Rules and Computation on the Semantic Web

Daniel Elenius, Mark-Oliver Stehr

SRI International, Menlo Park, California, USA
firstname.lastname@sri.com

Abstract. Diverse domains need an ability to express computations, and procedural attachments are not the final answer, because of problems regarding semantics, verifiability, extensibility, and evaluation. Rules do not have enough expressiveness, and a naive use of first-order logic suffers from a number of serious problems. The main problem stems from the difficulty of reconciling OWL's underlying *relational* knowledge representation with effective (and efficient) computation. We propose to use Hilbert's ϵ construct as an extension to SWRL, which solves or alleviates many of the problems we have run into. We hope that this paper will feed overdue discussions about computation on the Semantic Web, which goes beyond just Semantic Web Services.

1 INTRODUCTION

OWL has been used in a wide variety of fields beyond the original Semantic Web vision – e-Science, medicine, biology, geography, astronomy, defense, and the automotive and aerospace industries [1]. As its area of application has grown, so have demands of expressiveness beyond what OWL provides.

OWL Requirements and Use Cases [2] anticipated this development to some extent, and lists a number of optional features, called "objectives," many of which users are now starting to ask for. One of these features is "Procedural Attachments" (PAs):

> The language should support the use of executable code to evaluate complex criteria. Procedural attachments greatly extend the expressivity of the language, but are not well-suited to formal semantics. A procedural attachment mechanism for web ontologies should specify how to locate and execute the procedure. One potential candidate language would be Java, which is already well-suited to intra-platform sharing on the Web.

(The motivation for this objective is "Ontology interoperability". It is not clear what the authors had in mind here.) While we agree with the first sentence above, which expresses a *need*, we do not necessarily agree with the *solution* expressed in the rest of the paragraph.

Of course, many reasoners support procedural attachments of one form or another (e.g., SNARK[1]). The SWRL specification [3] introduces a number of "built-ins", such as arithmetic and string operations. These could be considered as PAs, but there is no way for users to add more PAs. Situated Logic Programs [4] introduces a "principled" way of adding "sensor" and "effector" actions as PAs in Java to Logic Programs.

[1] http://www.ai.sri.com/snark/

Unfortunately, procedural attachments are problematic for several reasons.

- If a language does not have a clear operational semantics, it is not clear under what circumstances the PA will be executed. For example, in a language like Prolog, one knows exactly how the code will execute, and there is an "obvious" place for the PA to be called. In OWL, this is not the case.
- PAs do not support *partial evaluation*. Thus, with a PA for addition, one could submit $1+1$ and get back 2, but given an equation $1+X=2$, the PA would not be able to figure out $X=1$ (or even return the function $+1$, as would be the case in functional programming).
- PAs are not formally defined. Thus, they have no semantics, and an ontology that uses the PA therefore has no semantics either. This is especially troublesome in situations where there is a need to formally verify or analyze an ontology. One example of this is reasoning about certain kinds of *policies* (see Section 3.2).
- If a fixed set of PAs is used (as in SWRL), it may be possible to formally define their meaning once and for all *outside* of the ontology language.[2] The problem with this approach is that no one can define additional PAs and expect to have reasoning support for them (unless they are added to the next revision of the language).

The alternative to PAs is to build the requisite expressiveness into the ontology language. Of course, adding the expressiveness to perform arbitrary computations in a language makes it very expressive and computationally complex, but in many situations there may be no choice.

The common way to formally define computations is by using functions, defined using (possibly recursive) equations. This approach is used in modern functional programming languages, such as Haskell and ML, as well as in, for example, equational logic [5] and first-order logic with equality. Programs in these languages are completely declarative, and can thus be given a very simple and straightforward semantics.

But OWL does not have functions. The closest thing is functional properties, which one can write Rules about, using extensions such as SWRL. But we will show, in Section 2, that Rules are insufficient for this task, because they cannot be used to return structured data. In Section 3, we outline two use cases that motivated the work in this paper. In Section 4, we propose an extension to Semantic Web rules languages, which alleviates the problems of using Rules to express computation, viz. Hilbert's ϵ-operator. Section 5 gives the syntax and semantics for our extension, as an extension to the syntax and semantics of SWRL. Section 6 discusses this project in a wider perspective of finding a logical and semantic framework for the Semantic Web, and relates our ideas to other work that has been done in the field. We end the paper with some concluding remarks, in Section 7.

[2] This is not done with the SWRL built-ins – they are defined neither in the language or outside of it.

