NPE - A Conceptual Model and Language for the Representation of Norms

Andrew Douglas
School of Electronics and Computer Science
University of Southampton
UK
Email: akd07r@ecs.soton.ac.uk

Dr Robert Walters
School of Electronics and Computer Science
University of Southampton
UK
Email: rjw1@ecs.soton.ac.uk

Dr Gary Wills
School of Electronics and Computer Science
University of Southampton
UK
Email: gbw@ecs.soton.ac.uk

***Abstract*—Originating in Deontic logic, norms and normative facts enable philosophers, designers and developers to reason over actions an actor may or may not perform with respect to social rules. Normative concepts have the power to enable us to reason with flexibility over actions and beliefs. In the context of Service Oriented Architectures (SOAs), where communities of potentially autonomous, heterogeneous, independently designed components may be interacting; the introduction of norms to influence behaviour could be indispensable. Norms should not be seen as constraints to behaviours, rather they are metrics against which behaviours can be measured. Applying this analogy to components in SOAs enables participants to reason over behaviours with the flexibility to react to changing community norms whilst putting their own interests first.**

This paper presents a formalism for the representation of norms in agent and service based architectures. Specifically, we propose a method for modelling norms in expected behaviours of actors. The result of our work provides a language, NPE, enabling participants to reason over norms, actions and beliefs.

I. Introduction

The increasing deployment of services as a viable distributed computing paradigm has promoted openness, flexibility and the standardisation of heterogeneous architectures. As SOA has matured, so the potential for the emergence of complex social networks of services has been witnessed. SOA provides viable high-level abstractions, models and approaches to enable the exposing of functionality in such a way as to promote the creation of complex workflows utilising multiple groups of services.

As with any society, the complex distributed eco-systems which we are witnessing the emergence of, require regulating through coordination and administration. In a wider context, societies as a whole utilise a notion of norms for much of this regulation. Norms offer restrictions on behaviour, or alternatively metrics against which behaviours can be compared. Often disputes can be avoided or resolved with greater efficiency not by rules or laws, but by informal social norms [1].

Web services have been recognised as one of the most promising approaches for the realisation of large scale SOAs and service oriented computing (SOC). One of the most striking advantages of the Web Service paradigm is that the task of aggregating complex workflows from libraries of simpler services, or even atomic services is relatively simple. Semantic Web Services take SOA and Web Service architectures one step further. By enabling actors to exchange semantically rich data complex tasks can be undertaken autonomously. One of the most interesting uses for semantic data with regards to SOC is the marking up of service descriptions, behaviours and workflows. This enables machine mediated partner selection, automated workflow production and many other activities, which would be useful to an actor wishing to behave autonomously. With increasingly complex SOAs and autonomy aided by Semantic Web Service techniques, there are many uses for norms. Most notably in behaviour control, and to aid service discovery. This research focuses on modelling norms with regards to the behaviours they restrict and provides a language to enable the representation of norms for either of the above tasks.

A. Norms

Deontic theory tells us that norms enable reasoning over what actions an actor may or may not perform with respect to social rules [2]. However, rules and norms differ in severity and emergence. Rules are typically concise, often governing principals or regulations laid out by an overseeing director. Norms are more fluid, typically standards or patterns, enforced by society. Norms are far more likely to change during the lifetime of a society, and may be specific to individual groups within a society. Norms shared by members of a group have authority over potential actions by way of approval or disapproval by other members. Two classes of social norms exist; formal and informal [3]. Informal norms exist because of trends or patterns of behaviour within a group and are often not sanctioned. Formal norms exist in conjunction with rules, providing penalties for dissent or non conformity. As societies become larger and increasingly decentralised, rules become harder to enforce, so norms become a tool to achieve social order [4].

Norms can change from society to society and between groups of associated actors. On a global stage there are many different societies and groups and it is usual for each to have it's own specific norms. Being able to react to differing norms while interacting with partners around the world is critical in todays business markets [5]. Norms offer a novel tool for both service adaption and as a control for social order on a global scale.

II. Agents and Service Oriented Architectures

A key part of many modern distributed systems is the participation of autonomous, self-interested agents. The flexibility of current systems enables actors to behave according to there own decision making metrics. Intelligent software agents have become the leading paradigm for autonomous distributed software design. They are by definition; autonomous, social, reactive and proactive [6]. All of which characteristics make them well suited for participation in any SOC approach [7].

Both SOC and and agent paradigms are concerned with enabling simpler access to data whilst potentially retaining a greater degree of control. As both models offer the flexibility to handle the unpredictability of connections on an open system such as the Web, they have the power to "become an essential part of most Web-based applications, serving as the glue that makes a system as large as the Web manageable and viable" [8]. Agents have evolved the ability to operate with-in SOAs, consuming Web Services to gain additional functionality they might not otherwise have had. Combining SOC with agent technologies aims to overcome the flaws each paradigm has, and produce a cohesive structure to enable more complex problems to be solved smarter and faster.

