identifying and resolving logical and functional conflicts. We present a model of conflict identification and resolution that focuses attention on conflicts that are most critical to the goals of the organisation.

Keywords: Policies, Norms, Policy Authoring, Conflict Resolution, Intelligent Agents

1 Introduction

Policies guide and regulate behaviour of various entities in a system. They are system-level constraints that are independent from the implementation of specific agents and represent the ideals of behaviour of these agents. The benefits of a policy-based approach include reusability, extensibility, context-sensitivity, verifiability, information security, support for simple and sophisticated components and reasoning about component behavior [12].

Policies can be classified into two categories — collective and individual. Collective policies are sets of rules applicable to various entities in a particular group. They represent an agreement among agents who are responsible for defining the rules and ensuring that common goals of the group are achieved successfully. For

* The research described here is supported by the award made by the RCUK Digital Economy programme to the dot.rural Digital Economy Hub; award reference: EP/G066051/1

example, NHS Care Record Guarantee for England is an agreement between NHS (National Health Service) in England and patients, brokered by patient organisations and senior healthcare experts. Personal preferences of entities are also often expressed by individual policies. For example, rules for sharing personal information in a social network. Policies operate in conjunction with individual and organisational goals, such as the provision of effective patient care. The key challenge here is: how to specify policies/norms that protect important information (secrets) and promote ideal action (normal operating procedures), while ensuring the achievement of individual/organisational goals?

It is difficult, even for experts to write consistent, unambiguous and accurate policies. Policies may conflict with each other and with organisational goals in many situations; i.e., logical and functional conflicts, respectively [4]. Identifying and resolving such conflicts, and functional conflicts in particular, is an important challenge. This question has been explored in the context of practical reasoning by Kollingbaum [8], where an agent architecture (NoA) is proposed that detects direct and indirect conflicts between norms and action choices. Conflicts are then either resolved through heuristic strategies or labelled for further deliberation. Conflicts between beliefs, obligations, intentions and desires are also explored in the BOID architecture [3], where the aim to to identify maximal subsets of consistent norms and intentions. Reasoning about plans and norms has been addressed by Meneguzzi [9], where a BDI agent programming language was extended to include normative constraint checking. In this way, agents can modify their behaviour in response to newly accepted norms, by creating new plans to comply with obligations and suppressing the execution of existing plans that violate prohibitions.

In other research it has been demonstrated how intelligent agents can assist humans in complex, norm-governed decision making [11] and prognostically reason about possible normative violations and replan to avoid these violations [10]. In addition to supporting human decision-making constrained by norms or policies, existing research has addressed the problem of supporting humans in authoring policies. In this research, however, there is very limited automated conflict detection and resolution for policy authoring. The authoring process is guided through the use of templates in the work of Johnson et al. [7]. In Uszok et al.[13] some reasoning support for conflict detection has been explored, but this in confined to the detection and resolution of logical conflicts.

Hence, conflict detection and resolution with specific focus on functional conflict detection is emphasised in this paper. Conflicts occurring in a system are of varying significance. More critical are the ones which have higher chances of occurring and the ones which can impair achievement of goals. Hence, it is crucial to identify the conflicts, resolution of which, will lead to maximum benefits and goal achievement. The key questions addressed are: how to identify conflicts that are most relevant to the domain and goals of organisation, and how to resolve

conflicts that are most important to the goals of organisation. Since focus of our research is authoring consistent and functional policies, refining policies and resolving relevant conflicts to maximise goal-achievement is investigated in this paper.

This paper is organised as follows: introduction to Polar Agent application that is developed for policy authoring and planning is given in Section 2. Our notions of policies and conflicts are illustrated in Section 3. Activity prioritisation model for identifying conflicts that are relevant to the domain and goals of organisation is described in Section 4. Conflict detection and resolution methods implemented by the Polar Agent are described in Section 5 and Section 6 respectively; specific focus remains on detecting functional conflicts (i.e., conflicts between policies and organisational goals). Finally, conclusion and scope for future work are presented in Section 7.

2 Polar Agent

Polar Agent is developed for authoring OWL-POLAR (OWL-based POlicy Language for Agent Reasoning) [5] policies and planning using OWL-POLAR policies (See Section 3). It consists of two core modules — Policy Authoring and Planning. Automated reasoning, conflict detection and resolution mechanisms are implemented by the authoring module. All possible plans for a specified goal can be generated using the planning module. The active policies are taken into consideration and normative status (i.e., active permissions, prohibitions and obligations) of each plan, before and after the execution, can be displayed. Intelligent agent assists in understanding the implications of policies and conflicts and correctly defining precedence between conflicting policies. The architecture of the Polar Agent is explained in Fig. 1.