2 GENERATING STRUCTURE

We will consider as an example the function `append`, which takes two lists and returns the concatenation of the two lists. This is a very simple example, but it is sufficient to bring out several points.[3]

In Prolog, we might write

```
append([H|T], B, [H|TB]) :- append(T, B, TB).
append([], B, B).
```

This procedure returns the result of appending the first two arguments as the third argument. So can we use "rules" after all for this kind of computation? No, because we have cheated. The version above uses a *functional* representation of lists. This is easy to see if we replace Prolog's built-in list operations with a list constructor, `c`, and an empty list element `nil`.

```
append(c(H,T), B, c(H,TB)) :- append(T, B, TB).
append(nil, B, B).
```

Here, `c` and `nil` are *constructors*. This means that these symbols, or applications of these symbols, are not further evaluated. They are structure-building operators. As an example, a list consisting of the three elements `1,2,3` is represented as `c(1,c(2,c(3,nil)))`. Nested terms with constructors, then, is how data structure is represented in a functional language.

Rule languages like SWRL that have been proposed as extensions to OWL are Datalog, that is, function-free, so they cannot express the Prolog encodings above.[4] However, this is somewhat beside the point – lists (or any other data structure) in OWL are represented *relationally*, not *functionally*. So, consider the representation of lists where we have a class `List`, with an individual `nil`, an object property `first` with domain `List`, and an object property `next` with domain and range `List`.[5] In this representation, the list [1,2,3] is represented by the following facts: `first(L,1), next(L,L2), first(L2,2), next(L2,L3), first(L3,3), next(L3,nil)`. Can we now define the `append` function, using rules, to work on this list representation? The answer is No, and the reason is that we are not allowed to use existentials in a rule head.[6] To see what kind of

[3] We realize that SWRL has a built-in `swrlb:listConcat` operation, but we could easily come up with different examples that cannot be handled by any SWRL built-in. Furthermore, the treatment here can be seen in part as the project of giving a semantics to the SWRL built-ins.

[4] SWRL is also limited to predicates of arity one and two, but more on this later.

[5] This is almost exactly like `rdf:List`, but note that `rdf:List` cannot be used in OWL DL, because each property must be either an object property or a datatype property. This means that one cannot have lists of mixed literals and individuals.

[6] This is a truth with modification, as SWRL strictly speaking allows `someValuesFrom` restrictions in the rule head, which imply an existential quantifier. But the general form used here is still not possible.

expression we need, we can define the `append` relation in first-order logic as

$$\begin{array}{l}\forall ht, h, t, b, tb \\ \quad \exists htb\ \texttt{append}(ht, b, htb) \wedge \texttt{first}(htb, h) \wedge \texttt{next}(htb, tb) \leftarrow \\ \qquad \texttt{first}(ht, h) \wedge \texttt{next}(ht, t) \wedge \texttt{append}(t, b, tb) \\ \forall b\ \texttt{append}(\texttt{nil}, b, b)\end{array} \tag{1}$$

The main observation here is: In a functional representation, we can give the structure of the return value as a term in the rule head. In a relational representation, structure is not represented using terms, but by using formulas. This causes a number of problems, which we outline below. It should be noted that not *all* functions represented as relations suffer from these problems. The problems occur only when we need to return structured data that was not part of the input. Also, one can often define many functions in terms of other functions. For example, once we have `append`, we can define many other list operations, such as `reverse`, in terms of `append`. But this does not really matter – the new relational structure has to be built *somewhere*.

The problems with the relational representation of structured data are

1. *Expressiveness.* To represent structured return values relationally, we need more than one formula, and at least one existential quantifier. This takes us beyond Horn logic, and leads to a more complex reasoning problem.
2. *Awkwardness of representation.* A relational representation results in a more verbose and awkward form than a functional representation, which makes it easier for humans to make mistakes in writing these functions.
3. *Extensionality.* When data structures are represented relationally, they are not *extensional*. For example, two `List`s with the same elements are not necessarily the same `List`. This means that we cannot reason about equality in an adequate way.
4. *Unnecessary nondeterminism.* If the relation is used twice with the same input arguments, the result may not be the same `List` individual both times. All we can say is that the FOL pseudo-function above returns *some* list with certain properties.
5. *Rule form.* Normally, rules are used in a goal-oriented way. The reasoner will know that it needs to check if some `append` atom is true, so it executes rules with `append` in the rule head. This is why rules normally have just one atom in the rule head, but here we have several. A reasoner attempting to use the rule above would have to "know" that `append` is the "interesting" predicate in the rule head, and the second and third atoms are just "side effects" (it would, of course, also have to keep track of such side effects).
6. *Unintended structures.* There is no syntactic way to exclude unintended structures such as cyclic lists. Relational structures are *graphs*, whereas terms are *trees*.