Agents can typically handle the complex workflows and task sets required by current businesses. Agent-oriented design paradigms have the flexibility and dynamism which can enable them to partake in interactions over the Web, which by it's nature is both open and uncertain. The ability to plan around failures and react to changes mean that agents will soon play a large role in SOC. It is likely that this role will be made even more important as increasing numbers of Semantic Web Services are included in SOAs. Semantically marking up services enables agents to discover new services and compose new workflows autonomously. Service composition is an important issue in SOC, agents capable of efficient selection and composition of Web Services can solve problems faster than others, overcoming barriers such as untrustworthy partners, unreliable connections and volatile pricing. Service selection for Semantic Web Services can be split in to three main areas; service discovery, service selection and invocation (binding). For an agent to autonomously consume a Web Service, it must be able to do all three. This will require the service to provide semantically marked up data about it's self to aid selection. The provision of semantically rich descriptions is one that has been covered in past research [9], [10], [11], [12]. We believe that there is a gap in previous research, enabling norms to be leveraged to aid service selection. By using norms to reason over behaviours articulated in a service description a selection can be made which takes into account social rules and values.

Both SOC and agent architectures are capable of forming complex distributed systems. Participants interacting within these systems may group together to form societies, either by design or unintentionally as the result of the pull of a specifically strong incentive. As with any society, control in a closed system, or on a small scale is easy to achieve. However, in an open system such as the Web, or with large groups of collaborating agents and services, achieving control can be very difficult. Here socially enforced rules in the form of norms can be useful to aid in controlling interactions. Formal or informal norms can be created and shared between participants in a distributed fashion. Failure to adhere to a norm would be punished by social exclusion or other participants not being as willing to interact. Utilising norms in this fashion enables social rules to change over time or to alter between groups, requiring a design of agent which could reason over normative events. Norms themselves could even be semantically annotated and published in repositories for the aid of agents who might wish to start interacting with a new social group.

Following a short discussion about the use of BDI agents, the rest of this paper will introduce NPE as a mechanism for the inclusion of norms into SOC and agent based paradigms. As an abstract model followed by a proposed language.

A. BDI Agents

There are many different types of agent however, the one which fits this researches view of what an agent needs to handle interactions with SOAs and process norms is the BDI architecture. The notions of beliefs (B), desires (D), and intentions (I) are well know in AI research [13]. Agents which leverage primarily intention based approaches are commonly referred to as BDI agents. In a BDI approach the choices an agent makes are influenced by what it believes about the surrounding world (B), states it desires to achieve (D) and the plans it has formulated to bring about its desires (I) [14]. The BDI runtime cycle works in general terms in the following manor; generate options from desires, select an option, update intentions, execute intentions, update belief base [15]. By using this workflow BDI agents step though their worlds situation by situation, always aware of the visible effects or their and any other agents behaviours.

It is the clear knowledge of beliefs and desires which make BDI agents a good choice for the application of advanced social technologies such as norms. A number of research papers have in the past suggested augmenting BDI architectures with a range of new abilities to aid social interactions [16], [14], [17], [18], [19]. From the perspective of this research having a clear and verifiable set of all the agent's beliefs gives BDI the advantage over other methodologies when applying norms. Having the ability to reason over actions which have happened, are proposed by an external source, or intended by the agent allows norms to be included at all points.

NPE has been designed with BDI in mind. Specifically NPE relies on an ability to reason over current beliefs to form conditions for activation or deactivation of a norm. Coupled with this every norm controls an activity which may or may not be allowed depending on the normative fact in question. It is intended that in final implementations, an NPE validating step could be added to the BDI control loop before desires are created. However this paper will not outline this implementation.

III. Norms for Behaviours

Norm based agents allow notions of normality and normal behaviours to influence and motivate their plans and actions. Agents utilising norm based approaches allow their plans to be shaped not only by concrete beliefs and rules, but also by more fluid notions of what might be ideal (or normal) for a situation. The norms which influence such agents are representations of social concepts existing between agents in any agent society. The same could be said to be true of any norm implemented in any society by any actor.

This research presents a model for normative facts, and suggests how such facts might be utilised. Normative facts are modelled as tuples containing all of the elements required for them to have effect. This includes triggering and deactivating conditions, and role and subject limiters. The following section outlines the components of our model for norms, and gives a short example.

This work is influenced heavily by agent research. In particular parallels can be drawn with work by prominent researchers in the field of norm-based agents [16], [17], [20], [21].