1. ***Policy Authoring Module*** consists of an interface for specifying policies using OWL-POLAR and Conflict Detection and Conflict Resolution modules.
   - ***Conflict detection*** module detects logical and functional conflicts. Prioritised activities are provided to this module to aid the goal-directed detection and resolution of conflicts.
   - ***Conflict resolution*** module consists of three core processes: Suggesting Possible Refinements, Explanation and Refinement: Various solutions are determined for the conflicts detected. Implications of a particular conflict, characteristics of conflicting policies, priority for resolving the conflict and implications of each solution are explained. The best possible solution is recommended to the author.
2. ***Planning Module*** contains an interface for specifying goals and an Activity Prioritisation Model. The Activity Prioritisation Model consists of a temporary database to store most probable plans and three core processes: Plan Generation, Plan Annotation and Activity Prioritisation.

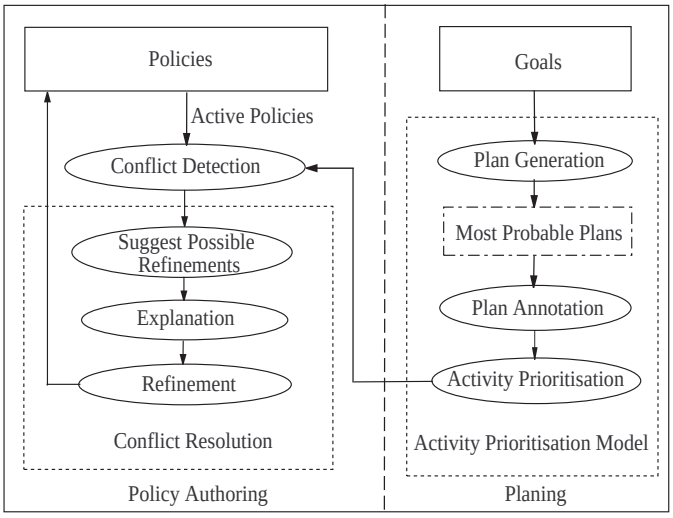


Fig. 1. Polar Agent Architecture

- ***Plan Generation*** module generates all possible plans for a particular goal. The most probable plans are stored in a temporary plan database. Most probable plans are determined based on various factors such as cost, previous experience and certainty of goal achievement etc. Depending on significance of goal, availability of resources and computational capacity; it is also possible to consider all plans that are generated.
- ***Plan Annotation*** is performed on the most probable plans. The annotated plans are used for prioritising activities.
- ***Activity prioritisation*** module prioritises activities. High priority activities are used by conflict detection module to detect and resolve logical as well as functional conflicts.

Our notion of policies and conflicts and various functions of the Polar Agent will be explained in the subsequent sections with specific emphasis on activity prioritisation and functional conflict detection (i.e., conflicts between policies and probable plans).

3 Policies and Conflicts

OWL-POLAR [5] is used as a language for expressing policies in this work. OWL-POLAR is based on OWL-DL and is a powerful OWL 2.0 knowledge representation and reasoning mechanism for policies. The expressiveness of OWL-POLAR is not restricted to DL. Hence, OWL-POLAR is sufficiently expressive to be used for specifying policies in real-life applications. Since policy-governed decision making and policy analysis is enabled within bounds of decidability,

```
<policy>
<var> ?x </var>
<var> ?a </var>
<var> ?b </var>
<addressee> ?x </addressee>
<role> Doctor(?x) </role>
<modality> P </modality>
<action var = "?a"> Access(?a), PatientRecord(?b),
  about(?a, ?b), hasActor(?a, ?x) </action>
</policy>
```

Fig. 2. RDF Syntax of an OWL-POLAR Policy

reasoning about policies and detection of policy conflicts is efficient using OWL-POLAR. An example of an RDF representation of an OWL-POLAR policy ***'Doctors are permitted to access patient's record.'*** is given in Fig. 2.

Since OWL-POLAR is based on OWL-DL, it is formally specified within OWL-DL. An OWL-DL ontology $o = (TBox_o, ABox_o)$ consists of a set of axioms defining the classes and relations ($TBox_o$) as well as a set of assertional axioms about the individuals in the domain ($ABox_o$). Concept axioms are of the form $C \sqsubseteq D$, where C and D are concept descriptions. Relation axioms are expressions of the form $R \sqsubseteq S$, where R and S are relation descriptions. The ABox contains concept assertions of the form $C(a)$, where C is a concept and a is an individual name; and relation assertions of the form $R(a, b)$, where R is a relation and a and b are individual names.