As we mentioned in the Introduction, we have found a way to eliminate or alleviate these problems. But before we describe our solution, we will look at two use cases that motivated this work.

3 USE CASES

The conclusions drawn in Section 2 come from two sources, and our original responses to these negative results had two different forms. These two use cases also provided the motivation to overcoming these problems. In Section 4 we present an innovative solution, and we plan to revisit our old solutions based on our new insights. But first let us see how the limitations of Rules showed up for us in the past.

3.1 Interoperability Analyzer

In the world of military training systems, and especially for larger training exercises, there is often a need to connect different training systems. The problem is that different training systems are usually not built to be used together.

Many interoperability solutions have been implemented or proposed for training and similar systems [6, 7]. However, none of these solutions encompass the whole range of interoperability problems. In particular, they usually only specify standards for networking and syntax, but fall short of a comprehensive semantics, and fail to address behavioral problems [8].

In [9], we describe the application of Semantic Web technologies to enable automated "purpose-aware" reasoning about this kind of interoperability, as part of the Open Netcentric Interoperability Standards for Training and Testing (ONISTT) program.[7]

ONISTT is developing (a) ontologies to express the capabilities needed to perform mission-related tasks, (b) ontologies to express the capabilities available from prospective resources for executing those tasks, and (c) an automated reasoner/analyzer that can determine if the collective capabilities of some subset of candidate resources can satisfy the needs of a specific target mission. Because "goodness" of fit among resources is not always a simple "yes" or "no", the analyzer may give a qualified answer, leaving it to human judgment to say whether the level of interoperability is "good enough" or "the best obtainable". The analyzer produces this output by applying *rules* to the data under analysis.

Examples of problems the analyzer can find are lack of (or incompatible) training system or tactical communications (where needed), or uncorrelated terrain data, which can cause problems like tanks hovering in the air, or airplanes flying through mountains, as seen from some participant's computer-generated view.

The analyzer is conceptually a *function*, which takes as inputs the capability and resource knowledge bases (KBs), and returns the result of applying these rules on the KBs. The result is a structure containing (1) the successes, failures, errors, and warnings, and the conditions under which they can occur, and (2) some "configuration artifacts" that describe *how* to connect the participant resources together in order to perform the training exercise, for example, descriptions of where and how to use mediation components.

[7] Supported by the Office of the Under Secretary of Defense for Personnel and Readiness

The set of rules to check for problematic conditions is meant to be extensible. Ideally, one should be able to specify these rules in a declarative way, for example, in SWRL, so that they can easily be inspected, edited, and so on. Ideally, these results would be encoded as ontological objects (e.g., a Warning class, with subclasses for different types of warnings), and we would then like the analyzer to create and return the appropriate instances, with the appropriate property values (or value chains).

However, for the reasons explained in Section 2, SWRL rules cannot create new instances, or indeed return any type of structured data, unless the data is part of the "input" arguments to the rule. We *can* do this in Prolog, by using compound terms, but SWRL is Datalog (i.e., function-free), and thus it does not have compound terms.

Our previous solution was to use Prolog. We translated the DLP fragment [10] of the OWL knowledge bases into Prolog, and hand-coded the analyzer rules as (rather procedural) Prolog rules. While this approach *works*, it is not as declarative and extensible as we would like.

3.2 Policy Logic

With increasing demands on wireless communications, radio spectrum is becoming a hot commodity. Deploying radios to different regions or countries means encountering a new spectrum environment, with new policies to abide by. Delays occur because policies are currently hard-coded into the radios' software or firmware, and time-consuming upgrades must be performed.

The solution to these problems offered by DARPA's NeXt Generation (XG) Communications Program is based on two key observations: a) in a typical situation, most of the spectrum, although allocated, is not in use; and b) radios could be made more flexible by adopting a scheme of declarative spectrum *policies*, offering the usual advantages of a policy-based approach, such as easier deployment, verification, management, and so on.

Under the new approach to spectrum management, highly capable sensors are used at runtime to scan allocated frequency bands in order to detect parts of the spectrum that are not currently in use. Policies are used to describe the constraints on using the spectrum. For example, how long must a band be empty before it can be used, what level of detected spectral density is considered background noise, and at what power level are you allowed to transmit? These policies vary depending on geographic region and time, and thus radios must be able to load new policies at runtime.

The overall picture is that each XG radio is equipped with a Policy Reasoner (PR). The radio provides the PR with facts about itself and the (sensed) environment, and the reasoner tells the radio whether or not it can transmit.