A. Scope of These Proposals

Within the scope of this research norms can be treated as social concepts (not hard and fast beliefs), being both role and society specific (norms change between societies and are specific to certain actors fulfilling a set role in the current society). Some of the details involved in this model have been omitted from this paper. Our aim is to give the reader an understanding of what a normative fact is, and how one might be represented. Details about handling sets of norms, or utilising norms for any one specific purpose have been omitted as they fall outside the scope of the creation of normative facts.

B. An Abstract Model for the Representation of Norms

To uniquely identify concepts and ideas within the following model the identifier $label$ is defined. A $label$ is in it's own right a concept, however it is separate from the core concepts involved in this model (norms, roles, actions, exceptions and actors). Within the scope of this abstract model, the superset $LABLES$ and it's subsets(L^{Norms}, L^{Actors}, L^{Roles} and $L^{Exceptions}$) are all finite. This stems from the one-to-one mapping between each $label$, and an element in the model (be it an $actor$, $norm$, $exception$, or $role$).

1) Roles: Actors fulfil roles within a system. Each $actor$ is identified by a $label$ from the finite set L^{Actors}. Norm specifications address a $role$ not individual actors. However, actors must be defined as they are required to identify violators in any implementing system.

Within any system there is a finite set of $ROLES$. In this model each $role$ is identified uniquely using a $label$ from the finite set L^{Roles}. Norms are $role$ specific - every $norm$ must have at least one $role$ under which it can be triggered. At the most basic level a $norm$ can be seen as a characteristic of their parent $role$, limiting the actions an $actor$ in that $role$ can undertake.

Each $role$ is identified by a $label$ from the set L^{Roles}. Each $actor$ holds a non-empty set of roles such that $ROLES \subseteq L^{Roles}$. Norms are $role$ sensitive and every $norm$ applies to a non-empty set of roles.

2) Actions: Actions are activities to be undertaken by an $actor$ fulfilling the $role$ addressed by the current $norm$. Actions are $role$ specific. For the purpose of this model, actions are treated as labels only. Each $action$ is identified by a $label$ from the finite set $L^{Actions}$.

An $action$ A defines a prescribed activity to be undertaken by an actor. Every $action$ A has exactly one $actor$ A^{Actor}. The set $ACTIONS = \{A_1, .., A_n\}$ is a finite set of actions available in the current world. Within the current world a set of identified actions exists such that $ACTIONS \subseteq L^{Actions}$.

3) Norms: Basic normative ideas and the concept of a norm are essential to any norm-based system. This is just as true for the following norm-based service selection framework, as it is for norm-based agents. The following model relies on three core norms; Permit, Forbid and Obliged. Any action not covered by one of the above norms can be assumed to be unrestricted in regards to this model. Norms cannot be simply combined, they may, however, be nested in certain orders.

With the above definitions in place it is now possible to define a conceptual representation of a norm. This model caters for three core types of norms; permissions, obligations, and forbidden. All other restrictions originate from these core types. Of the three types, permission enables an actor to carry out an activity. Forbidden is the contrary to permission in that it denies the actor the ability to undertake an activity. Obligate, is different in that it stipulates that not only is the actor permitted to undertake an action, they are forbidden not to undertake the action.

The set $L^{Norms} = \{perm, obli, forb\}$ is the set of labels used to identify the three core $norms$; Permission ($perm$), Obligation ($obli$), and Forbidden ($forb$). Having acquired a $label$ for each of the three core norms used within this system, it is now possible to define the following four valid formulations for activities within a norm declaration:

- $perm(a)$ = "Permitted to undertake action a"
- $perm(\neg a)$ = "Permitted to not undertake action a"
- $\neg perm(a)$ = "Not permitted to undertake action a"
- $\neg perm(\neg a)$ = "Not permitted not to undertake action a"

These norms are oversimplified, but show how a very simple representation of a norm could be created. There is much information missing from this representation, however, it does show that both norms and the activities they govern can be manipulated. With this representation in mind we can move on to define a few simple rules for the interaction of norms:

- $perm(a) \equiv \neg forb(a)$
- $perm(a) \wedge perm(b) \equiv perm(a \wedge b)$
- $obli(a) \equiv forb(\neg a)$
- $\neg perm(\neg a) \equiv obli(a)$

- $\neg perm(a) \not\equiv perm(\neg a)$
- $perm(a \to b) \not\equiv perm(b \to a)$
- $obli(a \to b \wedge c) \equiv (obli(a) \to obli(b \wedge c))$

Using the labels defined above, a model norm can be created. A norm in our model can the be defined as a tuple:

$$N = \langle n, l^{Actions}, R, s, c, d, e \rangle$$

Where:

- $n \in L^{Norms}$ (the label attributed to the current type of norm; perm, obli, or forb)
- $l^{Actions}$ is a specified set action labels where each $l \in L^{Actions}$
- $R \in L^{Roles}$is the role from which the action originates.
- $s \in L^{Subjects}$ is the subject upon which the action may act
- $c \in L^{Conditions}$ is a condition under which the action may be initiated and the norm activated. The boolean condition "true" can be used to denote a norm which will always be triggered given a matching role, subject, and activity.
- $d \in L^{Deactivations}$is a deactivation condition, under which an active norm will be deactivated. This condition have no significance when the norm is inactive. The boolean condition "false" may be used to signify a norm which once activated will always hold for a matching role, subject, and activity.
- $e \in L^{Exceptions}$ is an exception to be triggered if the norm is violated

The preceding tuple contains all of the information required to define a norm for an action or set of actions. To make each norm easier to read and differentiate, the syntax found in Figure 1 has been adopted. This puts the normative type declaration first forming a more familiar expression.

4) Example: With the aforementioned definitions in place, norms can be explicitly defined using formal semantics. For example, the normative statement in Figure 1 expresses that: Actors fulfilling the role R are obliged to undertake the action a on subject s when the precondition c is matched. Breaking this norm will trigger exception e. The norm will stay active so long as the deactivation condition d is not true. Such a formalisation, allows this abstract system to interpret the given norm as a given belief, and apply it to a plan using a provided matching algorithm.

$$obli(a, R, s, c, d, e)$$

Fig. 1. Example Norm 1

A fuller example can be demonstrated by the norm in Figure 2. This norm forbids actors in currently in the role "$shopper$" from undertaking the action "$leave\_shop$", once the condition "$goods\_in\_basket$" has been triggered. This norm can be deactivated once the condition "has_paid" has been satisfied. If the actor does "$leave\_shop$" while this norm is in the set ACTIVE, then the exception "shop_lifter" is triggered. This example is presents a more comprehensive view of the representation of a norm in our system. The subject "$goods$" is included for future use, although it is currently not applied.

$$forb(leave\_shop,\ shopper,\ goods,\ goods\_in\_basket,\\ has\_paid,\ shop\_lifter)$$

Fig. 2. Example Norm 2

C. Triggering

When processing a plan or observation the underlying normative architecture deconstructs the component activities into a sequential list of actions, states, roles, subjects and initial beliefs. This sequence is then checked against the normative model. Each norm has an activation condition c such that $c \in L^{Conditions}$. Conditions can be evaluated by an implementing system to evaluate either true or false; $c = \{true, false\}$. When an activation condition is met, the norm is the eligible for activation. Norms are triggered when the plan features an actor fulfilling a role "R", undertaking an action "a" on a subject "s", where R, a, s match the required role, action and subject of a norm in the system, and the preconditions of that norm are true. If the role, action subject, and preconditions all match, then the norm is triggered. Upon triggering the current norm is moved to the ACTIVE set. $ACTIVE$ is the set of all active (triggered) norms, such that $ACTIVE = \{N_1, .., N_i\}$. It must be true that $ACTIVE \subseteq NORMS$ so that any norm in $ACTIVE$ must also exist in the superset $NORMS$, the global set of all norms in the system. If the current norm is violated, it's exception clause is triggered.

Once active, the deactivation condition can then be evaluated. If the deactivation condition is evaluated to "true", then the norm is removed from the set active and becomes deactivated (and the activation condition again starts to be tested). The deactivation condition may never be set to boolean "true" (as that would invalidate the norm), it may however be set to boolean "false" to signal a norm which once active will always hold for a given combination of role, subject and action.

D. Violations

Violating an active norm causes an exception to be triggered. Only norms currently in the set $ACTIVE$ may violated and, thus trigger exceptions. Exceptions trigger new norms or force actors to undertake preset actions. Any norm can have a maximum of one exception (although exceptions can be nested as can norms). Exceptions are identified using labels from the finite set $L^{Exceptions}$. A norm specification will hold a single exception $e \in L^{Exceptions}$. Triggering an exception may or may not force the abandonment of any norm specification.

E. Consistency

For any norm based service selection model to be successful, the norms defined, stored and handled by the system must be consistent. Consistency applies to both the norms and the actions which the norms regulate. To ensure that all norms in use are consistent, it is imperative that the initial norm base is checked at startup and rechecked every time a new norm is added. Norms need to be consistent individually in regards to the actions they regulate, and consistent in regards to any interdependencies which may exist between multiple norms.