Conjunctive semantic formulae are used to express policies. A conjunctive semantic formula $F^o_{\boldsymbol{v}} = \bigwedge_{i=0}^{n} \phi_i$ over an ontology o is a conjunction of atomic assertions ϕ_i, where a vector of variables used in these assertions is represented by $\boldsymbol{v} = \langle ?x_0, \ldots, ?x_n \rangle$. For the sake of convenience $\bigwedge_{i=0}^{n} \phi_i \equiv \{\phi_1, \ldots \phi_n\}$ is assumed in order to consider a conjunctive formula . Based on this, $F^o_{\boldsymbol{v}}$ can be considered as $T^o_{\boldsymbol{v}} \cup R^o_{\boldsymbol{v}} \cup C^o_{\boldsymbol{v}}$, where $T^o_{\boldsymbol{v}}$ is a set of type assertions using the concepts from o, e.g. $\{student(?x_i), nurse(?x_j)\}$; $R^o_{\boldsymbol{v}}$ is set of of relation assertions using the relations from o, e.g. $\{marriedTo(?x_i, ?x_j)\}$ and $C^o_{\boldsymbol{v}}$ is a set of constraint assertions on variables. Each constraint assertion is of the form $?x_i \triangleleft \beta$, where β is a constant and $\triangleleft$ is one of the symbols $\{>, <, =, \neq, \geq, \leq\}$. A constant is either a data literal (e.g. a numerical value) or an individual defined in o.

Variables are divided into two categories: data-type and object variables. A data-type variable refers to data values (e.g. integers) and can be used only once in $R^o_{\boldsymbol{v}}$. On the other hand, an object variable refers to individuals (e.g. University_of_Aberdeen) and can be used any number of times in $R^o_{\boldsymbol{v}}$. Equivalence and distinction between the values of object variables can be defined using OWL properties *sameAs* and *differentFrom* respectively, e.g. *owl:sameAs(?x,?y)*. In the rest of the paper, the symbols α, ρ, φ, and e are used as a short hand for semantic formulae.

Definition 1 Given an ontology o, a conditional policy π is defined as $\alpha \longrightarrow N_{\chi:\rho}\left(\lambda : \varphi\right)/e$

- α, a conjunctive semantic formula, is the activation condition of the policy.
- $N \in \{O, P, F\}$ indicates the modality of the policy (i.e., if the policy is an obligation, permission or prohibition).
- χ is the policy addressee and is described by ρ using only the *role* concepts from the ontology (e.g. $?x : student(?x) \wedge female(?x)$, where *student* and *female* are defined as sub-concepts of the concept *role* in the ontology). That is, ρ is of the form $\bigwedge_{i=0}^{n} r_i(\chi)$, where $r_i \sqsubseteq role$. Note that χ may directly refer to a specific individual (e.g. *John*) in the ontology or a variable.
- $\lambda : \varphi$ is the regulated action or state. λ is a variable referring to an action or a state that is regulated by the policy; where λ, as an action instance or a state, is described by φ using the concepts and properties from the ontology (e.g. $?a : SendFileAction(?a) \wedge hasReceiver(?a, John) \wedge hasFile(?a, TechReport218$ *.pdf*), where *SendFileAction* is an *action* concept). Each action concept has only a number of functional relations (aka. functional properties) [1] and these relations are used while describing an instance of that action.
- e is the expiration condition of the policy.

∎

As stated in Section 1 conflicts are practically unavoidable. Efficient conflict detection and resolution mechanisms are required for authoring consistent, unambiguous and functional policies. Here, following Castelfranchi [4], we distinguish between logical and functional conflicts.

Logical Conflicts may arise between policies. If the same action is both prohibited and permitted or both prohibited and obligated at the same time, the entity adopting these policies will not be able to decide which policy should be respected. To avoid such conflicts, a thorough understanding of the exact meaning and the implications of a policy, individually and as a part of a set of policies, is required.

Definition 2 Given logically conflicting policies $\pi_i = \alpha^i \longrightarrow A_{\chi^i:\rho^i}\left(\lambda^i : \varphi^i\right)/e^i$ and $\pi_j = \alpha^j \longrightarrow B_{\chi^j:\rho^j}\left(\lambda^j : \varphi^j\right)/e^j$, The logical conflict L is defined as : $L \longmapsto \pi_i \times \pi_j$. ∎

For example, consider two policies defined in healthcare domain.

- $\pi 1$ - Doctors are prohibited from modifying the chemotherapy regime of a patient.
- $\pi 2$ - Oncology specialists are permitted to modify the chemotherapy regime of a patient.

Since an Oncology Specialist is a subclass of Doctor, both the policies $\pi 1$ and $\pi 2$ are applicable. Hence, a logical conflict exists between $\pi 1$ and $\pi 2$. This conflict arises due to $\pi 1$ being ill-formed. It should refer to General Practitioners rather than Doctors in general, since GP is a subclass of Doctor and GP and Oncology Specialists are distinct.

Functional Conflicts arise between policies and underlaying goals of an organisation [4] or when a policy becomes non-functional as a consequence of existence of some other policy. Identifying and resolving functional conflicts is a crucial challenge and little work has been done to date [8].