SRI was tasked with developing a policy language and reasoning techniques for the XG program. Our initial solution is outlined in [11]. Among the requirements on the policy language and reasoner were the following.

Accreditability. It should be possible to accredit the policy reasoner, individual policies, and radio software, all independently of each other. This reduces

the combinatorial nightmare of having to accredit each combination of radio and policies, as is done today. Furthermore, it removes the need to re-accredit the radio and its software every time a new policy is introduced, which currently makes the goal of rapid deployment impossible.

Extensibility. It should be possible to easily add new domain knowledge and policies, as well as to extend the expressiveness of the language if needed for new domains.

Expressiveness. We examined current spectrum policies, as well as imagined future ones, to determine what kind of constraints we need to be able to express in the language. In particular, we need the following features:

- *Functions*, such as "power masks"
- *Computations*, such as temporal and geospatial reasoning
- *Orderings* (e.g., frequency less than 5000.0)

There was also a desire to build on OWL, because of its status as a widely used standard. However, because of the problems described in Section 2, we found this requirement to be incompatible with the other requirements. We did not have the option of using a general procedural attachment mechanism for the computational needs mentioned above, because of the strict need of accreditability. Formalizing the procedural attachments in an external language was also not a solution because of the need for extensibility. Something had to give, so we decided to build our policy language (called CoRaL) on a foundation of functional representation instead of using OWL (and SWRL). This sets it apart from other policy languages, such as KaOS, Rei, or Ponder [12].

4 SOLUTION

The main innovation to make structure-generating computation in OWL work is the ϵ operator.[8] ϵ is a term constructor. $\epsilon x\ \phi(x)$ denotes *some* x such that $\phi(x)$ is true. Put differently, $\epsilon x\ \phi(x)$ is a *witness* to the formula $\exists x : \phi(x)$.

This lets us "push existentials into terms" so that we can define some (anonymous) individual with certain properties as the return value of a function. The `append` relation from Section 2 can now be expressed as

$$\begin{array}{l} \forall ht, h, t, b, tb \\ \quad \texttt{append}(ht, b, \epsilon\, htb\, \texttt{first}(htb, h) \wedge \texttt{next}(htb, tb)) \leftarrow \\ \qquad \texttt{first}(ht, h) \wedge \texttt{next}(ht, t) \wedge \texttt{append}(t, b, tb) \\ \forall b\ \texttt{append}(\texttt{nil}, b, b) \end{array} \tag{2}$$

Let us look at the advantages of this representation compared to (1), above.

1. *Expressiveness.* Using ϵ, our rules are once again Horn clauses, but this simplicity may be deceptive – we hypothesize that the ϵ may be almost as difficult to reason with as generic existential quantifiers.

[8] See `http://plato.stanford.edu/entries/epsilon-calculus/` for some background.

2. *Awkwardness of representation.* We would argue that the new form is less awkward to write and to understand (once we have wrapped our heads around the ϵ-construct). It becomes obvious what parts of the formula are merely "side conditions".
3. *Extensionality.* Relationally encoded structures are still not extensional in general. For example, given some individuals a, b, and c, and the facts $\texttt{first}(a, c)$, $\texttt{next}(a, nil)$, $\texttt{first}(b, c)$, $\texttt{next}(b, nil)$, we cannot infer $a = b$. However, the list $\epsilon\, x\, \texttt{first}(x, c) \wedge \texttt{next}(x, nil)$ and $\epsilon\, y \texttt{first}(y, c) \wedge \texttt{next}(y, nil)$ denote the same element (the two terms are "equal modulo alpha-conversion").
4. *Unnecessary nondeterminism.* The elimination of the existential quantifiers means that (2) is deterministic. This may seem strange since the result is still essentially existentially quantified, but things will become clearer when we explain the precise semantics of the ϵ construct, in Section 5. Briefly, the denoted element is arbitrary but fixed by the interpretation, i.e., fixed *once and for all.*
5. *Rule form.* The rule is now in the form of a standard Horn clause, i.e. it has only one atom in the rule head.
6. *Unintended structures.* One can still build cyclical structures, e.g. $\epsilon\, x\, \texttt{first}(x, a) \wedge \texttt{next}(x, x)$.

So, while we have not eliminated all the problems, a few of them are gone, and a few are alleviated.