Consistency checking for norms, in a norm base of any model, should be assisted by an initial process of normalisation for any normative statements included (to prevent the same norm being defined several ways and to ensure that there are several consistent methods for defining the majority of required norms). This process should simplify all the norms in the base to a number of default types. This process both makes consistency checking easier to fulfil and simplifies the norm base.

It is important to note, that there are two types of constancy; local consistency and global constancy. Norms are required to be locally consistent (every norm in a local set must be consistent with every other). No norm can be added which might cause the local set of norms to become inconsistent. Globally, norms within the local set will inevitably be inconsistent with.

IV. A Language for Norm Process Expectations

For any system to be viable and useful developers will require both a concrete implementation of the abstract model and a language in which to define and store any norms to be implemented. It is true that a system could be created with no base (external) language for norm definitions and only basic internal representations. Such a system however, would have a very limited appeal in wider applications. Creating a language in which norms can be represented and interpreted by both computers and humans, enables the rapid creation of a norm base for any existing or new system.

This section outlines a language for the description of norms in regards to processes (actions). The language for norm-based process expectations (NPE), aims to provide a mechanism to easily create sets of norms which relate to practices during an interaction. This language has been built on a number of existing protocols.

A. Example Situation

An example situation will be used to illustrate the type of actions and processes norms defined in an NPE document might be expected to govern. For the purposes of this research, it was decided that a realistic example of a service providing an interface to a business resource should be used. The following example is based on two competing fictional B2C service providers, referred to as Frango and Dupeme. Both of these companies provide access to an important resource through a publicly visible Web Service. Both of these companies also provides information about their service by way of an OWL-S document with a process description. In order for an consumer to access the resource, they must choose which company to use, and follow the workflow described in the process model. At first glance Frango and Dupeme are virtually indistinguishable from one another. They both provide access to a resource, they both required very similar sets of inputs. The only real differences are in the workflows, exposed by the OWL-S process models, which are required to be followed for the service to be used. The question we are attempting to answer with this very simple example is; given two workflows, can a consumer employ norms to differentiate between the two services and thus choose one?

B. Prolog-based NPE

The initial NPE language is Prolog-based, and intended to be used to prove the concepts behind the NPE model. Prolog was chosen as its syntax already consists of sets of clauses (rules and goals) which map well onto norms, and concepts outlined previously.

Following strictly the definitions for a norm prescribed above, Prolog-based NPE is essentially a distilled version of a pure Prolog representation of a normative action. A complete normative statement can be represented as a single Prolog-based NPE rule. Correctly formatting each fact can improve readability, without the need to change the structure of any norm. An example of correct formatting, and the basic structure of Prolog-based NPE can be found in Figure 3. As well as improving readability, Prolog-based NPE also adds the notion of exceptions to facts. Correct processing of exceptions will require a specially modified Prolog interpreter. However, the syntax of Prolog has been adhered to as best as it can be, so as to enable a new processor to be synthesised out of an old Prolog one, as well as to enable syntax checking without the need for a new processor.

Prolog-based NPE is the starting point in the investigation, of how to represent norms in a machine and human readable form. It is, however, not perfect; the resultant Prolog facts are large and non-normalised. If designed without care multiple facts could represent the same normative statement. However, many of these problems could be overcome with the careful design of facts.

An initial norm example has been created grounded in the real-world. Figure 4 shows this simple norm which governs a basic everyday shopping interaction. The norm states that a shopper is obliged to pay (#pay) for goods when they hold the belief that they have finished their shopping (#shopping_complete). This norm is deactivated when an exchange of credit has been complete (#transfer_complete). If this norm is violated a failure_to_pay_exception is thrown (in the real-world possibly resulting in the action #police_called). This norm roughly equates to having to pay for goods when shopping is finished.

Following on from this we can convert the norm defined as a tuple in Figure 2 into Prolog-based NPE (Figure 5). To do this a “forb” prolog fact is created and the action leave_shop is converted to the identifier #leave_shop. Handling of activation and and deactivation conditions is carried out by the addition of a keyword condition, in this case “goods_in_basket” translates to “in(#goods_in_basket)” and “has_paid” becomes “once(#paid)”. The meaning of these two conditions can be taken as true if; the actor has the belief (in their belief base) that there are currently goods in their basket in the and once the

```
<npe_document> ::= <IDENTIFIER> "{" <norm_fact> "}".

<norm_fact> ::= ( "perm"
              | "obli"
              | "forb"
              )
              "("
                 <action> ","
                 <roles> ","
                 <subject> ","
                 <preconditions> ","
                 < deactivation_conditions > ","
                 <exception>
              ")."

<action> ::= "#"<IDENTIFIER>

<roles> ::= "[" <role> ( "," <role> )* "]"

<role> ::= <IDENTIFIER>

<subject> ::= <IDENTIFIER>

<preconditions> ::= <condition>

< deactivation_conditions > ::= <condition>

<condition> ::= <prolog_predicate>

<prolog_predicate> ::= <atom> | <atom>"(" <term_list> ")"
<term_list> ::= <term> | <term_list> "," <term>
<term> ::= <variable> | <structure>
< structure > ::= <atom>"(" <term_list> ")"
<atom> ::= <variable>
<variable> ::= <text> (<text>)*
<text> ::= <char> | <charupper> | <digit> | <symbol>
<char> ::= a | b | c | d | ... | y | z
<charupper> ::= A | B | C | D | ... | Y | Z
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<symbol> ::= _ | #

<exception> ::= "throw( error (" <IDENTIFIER> "))"
```