Definition 3 Given functionally conflicting policies $\pi_i = \alpha^i \longrightarrow A_{\chi^i:\rho^i}\left(\lambda^i : \varphi^i\right) / e^i$ and $\pi_j = \alpha^j \longrightarrow B_{\chi^j:\rho^j}\left(\lambda^j : \varphi^j\right) / e^j$, The functional conflict F is defined as : $F \longmapsto (\pi_i \times \pi_j)_{type}$, where *type* denotes type of the conflict, for example forbidden side-effects, forbidden pre-condition, inaccessible input/output, etc. ▌

Consider, for example, two policies defined in healthcare domain.

- $\pi 1$ - Nurses are permitted to perform various tests on patients
- $\pi 2$ - Nurses are prohibited from updating health-details of patients.

As a side-effect of performing various tests, nurses update health-records of patients. Hence, a functional conflict (due to side effect) exists between $\pi 1$ and $\pi 2$.

Let us consider another example.

- $\pi 1$ - Doctors are obliged to study cases of patients.
- $\pi 2$ - Doctors are prohibited from accessing complete medical history of patients.

Access to complete medical history of a patient is required as a pre-condition for a thorough study of a case. Hence, a functional conflict (due to precondition) exists between $\pi 1$ and $\pi 2$.

Conflict detection and resolution are computationally expensive. Hence, the reasoning mechanism must focus on the most relevant conflicts, given the goals of the organisation/agent. Conflicts occurring in a system are of varying significance. Some conflicts have higher chances of occurring and they need to be resolved statically, while others have very rare chances of occurring and they can be resolved at runtime [6]. We argue that, more critical are the conflicts that can impair achievement of goals. Hence, it is crucial to identify the conflicts, resolution of which, will lead to maximum benefits and goal achievement.

4 Activity Prioritisation Model

An activity prioritisation model that focuses on detecting and resolving an optimum set of conflicts is implemented. The aim of the activity prioritisation model is to identify the activities that are most important given the domain and goals of organisation. This model is based on the Page Rank Algorithm [2] – an algorithm to measure the relative importance of each element within a set. In the activity prioritisation model relative importance of each activity is determined based on the weight of the activity. The weight of each activity is computed according to probability of execution of the activity while achieving a goal (i.e., frequency of occurrence of the activity in all the plans), average cost of achieving the goal from the activity and probability of successful execution of the activity.

First, all possible plans for achieving a goal G from an initial state T are generated. Let us assume that A_c is a set of all atomic actions defined in the domain and $Plans$ is a set of all plans generated. The definition of a plan is given below (Definition 4).

Definition 4 A plan $p \in Plans$ is a sequence of atomic actions $(\alpha_1, \alpha_2, ..., \alpha_n)$ such that $\alpha_i \in A_c$. ▮

Once possible plans are identified, each activity is annotated as specified in Definition 5. Fig. 3 shows an example of graphical representation of plans and plan annotation.

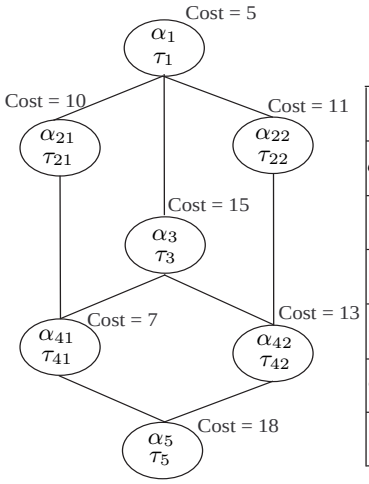
Definition 5 An annotation τ_i for action α_i is a tuple $< A_i^{precede}, A_i^{follow}, \Upsilon_{i\to g}, weight_i^o, weight_i^u >$, where $A_i^{precede}$ is the set of all actions preceding α_i in all the plans containing α_i; A_i^{follow} is the set of all actions that follow α_i in all the plans containing α_i and $\Upsilon_{i\to g}$ is a set of costs of every path from α_i to α_g, where α_g is the last activity of the plan. Original weight and ultimate weight of α_i are represented by $weight_i^o$ and $weight_i^u$ respectively. ▮

The weights of the activities are then computed. Formulae for calculating the original weight and ultimate weight of an activity α_i are given below:

$$weight^o_{\alpha_i} = Pr(\hat{\alpha_i}) \times \left(\frac{\Upsilon^{avg}}{\Upsilon^{avg}_{i\to g}} \right) \tag{1}$$

$Pr(\hat{\alpha_i})$ represents the probability of successful execution of α_i (typically based on past experience). $\Upsilon^{avg}_{i\to g}$ is the average cost of achieving goal for α_i i.e., average of the values stored in $\Upsilon_{i\to g}$ (Equation 2). Υ^{avg} is the sum of average costs of achieving the goal for all the activities being considered (Equation 3).

$$\Upsilon^{avg}_{i\to g} = \frac{\sum \Upsilon_{i\to g}}{|\Upsilon_{i\to g}|} \tag{2}$$