4.1 N-ary Relations, or Multiple Arguments

OWL is limited to predicates of arity one and two (i.e., classes and properties, respectively), and one argument position is used up for the return value, so it looks like we can not define functions of more than one argument. The solution is well known: We use an individual as the argument, and give it properties pointing to the "real" arguments. For example, to return to the `append` example, we would use an individual with propertes such as `list1` and `list2`, and the values for the two properties would be the two list individuals to append.

$$\begin{aligned}
&\forall args, ht, h, t, b, tb \\
&\quad \texttt{append}(args, \epsilon\, htb\, first(htb, h) \wedge \texttt{next}(htb, tb)) \leftarrow \\
&\qquad \texttt{list1}(args, ht) \wedge \texttt{list2}(args, b) \wedge \texttt{first}(ht, h) \wedge \texttt{next}(ht, t) \wedge \\
&\qquad \texttt{append}(t, b, tb) \\
&\forall b\ \texttt{append}(\texttt{nil}, b, b)
\end{aligned} \tag{3}$$

Here, the ϵ brings us another advantage – when we want to invoke our new version of `append`, we can do it as one formula

$$\texttt{append}(\epsilon\, x\, \texttt{list1}(x, l1) \wedge \texttt{list2}(x, l2), res)$$

as opposed to the non-ϵ version

$$\texttt{list1}(args, l1) \wedge \texttt{list2}(args, l2) \wedge \texttt{append}(args, res)$$

A couple of issues are worth pointing out regarding the handling of arguments.

– In OWL, we cannot define functions that take data values as arguments. The domain of datatype properties cannot be, e.g., `xsd:int`. The solution is to use the multiple-arguments method with one or more data-valued arguments. The argument container is always an individual.
– The SWRL built-ins use a different method of encoding multiple arguments – `rdf:List` is used. We would argue that this method is more awkward, because more individuals that are merely "argument containers" are needed. For example, for three arguments, one needs three list elements. The advantage of using lists is that one can support an arbitrary number of arguments, as SWRL does with many of its operations (e.g. addition, multiplication). Another idiosyncracy of the SWRL built-ins is that many use the first argument as the return value. It should be noted that our method and the SWRL method are not mutually exclusive, and ϵ also helps if the SWRL method is used.

5 SYNTAX AND SEMANTICS

To define the syntax and semantics of our extension, we start with SWRL, adding a syntactic element, and extending the semantics of SWRL to cover the new element. We use the Abstract Syntax of OWL/SWRL. Extending the XML Presentation Syntax could be done analogously, but showing this would take up too much space here. We do *not* define an RDF/XML syntax for the new element, for the same reason that SWRL FOL [13] does not – RDF cannot accurately capture even SWRL, so we prefer not to add to that particular confusion. (It should be noted that one can still refer to other ontologies in RDF/XML format from ontologies in the XML Presentation Syntax.)

5.1 Abstract Syntax

We have said that ϵ-constructs are *terms*, and SWRL already has embryonic terms in the form of "i-objects" and "d-objects". An i-object can be either an individual or an individual-valued variable. A d-object can be either a data literal or a data-valued variable. We extend the SWRL abstract syntax productions for `i-object` and `d-object` so that both can also be ϵ-terms (see Figure 1). Note that the ϵ-construct can be nested. Normally, the formula inside the ϵ-construct will be limited to a set of SWRL atoms. As usual, this is interpreted as a conjunction of atoms. In effect, ϵs thus give us the existential-conjunctive fragment, which is enough to specify any RDF graph. If needed, one can extend the ϵ to also allow SWRL FOL `foformula`s under the ϵ, that is, to allow any first-order formula to appear in this position. We provide for both options in this syntax and semantics.

5.2 Semantics

Previously, we have defined the semantics of our Framework through a translation to our Universal Policy Logic [14–16]. However, there is also a need for a more direct, "native," semantic treatment. We therefore present the semantic extensions as extensions to the semantics of SWRL.

Fig. 1. SWRL abstract syntax augmented with the ϵ-construct.

```
i-object ::= i-variable | individualID |
             'eps(' i-variable ( { atom } | foformula ) ')'
d-object ::= d-variable | dataLiteral |
             'eps(' d-variable ( { atom } | foformula) ')'
```

Recall from the OWL semantics [17] that, given a datatype map D, an abstract OWL interpretation is a tuple of the form $I = < R, EC, ER, L, S, LV >$ where R is a set of resources, $LV \subseteq R$ is a set of literal values, and EC is a mapping from classes and datatypes to subsets of R and LV, respectively, ER is a mapping from properties to binary relations on R, L is a mapping from typed literals to elements of LV, and S is a mapping from individual names to elements of $EC(\texttt{owl:Thing})$. We will use O for $EC(\texttt{owl:Thing})$ in the following.