Fig. 3. Prolog-based NPE (In BNF Form)

```
obli (
      #pay,
      Shopper,
      goods,
      in (#shopping_complete),
      once(# transfer_currency ),
      throw( error ( failure_topay_exception ))) .
```

Fig. 4. An Example of the Norm Obligated

actor has undertaken the action “#paid”. A full explanation of keywords and conditions can be found in the next section. The NPE representation of the norm also contains a representation of the exception which would be thrown on violation. In this case the exception is mapped directly from the original tuple form.

These two examples also show how two norms can be very different in their appearance and terms, but govern very similar instances. Both of these norms attempt to make sure that the shopper pays for goods before leaving the shop. An important task for any implementing system, would be to decide whether to allow both norms to co-exist.

```
forb (
      #leave_shop,
      Shopper,
      goods,
      in (#goods_in_basket),
      once(#paid),
      throw( error ( shop_lifter ))) .
```

Fig. 5. An Example of the Norm Forbidden

1) Keywords: Prolog-based NPE tries to stick as closely as possible to the original syntax of Prolog. This limitation aims to ensure that interpreters can be easily created, and the testing of documents is simple. On the whole this objective has been achieved successfully, however some changes are necessary to facilitate higher degrees of coupling with process descriptions and handling of complex scenarios. Most notable is the inclusion of identifiers preceded by the character “#” linking NPE facts to regions or tasks in an underlying process model or belief base. These identifiers can be processed normally (handled as literals or ignored) by many standard Prolog interpreters. A specialised NPE processor would then make use of these identifiers for linking to a given process model. A set of specialised queries for condition processing have also been added. Again these are normal Prolog statements, but they have special meaning in an NPE setting. As with the above identifiers these queries would be passed to a specialised NPE interpreter for further evaluation. Most of these queries relate to evaluating the contents of current states, the agents belief base, or the current process model being evaluated. These keywords include:

- in(#region)
  - A query to test if at this time, the actor believes that it is (or would be) “in” the given task “#region”.
- Once(#task)
  - True once the actor evaluates that the task “#task” is being undertaken
- if(#belief)
  - True if the belief “#belief” is found in the belief base of the actor. This may be the actors actual belief base, or one created during the evaluation of any proposed workflow.

2) Exceptions: Exceptions are an important part of NPE. They enable developers to set what happens when a norm is violated. This paper will not go into detail about how exceptions might be defined or stored. Prolog-based NPE contains a mechanism for the definition of which exception is to be thrown on violation. This call would have to be interpreted by the control logic as it see's fit. We can envisage that exceptions are likely to come in two variants; blocking and non-blocking. Blocking exceptions will stop the flow of the NPE interpreter and signal that a norm has been broken that is of "severe" significance. Non-blocking norms will not stop the flow of the interpreter, instead they contain a score or note which is recorded by the interpreter. This could be used later for the grading of workflows. Exceptions could also be used to chain norms together. For example, violation of the norm in Figure 5 would cause the exception "shop_lifter" to be thrown. So long as this is a non-blocking exception, this could cause any norm with the triggering condition "if(error(shop_lifter))" to be triggered during the next interpretation cycle.

ISO Prolog provides a built-in exception handling mechanism through the catch/3 and throw/1 control constructs. However, the technique applied is somewhat limited and certainly does not give the control expected by any developer used to modern object oriented languages. In the scope of the current research, exceptions are seen as calls thrown by the interpreter when it detects that a norm has been violated. The exception call is defined within the triggering norm, and may or may not halt processing.

C. Example

Returning to the previously outlined example, we now have a language in which we can describe a set of norms for the what we believe to be acceptable for a service to expect us to do. It is assumed that the consumer in question is a BDI like agent, with a belief base, the ability to comprehend OWL-S documents, and an NPE interpreter. An external matching service has provided the consumer with the OWL-S documents for Frango and Dupeme, the consumer must now decided which of the services it wishes to interact with. Matching based on inputs and outputs has been unable to differentiate between either. So the consumer extracts the process models from the provided OWL-S documents, and uses an NPE based interpreter to reason over each step in the proposed workflows.