$Assumption : Pr(\hat{\alpha_i}) = 1 \ \forall \ \alpha_i$

	$\Upsilon_{i\to g}$	$\Upsilon^{avg}_{i\to g}$	ϖ^o_i	ϖ^u_i	Priority
α_1	40,37, 51,47	43.75	5.43428	5.43428	2
α_{21}	35	35	6.79286	2.05277	5
α_{22}	42	42	5.66071	1.71064	7
α_3	40,46	43	5.52907	1.67086	6
α_{41}	25	25	9.51	2.97771	3
α_{42}	31	31	7.66935	2.45656	4
α_5	18	18	13.20833	5.43427	1

$\sum \Upsilon^{avg}_{i\to g} = 237.75$

Fig. 3. Graphical Representation of Plans to Achieve Goal G, Plan Annotation and Activity Prioritisation

$$\Upsilon^{avg} = \sum_{\alpha_i \in \bigcup\limits_{p \in Plans} p} \Upsilon^{avg}_{i\to g} \tag{3}$$

Hence, $weight^o_i$ is inversely proportional to the average cost of achieving the goal, i.e., lower the average cost of achieving the goal more is the weight of the activity. $weight^o_i$ is directly proportional to the probability of successful execution of the activity, i.e., higher the probability more is the weight of the activity. The assumption here is that lower cost plans have higher precedence over higher cost plans.

After determining original weights of all the activities, ultimate weight of each activity is computed by taking into account the weight induced by its preceding activities.

$$\begin{aligned} weight^u_\alpha &= weight^o_\alpha && if \quad \left|A^{precede}_\alpha\right| = 0 \\ &= \sum_{\beta \in A^{precede}_\alpha} \left(weight^u_\beta \times \frac{weight^o_\alpha}{\sum\limits_{\epsilon \in A^{follow}_\beta} weight^o_\epsilon} \right) && otherwise \end{aligned} \tag{4}$$

The activities preceding α are represented by β. The activities that follow β (including α) are represented by ϵ. If α is the first activity in a plan, then ultimate weight remains the same as the original weight of α. If an activity is not

the first activity in the plan, ultimate weight of activity α is the sum of weights induced by all its preceding activities β. A portion of the ultimate weight of β is induced on each activity that follows β in porportion to the original weights of the following activities. If β has only one following activity the ultimate weight of β is completly induced on that activity. If β has more than one following activities, then its weight is divided among all of them in proportion to their original weights. Activities are prioritised according to their ultimate weights. Higher the ultimate weight, higher is the priority. Original weight is considered if two activities have the same ultimate weights.

Fig 3 shows an example of activity prioritisation. α_1 is the first activity in the plan. Hence, its ultimate weight remains the same as its original weight. The ultimate weight of α_1 is divided among its preceding activities α_{21}, α_{22} and α_3 in proportion to their original weights. Since the activities α_{21}, α_{22}, α_{41} and α_{42} have only one preceding activity each, their weights are not divided. The ultimate weight of α_{41} is the sum of its share from the ultimate weights of α_3 and α_{21}. Similarly, the ultimate weight of α_{42} is computed. Finally, the ultimate weight of α_5 is the sum of the ultimate weights of α_{41} and α_{42}. The ultimate weights of α_1 and α_5 are approximately the same. Hence, their priorities are decided according to their original weights.

5 Conflict Detection

The most significant (i.e., high priority) activities in the plans generated are identified using the activity prioritisation model. Only the active policies that regulate the high priority activities are reasoned about to detect and resolve the most significant logical and functional conflicts. If no policy exists that regulates a high priority activity, a temporary permission is generated and used by the reasoning mechanism.

A significant amount of prior research has focused on the detection and resolution of conflicts between policies i.e., logical conflicts [5][8][14]. Little work to date has been done in order to detect and resolve functional conflicts. The question has been explored in the context of practical reasoning, where an agent architecture (NoA) is proposed that detects direct and indirect conflicts (i.e., conflicts occurring due to the side-effects of an action) between norms and action choices [8]. OWL-POLAR reasoning mechanism is extended by implementing algorithms to detect functional conflicts arising due to side-effects of an action (i.e., a policy suggests some action that has a side-effect that is prohibited by some other policy) and pre-conditions of action (i.e., policy suggests some action, but its pre-condition cannot be fulfilled due to some prohibition).

Consider two policies $\pi_i = \alpha^i \longrightarrow A_{\chi^i:\rho^i}\left(\lambda^i : \varphi^i\right)/e^i$ and $\pi_j = \alpha^j \longrightarrow B_{\chi^j:\rho^j}$ $\left(\lambda^j : \varphi^j\right)/e^j$. These policies will conflict with each other if the necessary conditions are satisfied i.e., modalities of π_i and π_j are conflicting (Condition 1) and

π_i and π_j are active for the same policy addressee in the same state of the world Δ (Conditions 2, 3 and 4).