SWRL extends this semantics in the following way: Given an abstract OWL interpretation I, a binding $B(I)$ is an abstract OWL interpretation that extends I such that S maps i-variables to elements of O, and L maps d-variables to elements of LV. Conditions are given for when a formula f is satisfied by a binding, written $B(I) \Rightarrow f$.

We extend the SWRL semantics to support the ϵ-construct, in the following way. We add three more elements to I: The well-orderings $\prec_O$ on O and $\prec_{LV}$ on LV, and a distinguished *error element* $\mathfrak{err}$.

To facilitate our presentation, we introduce the notation $[x := y]B$ for the binding that maps x to y, and is otherwise the same as B.

We extend S to map individual-valued ϵs to elements of $O \cup \{\mathfrak{err}\}$, and L to map data-valued ϵs to elements of $LV \cup \{\mathfrak{err}\}$, as follows

$S(\texttt{eps(I-variable(}V\texttt{)}\ A_1 \ldots A_n\texttt{)})$ = The unique $o \in O$ such that for all $i \in 1 \ldots n,\ B'(I) \Rightarrow A_i$, where $B' = [V := o]B$ and o is minimal w.r.t. $\prec_O$, or $\mathfrak{err}$ if such an o does not exist.

$S(\texttt{eps(D-variable(}V\texttt{)}\ A_1 \ldots A_n\texttt{)})$ = The unique $l \in LV$ such that for all $i \in 1 \ldots n,\ B'(I) \Rightarrow A_i$, where $B' = [V := l]B$ and l is minimal w.r.t. $\prec_{LV}$, or $\mathfrak{err}$ if such an l does not exist.

For SWRL FOL, replace the atom list $A_1, \ldots, A_n$ with a single formula f.

All formulas containing $\mathfrak{err}$ are unsatisfied. Note that these are recursive definitions, because the formula f can itself contain ϵ-constructs. Also note that these new terms cannot be used in regular OWL statements, only in SWRL statements, where i-objects and d-objects are allowed. This is intentional, and keeps things relatively simple.

The conditions for an interpretation to satisfy an ontology is as defined in the SWRL and SWRL FOL semantics.

6 DISCUSSION

Our work could be seen as a step toward the long-standing goal in computer science to bring computation and logic together in some uniform way. In the

case of OWL and SWRL, the main problem is the inability to express structure-generating computation, and logically this was solved through the use of the ϵ-operator. However, there remain problems of extensionality, unintended structures, and, most important, reasoning complexity. To represent RDF graphs, we need existential variables in the rule heads. We are extending SWRL, which is already an undecidable language. However, for SWRL by itself, there is some hope of finding decidable fragments, for example, using "DL-safe" rules [18]. Existentials in the rule head, on the other hand, is a decidedly "unsafe" thing to do, and it is unlikely that we will find a "direct" approach to reasoning with this extension. There are ways ahead, however.

We could build our Semantic Web language stack on a new foundation, using functional instead of relational representation. This has the obvious disadvantage of not building on all the work that has been done on the current set of languages (RDF, OWL, SWRL, etc.).

On the other hand, there are already two very serious issues in the current Semantic Web language stack. First, there is ample evidence that RDF is not the right starting point. RDF is too weak to encode first-order logic without paradox [19], it cannot handle rules properly [3, 13], it cannot handle nonsemantic entities like comments [1], and it is problematic even in its use to encode plain OWL [20]. The second issue is the combination of monotonic and nonmonotonic features, which has given rise to the discussion of "single stack" vs. "twin towers" approaches to language layering [21, 22].

One approach here is to use a very expressive language as a semantic framework, from which one can cut out appropriate sublanguages. First-order logic has been suggested for this role [23]. However, this primarily addresses the issue of model-theoretic semantics, not the operational semantics or computation model.

A more general solution, that we call the *logic translation* approach, is to keep all the existing languages, plus extensions such as the ϵ-operator, and map them to a different formalism, which is more suitable for automated reasoning and computation. Several such formalisms exist:

- *Logic Programming and Prolog.* This approach did not achieve great success in this context – programming in Prolog requires a deep familiarity with its operational semantics and how to control backtracking for efficiency, using "impure" nonlogical features of the language. Prolog programs are thus far from the declarative specifications that this vision was aiming at.
- *Algebraic Specification and Equational Logic.* In this approach, a subset of first-order logic is used to specify data types, along with the operations one can perform on them [24, 5]. For example, a very direct mapping from $\mathcal{SHOIN}$ to the algebraic specification language CASL has been defined in [25].
- *Typed λ-Calculus and Higher-Order Logic.* This approach is based on higher-order functions, using β-reduction as its operational semantics [26]. This approach has given rise to practical programming languages such as Haskell

and ML. An interesting contribution was made by [27], which discusses the use of λ-calculus for Semantic Web Services.