The consumer has a set of norms which includes the two outlined in Figure 6 and Figure 7. When the consumer evaluates the workflow for the Frango service they find that the norm in Figure 6 is violated. This consumer has never used this service before, has no prior knowledge, and no reason to trust it, thus the step which requires the consumer to share their email address with the service triggers the norm. This norm is however non-blocking and so a note placed by the service. The second service for Dupeme is also processed. This is stopped very early on, as before the consumer and service have got to the point when an agreement about the terms of any access to the resource have been made, Dupeme wants the consumer to send credit card and payment details. This violates the second norm (Figure 7). This norm is serious enough to be blocking, so the processing of the Dupeme workflow stops. The consumer is now left with the following choice; use Frango which is asking for my email address despite the fact that I have no reason to trust it with such data, or use Dupeme, a service which triggered a serious norm violation by asking for sensitive payment details before a deal on access has been made. The final choice would be left to a matching algorithm of the consumers choosing, this research would like to suggest that in this case the Frango service would be the best solution, as it violates only a minor norm.

```
forb(
        #send_email,
        Consumer,
        email_address,
        in(# seller_unverified ),
        once(# seller_verified ) || once(
              $transaction_safely_completed ),
        throw( error ( seller_not_verified ))).
```

Fig. 6. Example Norm 3

```
forb(
        #send_payment_details,
        Consumer,
        cc_number,
        **not**( in(#access_agreed)),
        in(#access_agreed),
        throw( error (no_agreement))).
```

Fig. 7. Example Norm 4

D. Scope of This Language

Prolog-based NPE is not perfect. It has been designed to show how a language might be created to represent norms within the abstract model outlined in this paper. Details of how exceptions are to be handled and how an actor might produce a plan against which these norms can be tested have been left out as have concrete implementation issues. It was our intention in this paper to lay down an initial case for this language, and show how it might be used in a fuller system. By limiting the scope it is hoped that this language can be made easier to understand, and that a wider range of applications can be considered.

V. Future Work

We believe the proposed NPE model is applicable to a wide range of different areas. A model for norms has been adopted which can readily be advanced to other technologies such as peer-to-peer systems, or Grid Services. An initial Prolog language has been created, but the simplicity of the underlying model should allow it to be easily delivered using a different technology. The next step we believe is to build an ontology of norms to store this data, with a focus on delivering norms to agents and services on the Semantic Web.

The work presented here offers a model for norms which is applicable to many different areas of use, It is our intention to narrow this use down with a future version aimed at modelling norms in workflows and semantically annotated workflow descriptions.

Future work may also focus on the use of norms in a wider global context. An ontology for norms would enable semantic repositories for norms to be created. As norms differ between groups and societies around the world, an applicable extension would be to have numerous repositories, each holding norms related to a different group. In this way a business wishing to utilise a service in a country or culture differing from there own would be able to search for the applicable norms.

VI. Conclusion

In this paper we have shown how norms can be represented using a simple model. That model has enabled the production of NPE, a language for norms based on Prolog. It has been shown that norms can be a useful tool for agents and in SOC environments. Not only can they control participants in an open computing environment, they can also serve to validate potential actions. Applicable to both the monitoring of external, and the planning of internal actions, NPE is a versatile solution to the question of how norms might be used.

A simple example showed how NPE can be used to represent norms, and what might happen if they are violated. This example showed how norms could be used to differentiate between similar services on the basis of behaviours exposed by there process models. This approach differs from many existing IOPE based matching methods [22], [23] as it enables the differentiation of services based on small differences within proposed workflows.

Our approach has shown that there are numerous ways in which norms can be utilised by actors interacting using SOAs. The flexibility normative approaches contribute to agent and SOC, in enabling participants to adapt to differing social and economic situations make them well suited for use in an ever expanding global economy. A language, NPE, has been suggested as a starting point for the representation norms. Future work will focus on refining this language, making the automated selection and utilisation of norms easier and more available.

References

[1] P. H. Huang and H. Wu, "More order without more law: A theory of social norms and organizational cultures," *Journal of Law, Economics and Organization*, vol. 10, no. 2, pp. 390–406, October 1994.

[2] S. Beller, "A model theory of deontic reasoning about social norms," in *Proceedings of the Twenty-third Annual Conference of the Cognitive Science*. Erlbaum, 2001, pp. 63–68.

[3] F. Dignum, "Abstract norms and electronic institutions," in *Proceedings of International Workshop on Regulated Agent-Based Social Systems: Theories and Applications*, 2002, pp. 93 – 104.