1. A conflicts with B. That is, $A \in \{O, P\}$ while $B \in \{F\}$.
2. There exists a substitution σ_i s.t. $\Delta \vdash \left(\alpha^i \wedge \rho^i\right) \cdot \sigma_i$, but no substitution σ_i' s.t. $\Delta \vdash \left(e^i \cdot \sigma_i\right) \cdot \sigma_i'$
3. There exists a substitution σ_j s.t. $\Delta \vdash \left(\alpha^j \wedge \rho^j\right) \cdot \sigma_j$, but no substitution σ_j' s.t. $\Delta \vdash \left(e^j \cdot \sigma_j\right) \cdot \sigma_j'$
4. $\chi^i \cdot \sigma_i = \chi^j \cdot \sigma_j$

Algorithm 1 Anticipate if π_i may functionally conflict with π_j.

```
1: Input:   Policy π_i = α^i ⟶ A_{χ^i:ρ^i} (λ^i : φ^i) /e^i,
            Policy π_j = α^j ⟶ B_{χ^j:ρ^j} (λ^j : φ^j) /e^j
            type
2: if ( (A ∈ {O, P} and B ∈ {F}) then
3:    ⟨Δ, σ_i⟩ = freeze(α^i ∧ ρ^i)
4:    rs       = query(Δ, α^i ∧ ρ^i)
5:    for all (σ_k ∈ rs) do
6:       ⟨Δ, σ_j⟩ = update(Δ, (α^j ∧ ρ^j) · σ_k)
7:       if (isConsistent(Δ)) then
8:          if (query(Δ, e^i · σ_i) = ∅ and query(Δ, (e^j · σ_k) · σ_j) = ∅) then
9:             if (type = sideeffect ) then
10:               return checkSideEffects(λ^i, λ^j, σ_k, Δ)
11:            end if
12:            if (type = precondition ) then
13:               return checkPreconditions(λ^i · conjunct_query_preconds, λ^j, σ_k, Δ)
14:            end if
15:         end if
16:      end if
17:   end for
18: end if
19: return false
```

Algorithm 1 is presented for checking the necessary conditions stated above. Two policies π_i and π_j as specified above and the type of functional conflict that is being checked for, are expected as inputs by the this algorithm. The first step of the algorithm is to test if A conflicts with B (line 2). If they are conflicting, testing further requirements is continued with. A canonical state of the world Δ, in which π_i is active, is created by freezing [1] $\left(\alpha^i \wedge \rho^i\right)$ with a substitution σ_i;

[1] In order to test whether qA subsumes qB, the standard technique of query freezing is used to reduce query containment problem to query answering in Description Logics [5]. In this technique a canonical knowledge-base is built from the query by replacing variables in the query with fresh individuals, adding each individual appearing in the query to the canonical knowledge-base and inserting relationships between individuals and constants defined in the query into the canonical knowledge-base. As a result of this process, the canonical knowledge-base contains a pattern that exists only in ontologies that satisfy the query.

mapping the variables in $\left(\alpha^i \wedge \rho^i\right)$ to the fresh individuals in Δ (line 3). Δ is then queried with $\left(\alpha^i \wedge \rho^i\right)$ (line 4). The results of this query satisfy $\left(\alpha^i \wedge \rho^i\right) \cdot \sigma_i$. For each σ_k satisfying $\left(\alpha^i \wedge \rho^i\right) \cdot \sigma_i$, Δ is updated by freezing $\left(\alpha^j \wedge \rho^j\right) \cdot \sigma_k$, without removing any individual from its existing *ABox* (line 6). As a result of this process, σ_j is the substitution mapping the variables in $\left(\alpha^j \wedge \rho^j\right) \cdot \sigma_k$ to the new fresh individuals in the updated Δ, so that $\chi^i \cdot \sigma_i = \left(\chi^j \cdot \sigma_k\right) \cdot \sigma_j$. The consistency of the resulting state of the world Δ is tested (line 7). If this is not consistent, it is concluded that it is not possible to have a state of the world satisfying the requirements. If the resulting Δ is consistent, the expiration conditions of the policies are checked. If both policies are active in the resulting state of the world (line 8), testing for the sufficient conditions is continued with. If policies are being checked for functional conflicts arising due to side-effects then (line 9), the function 'checkSideEffects' is called and the value returned by the function is returned (line 10). If policies are being checked for functional conflicts arising due to pre-conditions then (line 12), the function 'checkPreconditions' is called and the value returned by the function is returned (line 13). If any of these requirements do not hold then *false* is returned (line 19). The functions 'checkSideEffects' and 'checkPreconditions' are explained in the subsequent algorithms.

Algorithm for Detecting Functional Conflicts Arising Due to Side-Effects: The policies π_i and π_j may conflict functionally (due to side-effects) if, in additions to the necessary conditions stated above, the following sufficient condition is also satisfied.

- λ^j is the effect of performing λ^i (Reasoning about an atomic action λ^i and a state λ^j).