- *Executable Logical Frameworks.* In this approach, one encodes the syntax (and sometimes model-theoretic and/or proof-theoretic semantics) of the language in another language that allows the specification to be executed, for example, Rewriting Logic [28].

An extension of the logic translation approach, which is worth investigating in the context of Semantic Web languages, has been used in [29] to map a classical logic into a computational type theory. From the viewpoint of general logics [30], the mapping consists of two stages: a logic translation and a *theory interpretation* [31]. The logic translation allows us to move across logics like above, but the theory interpretation is essential, because it allows us to move from an abstract axiomatic to a more concrete computational representation. The combination of logic translation and theory interpretation is more powerful than a logic translation alone, because it effectively allows us to make interpretation choices and restrict the class of models to those choices that are computationally meaningful.

A separate strand of research has been Semantic Web Services (SWSs)[32]. Here, the idea is usually that computations on the web are characterized by their inputs, outputs, preconditions, and effects. But the computations themselves are left to be defined in some external way. The focus is on discovering and composing these services.

7 CONCLUSIONS

Diverse domains need an ability to express computations, and so-called procedural attachments are not the final answer, because of problems regarding semantics, verifiability, extensibility, and evaluation. We have also shown that "rules" (such as SWRL rules) do not have enough expressiveness and that a naive use of first-order logic for computation over relational representations suffers from a number of serious problems.

Our work is motivated by two real-world use cases where these issues appeared, and we were forced to either let go of declarativity and formal semantics, or to drop OWL and SWRL in favor of a different language.

We have presented a partial solution to these problems, viz. the extension of SWRL to include the ϵ-construct, which lets us "push existentials into formulas" and thereby solve or alleviate many of the problems we ran into. Reasoning and computation techniques for this extension still have to be investigated.

We have shown how our goals could be viewed in the context of the long-standing project of computer science to bring logic and computation together in a uniform way. We hope that this paper will feed overdue discussions about computation on the Semantic Web, beyond just Semantic Web Services.

ACKNOWLEDGMENTS

We thank David Martin for useful feedback. This research was supported by DARPA's neXt Generation (XG) Communications Program under Contract Number FA8750-05-C-0230.

REFERENCES

1. Cuenca Grau, B., Horrocks, I., Parsia, B., Patel-Schneider, P., Sattler, U.: Next steps for OWL. In: Proc. of the Second OWL Experiences and Directions Workshop. Volume 216 of CEUR (`http://ceur-ws.org/`). (2006)
2. Jeff Heflin (ed.): OWL Web Ontology Language Use Cases and Requirements (2004)
3. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web Rule Language Combining OWL and RuleML (W3C Member Submission (2004)
4. Grosof, B.N.: Representing E-Commerce Rules Via Situated Courteous Logic Programs in RuleML. Electronic Commerce Research and Applications (ECRA) **3** (2004) 2–20
5. Dershowitz, N., Plaisted, D.A.: Equational programming. In Hayes, J.E., Michie, D., Richards, J., eds.: Machine Intelligence 11: The logic and acquisition of knowledge. Oxford Press, Oxford (1988) 21–56
6. Distributed Interactive Simulation Committee of the IEEE Computer Society: IEEE standard for distributed interactive simulation – application protocols (1998) IEEE Std 1278.1a-1998.
7. Simulation Interoperability Standards Committee of the IEEE Computer Society: IEEE standard for modeling and simulation (M&S) high level architecture (HLA)-framework and rules (2000) IEEE Std 1516-2000.
8. Tolk, A., Turnitsa, C.D., Diallo, S.Y.: Composable M&S web services for net-centric applications. Journal for Defense Modeling and Simulation (JDMS) (2006)
9. Elenius, D., Ford, R., Denker, G., Martin, D., Johnson, M.: Purpose-aware reasoning about interoperability of heterogeneous training systems (submitted to ISWC 2007) (2007)
10. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description logic programs: Combining logic programs with description logic. In: Proc. of the Twelfth International World Wide Web Conference (WWW 2003), ACM (2003) 48–57
11. Elenius, D., Denker, G., Stehr, M.O., Senanayake, R., Talcott, C., Wilkins, D.: CoRaL - Policy Language and Reasoning Techniques for Spectrum Policies (accepted at policy 2007) (2007)
12. Tonti, G., Bradshaw, J., Jeffers, R., Montanari, R., Suri, N., Uszok, A.: Semantic web languages for policy representation and reasoning: A comparison of KAoS, Rei, and Ponder. In: 2nd International Semantic Web Conference (ISWC2003), Springer-Verlag (2003)
13. Patel-Schneider, P.F.: A Proposal for a SWRL Extension towards First-Order Logic (W3C Member Submission) (2005)
14. Stehr, M.O.: Towards a Universal Policy Logic. Technical report, SRI International (2007)
15. Elenius, D.: Translating OWL DL to the Universal Policy Logic. Technical report, SRI International (2007)
16. Elenius, D.: OWL Extensions. Technical report, SRI International (2007)
17. Patel-Schneider, P.F., Hayes, P., Horrocks, I.: OWL web ontology language semantics and abstract syntax (W3C recommendation) (2004)