[4] C. Castelfranchi, "Formalising the informal? dynamic social order, bottom-up social control, and spontaneous normative relations," *Journal of Applied Logic*, vol. 1, no. 1, pp. 47–92, Febuary 2003.

[5] J. Logsdon and D. Wood, "Business citizenship: From domestic to global level of analysis," *Business Ethics Quarterly*, vol. 12, no. 2, pp. 155–187, April 2002.

[6] M. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice," *Knowledge Engineering Review*, vol. 10, no. 2, June 1995.

[7] W. Shen, Q. Hao, S. Wang, Y. Li, and H. Ghenniwac, "An agent-based service-oriented integration architecture for collaborative intelligent manufacturing," *Robotics and Computer-Integrated Manufacturing*, vol. 23, no. 3, pp. 315–325, June 2007.

[8] M. Huhns and M. Singh, "Agents are everywhere," *IEEE Internet Computing*, vol. 1, no. 1, pp. 87–87, January 1997.

[9] A. ShaikhAli, O. F. Rana, R. Al-Ali, and D. W. Walker, "Uddie: an extended registry for web services," in *Proceeding of the 2003 Symposium on Applications and the Internet Workshops*, IEEE. IEEE Computer Society, January 2003, pp. 25–29.

[10] P. A. Bonatti and P. Festa, "On optimal service selection," in *Proceedings of the 14th international conference on World Wide Web*, ACM. ACM, 2005, pp. 530 – 538.

[11] M. Klusch, B. Fries, M. Khalid, and K. Sycara, "Owls-mx: Hybrid owl-s service matchmaking," in *Proceedings of 1st Intl. AAAI Fal l Symposium on Agents and the Semantic Web*, 2005.

[12] V. X. Tran and H. Tsuji, "Owl-t: A task ontology language for automatic service composition," in *Proceedings of the IEEE International Conference on Web Services, 2007.* IEEE Computer Society, July 2007, pp. 1164–1167.

[13] A. S. Rao and M. P. Georgeff, "Modeling rational agents within a bdi-architecture," in *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, J. Allen, R. Fikes, and E. Sandewall, Eds. Morgan Kaufmann Publishers Inc., 1991, pp. 473–484.

[14] F. Dignum, D. Morley, E. Sonnenberg, and L. Cavedon, "Towards socially sophisticated bdi agents." in *Proceedings of the Fourth International Conference on MultiAgent Systems*. IEEE Computer Society, 2000, p. 111.

[15] A. S. Rao and M. P. Georgeff, "Bdi agents: from theory to practice," in *Proceedings of the First International Conference on Multi-Agent Systems ICMAS 95*, 1995.

[16] F. Dignum, "Autonomous agents with norms," *Artificial Intelligence and Law*, vol. 7, no. 1, pp. 69–79, March 1999.

[17] J. Broersen, M. Dastani, J. Hulstijn, Z. Huang, and L. van der Torre, "The boid architecture: conflicts between beliefs, obligations, intentions and desires," in *Proceedings of the fifth international conference on Autonomous agents*, SIGART. ACM, 2001, pp. 9–16.

[18] F. L. y Lopez, M. Luck, and M. d'Inverno, "A framework for norm-based inter-agent dependence," in *Proceedings of 3rd Mexican International Conference on Computer Science*, September 2001, pp. 15–19.

[19] L. deSilva and L. Padgham, "Planning on demand in bdi systems," in *Proceeding of he International Conference on Automated Planning and Scheduling (ICAPS)*, June 2005.

[20] M. Kollingbaum and T. Norman, "Noa - a normative agent architecture," in *Eighteenth International Joint Conference on Artificial Intelligence IJCAI'03*, G. Gottlob and T. Walsh, Eds. Morgan Kaufmann Publishers Inc., 2003, pp. 1465–1466.

[21] G. Andrighetto, M. Campenni, R. Conte, and M. Paolucci, "On the immergence of norms: a normative agent architecture," in *Proceedings of AAAI Sympoisa Fall 2007*, AAAI. AAAI Press, 2007.

[22] M. Klusch, P. Kapahnke, and B. Fries, "Hybrid semantic web service retrieval: A case study with owls-mx," in *Proceedings of the Second IEEE International Conference on Semantic Computing*. IEEE Computer Society, August 2008, pp. 323–330.

[23] K. Sycara, M. Klusch, S. Widoff, and J. Lu, "Dynamic service matchmaking among agents in open information environments," *SIGMOD Rec.*, vol. 28, no. 1, pp. 47–53, 1999. [Online]. Available: http://portal.acm.org/citation.cfm?id=309895