Algorithm 2 Check Side Effects

```
1: Input: λ^i, λ^j, σ, Δ
2: s = clone(Δ)
3: res1        = query(s, λ^j · σ)
4: applyActionToState(λ^i · σ, s)
5: res2        = query(s, λ^j · σ)
6: if ((res1 = ∅)and(res2 ! = ∅)) then
7:    return true
8: end if
9: return false
```

Algorithm 2 is used to check side-effects of an atomic task. Permitted/obligated action (λ^i), forbidden state (λ^j), substitution (σ) for which both the policies are active for the same individuals and the state of the world (Δ) where both the policies are active for the same individuals at the same time, are expected as inputs. The state of the world (Δ) is cloned (line 2). The clone is queried for ($\lambda^j \cdot \sigma$) and the results are stored in the set 'res1' (line 3). The atomic action ($\lambda^i \cdot \sigma$) is applied on the clone (line 4). Again the clone is queried for ($\lambda^j \cdot \sigma$) and the results are stored in the set 'res2' (line 5). If 'res1' is empty and 'res2'

is not empty then it is concluded that λ^j is an effect of performing λ^i and *true* is returned, else *false* is returned.

Algorithm for Detecting Functional Conflicts Arising Due to Preconditions: The policies π_i and π_j may conflict functionally (due to preconditions) if, in additions to the necessary conditions stated above, the following sufficient condition is also satisfied.

- λ^j is a precondition of λ^i (Reasoning about an atomic action λ^i and a state λ^j).

Algorithm 3 Check Preconditions

1: **Input:** $conjunct\_query\_preconds$, λ^j, σ, Δ
2: $s = clone(\Delta)$
3: $\langle s, \_ \rangle = update(s, conjunct\_query\_preconds \cdot \sigma)$
4: $rs = query(s, \lambda^j \cdot \sigma)$
5: **if** $(rs \; != \emptyset)$ **then**
6: return **true**
7: **else**
8: return **false**
9: **end if**

Algorithm 3 is used to check pre-conditions. A conjunction query of preconditions of the permitted / obligated action ($conjunct\_query\_preconds$), forbidden state (λ^j), substitution (σ) for which both the policies are active for the same individuals and the state of the world (Δ) where both the policies are active for the same individuals at the same time, are expected as inputs. The state of the world (Δ) is cloned (line 2) and the clone is updated by freezing ($conjunct\_query\_preconds \cdot \sigma$) (line 3). The clone is queried for ($\lambda^j \cdot \sigma$) (line 4). If the result is not empty then, it is concluded that one of the preconditions of the permitted/obligated action i.e., one of the elements of $conjunct\_query\_preconds$ is the same as the forbidden state λ^j and *true* is returned (line 6), else *false* is returned (line 8).

6 Conflict Resolution

Various strategies can be used to resolve conflicts, e.g. adding a new policy, modifying action constraints of a policy, modifying activation window of a policy, norm-curtailment [14], and prioritising policies etc. However, addition of new policies or modification of the conflicting policies might introduce new conflicts. Hence, new/modified policy must be checked for consistency against all existing policies. Prioritising policies is guaranteed not to introduce new conflicts; re-checking of the whole set of policies is not required.

Classic forms of policy prioritisation are lex-superior (policies/norms from higher authority take precedence) and lex-posterior (most recent policies/norms take

precedence) [8]. In the lex-specialis technique generic policy is overridden by specific policy. All these techniques do not permit a resolution that involves different policies taking precedence depending on context. Also, it is not appropriate to apply one resolution method to resolve all the conflicts that occur in the system [6]. User-defined precedence may play an important role in resolving conflicts. Policy prioritisation based on cost of violation, specificity and user-input is implemented in Polar Agent.

Extending OWL-POLAR further, a policy refinement method for resolving conflicts, is implemented. In this method, a list of references to overriding policies (e.g. policy names) is maintained by overridden policy. Consider two conflicting policies $\pi_i = \alpha^i \longrightarrow A_{\chi^i:\rho^i}\left(\lambda^i : \varphi^i\right)/e^i/\Pi^i$ and $\pi_j = \alpha^j \longrightarrow B_{\chi^j:\rho^j}\left(\lambda^j : \varphi^j\right)/e^j/\Pi^j$. Where Π^i and Π^j are the sets containing references to overriding policies. Let us consider that after using standard techniques for deciding precedence, policy π_i overrides π_j. Hence, Π^j will be modified as $\Pi^j = \{\pi_i\}$. Consider that policy $\pi_k = \alpha^k \longrightarrow C_{\chi^k:\rho^k}\left(\lambda^k : \varphi^k\right)/e^k/\Pi^k$ also conflicts with π_j and π_k overrides π_j. Hence, Π^j will be modified as $\Pi^j = \{\pi_i, \pi_k\}$. Conflict detection mechanism of OWL-POLAR is augmented so that resolutions are taken into account.