18. Motik, B., Sattler, U., Studer, R.: Query Answering for OWL-DL with Rules. In: Proc. of 3rd International Semantic Web conf. (ISWC 2004). Volume 3298., Springer-Verlag (2004) 549–563
19. Patel-Schneider, P.F.: Building the Semantic Web Tower from RDF Straw. In: Proc. of the Nineteenth International Joint Conference on Artificial Intelligence. (2005)
20. Horrocks, I., Patel-Schneider, P., van Harmelen, F.: From SHIQ and RDF to OWL: The making of a web ontology language. Journal of Web Semantics **1** (2003) 7–26
21. Horrocks, I., Parsia, B., Patel-Schneider, P., Hendler, J.: Semantic web architecture: Stack or two towers? In Fages, F., Soliman, S., eds.: Principles and Practice of Semantic Web Reasoning (PPSWR 2005). Number 3703 in LNCS, Springer (2005) 37–41
22. Kifer, M., Bruijn, J.d., Boley, H., Fensel, D.: A realistic architecture for the semantic web. In: Proceedings of the International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML-2005). Number 3791 in Lecture Notes in Computer Science, Ireland, Galway, Springer (2005) 17–29
23. Guha, R.V., Hayes, P.: LBase: Semantics for Languages of the Semantic Web (W3C Note) (2003)
24. Wirsing, M.: Algebraic specification. In van Leeuwen, J., ed.: Handbook of Theoretical Computer Science. Volume B. North-Holland (1990) 675–788
25. Lüttich, K., Mossakowski, T., Krieg-Brückner, B.: Ontologies for the semantic web in Casl. In Fiadeiro, J.L., Mosses, P.D., Orejas, F., eds.: WADT. Volume 3423 of Lecture Notes in Computer Science., Springer (2004) 106–125
26. Barendregt, H.P.: Lambda-calculi with types. In Abramsky, S., Gabbay, D.M., Maibaum, T.S.E., eds.: Background: Computational Structures. Volume 2 of Handbook of Logic in Computer Science. Claredon Press, Oxford (1992)
27. Halpin, H., Thompson, H.S.: One document to bind them: Combining XML, web services, and the semantic web. In: Proc. of the World Wide Web Conference (WWW 2006). (2006)
28. Martí-Oliet, N., Meseguer, J.: Rewriting logic as a logical and semantic framework. In Meseguer, J., ed.: Rewriting Logic and Its Applications, First International Workshop, Asilomar Conference Center, Pacific Grove, CA, September 3-6, 1996, Elsevier Science B.V., Electronic Notes in Theoretical Computer Science, Volume 4, `http://www.elsevier.nl/locate/entcs/volume4.html` (1996) 189–225
29. Felty, A., Howe, D.J.: Hybrid interactive theorem proving using Nuprl and HOL. In McCune, W., ed.: Automated Deduction – CADE-14, 14th International Conference on Automated Deduction, Townsville, North Queensland, Australia, July 13–17, 1997, Proceedings. Volume 1249 of Lecture Notes in Artificial Intelligence., Springer-Verlag (1997) 351–365
30. Meseguer, J.: General logics. In Ebbinghaus, H.D., et al., eds.: Logic Colloquium'87, Granada, Spain, July 1987, Proceedings. North-Holland (1989) 275–329
31. Naumov, P., Stehr, M.O., Meseguer, J.: The HOL/NuPRL proof translator — A practical approach to formal interoperability. In: Theorem Proving in Higher Order Logics, 14th International Conference, TPHOLs'2001, Edinburgh, Scotland, UK, September 3–6, 2001, Proceedings. Volume 2152 of Lecture Notes in Computer Science., Springer-Verlag (2001) 329 – 345
32. McIlraith, S.A., Son, T.C., Zeng, H.: Semantic web services. IEEE Intelligent Systems **2** (2001) 46–53