7 Conclusion and Future Work

The key questions addressed in this paper are: how to identify conflicts that are most relevant to the domain and goals of organisation, and how to resolve conflicts that are most important to the goals of organisation. The activity prioritisation model is used to identify the most significant activities that can be performed while achieving a goal. Active policies that regulate high priority activities are reasoned about to detect logical and functional conflicts. This ensures identification of conflicts that are most relevant to the domain and goals of organisation. Only those conflicts that have very high chances of impairing the goal-achievement (i.e., conflicts involving high priority activities) will be resolved using the conflict resolution techniques described in the paper. Depending on significance of goal, resource availability and computational capacity a threshold can be defined for number of activities to be considered for conflict detection and resolution. A threshold for number of conflicts that can be resolved offline can also be defined. Thus, less policy violations and decreased possibility of failures in plan execution at runtime is ensured by the activity prioritisation model.

In future, the reasoning mechanism of OWL-POLAR will be extended to detect other types of functional conflicts, e.g. inaccessible input/output and incoherent domain etc. The activity prioritisation model will be evaluated; and it will be investigated if it identifies an optimum set of conflicts to resolve, such that the goal is achieved with likelihood $\geq$ threshold. Given a set of policies, user-performance in resolving conflicts and refining policies will be evaluated. Two conditions, control and aided, will be considered for two scenarios of similar complexity. It will

be investigated and evaluated if the automated support mechanisms developed for helping policy authors in resolving policy conflicts and refining policies can also aid collaborative authoring and refinement.

References

1. W. O. W. Group, OWL 2 Web Ontology Language: Document overview `http://www.w3.org/TR/owl2-overview`
2. Brin S., Page L.: The Anatomy of a Large-Scale Hypertextual Web Search Engine. Computer Networks and ISDN Systems. 30(1-7), 107–117 (1998)
3. Broersen J., Dastani M., Hulstijn J., Huang J., van der Torre L.: The BOID Architecture - Conflicts Between Beliefs, Obligations, Intentions and Desires. In proceedings of the 5th International Conference on Autonomous Agents, pp 9–16. ACM Press (2001)
4. Castelfranchi C.: Formalizing the Informal?: Dynamic Social Order, Bottom-Up Social Control, and Spontaneous Normative Relations. Journal of Applied Logic. 1(1-2), 47–92 (2003)
5. Şensoy M., Norman T.J., Vasconcelos W.W., Sycara K.: OWL-POLAR: Semantic Policies for Agent Reasoning. In proceedings of the 9th International Semantic Web Conference, LNCS, vol. 6414, pp. 679–695. Springer-Verlag (2010)
6. Dunlop N., Indulska J., Raymond K.: Methods for Conflict Resolution in Policy-Based Management Systems. In proceedings of the 7th International Enterprise Distributed Object Computing Conference, pp 98–109. IEEE Computer Society (2003)
7. M. Johnson, J. Karat, C.-M. Karat, K. Grueneberg.: Usable Policy Template Authoring for Iterative Policy Refinement. In proceedings of the 2010 IEEE International Symposium on Policies for Distributed Systems and Networks, pp 18–21. IEEE Computer Society (2010)
8. Kollingbaum M.J.: Norm-Governed Practical Reasoning Agents. Dept. of Computing Science, University of Aberdeen (2005)
9. Meneguzzi F.: Extending Agent Languages for Multiagent Domains. University of London, King's College London (2009)
10. Oh J., Meneguzzi F., Sycara K., Norman T.J.: An Agent Architecture for Prognostic Reasoning Assistance. In Proceedings of the 22nd International Joint Conference on Artificial Intelligence, pp. 2513–2518. IJCAI (2011)
11. Sycara K., Norman T.J., Giampapa J.A., Kollingbaum M.J., Burnett C., Masato D., McCallum M., Strub M.H.: Agent Support for Policy-Driven Collaborative Mission Planning. The Computer Journal, 53(5), 528–540 (2009)
12. Tonti G., Bradshaw J.M., Jeffers R., Montanari R., Suri N., Uszok A.: Semantic Web Languages for Policy Representation and Reasoning: A Comparison of KAoS, Rei, and Ponder. In the proceedings of 2003 International Semantic Web Conference, LNCS, vol. 2870, pp. 419–437. Springer-Verlag (2003)
13. Uszok A., Bradshaw J.M., Breedy M.R., Bunch L., Feltovich P., Johnson M., Jung H.: New Developments in Ontology-Based Policy Management: Increasing the Practicality and Comprehensiveness of KAoS. 2008 IEEE Workshop on Policies for Distributed Systems and Networks, pp 145–152. IEEE Computer Society (2008)
14. Vasconcelos W. W., Kollingbaum M.J., Norman T.J.: Normative Conflict Resolution in Multi-Agent Systems. Auton Agent Multi-Agent Systems. 19(2), 124–152 (2009)