# Integrating Positional and Slotted Knowledge on the Semantic Web*

Harold Boley

Institute for Information Technology – e-Business,
National Research Council of Canada,
Fredericton, NB, E3B 9W4, Canada
Harold.Boley@nrc-cnrc.gc.ca

**Abstract.** There have traditionally been two paradigms in knowledge representation, one using positional information, the other using unordered slots or attribute-value pairs. Both resurfaced in the Web through XML's positional child elements and RDF's slot-like properties. In the Semantic Web, positional Horn logic as well as slotted description logic and F-Logic have been employed. The positional-slotted Semantic Web language POSL is proposed as part of SWRL to reconcile sorted Horn logic's positional and F-Logic's slotted representations, accessing description logic's classes as variable types. Its syntax integrates Prolog's positional and F-Logic's slotted syntaxes for representing facts and rules on the Web, referring to RDFS or OWL classes for order-sorted typing. The presentation, shorthand, and exchange syntax of POSL is interconvertible with the XML markup of Object-Oriented RuleML. The POSL semantics enhances Herbrand models by accomodating slotted clause instantiation and ground equality, further restricted through types. Analogous enhancements apply to unification in the proof theory and (Java-based) OO jDREW implementation. Webizing uses URIs in the IETF form of N3 for individuals, relations, slots, and types. Webized atoms further permit RDF descriptions as slotted facts processed by RDF/POSL rules. RDF blank nodes become clause-global, module-local existential variables. The POSL design incorporates these notions in an orthogonal manner, which has allowed their independent revision. Two recent applications, RACOFI Music and the New Brunswick Business Knowledge Base, were enabled by POSL technology.

## 1 Introduction

This paper discusses the design, syntax, semantics, implementation, and use of integrated positional and slotted knowledge on the Semantic Web. It introduces the positional-slotted (POSL) language, which integrates Prolog's positional and F-Logic's slotted knowledge representation for the Semantic Web, permitting URIs in the style of Notation 3 (N3) for all language elements. POSL was also

* Thanks to Marcel Ball for the POSL converters, ANTLR grammar, and OO jDREW implementation. This research was funded by NRC as part of the Semantic Web Lab.

directly inspired by RDF [Hay04] and RuleML [Bol03], and is part of the SWRL effort [http://www.daml.org/rules/proposal].

Recent experience with the development of Semantic Web languages such as OWL [DS04] has shown the many advantages of studying expressive classes and formal semantics using an 'abstract' or 'human-oriented' syntax layer above the XML level. Also, as pioneered by Notation 3 (N3) [http://www.w3.org/2000/10/swap/Primer], a concise non-XML ASCII syntax is very useful in developing knowledge bases, which can then be parsed into some much more tedious XML markup such as RDF/XML for (distribution and) processing through the multitude of XML-aware tools. For new Semantic Web languages this reinforces what has been similarly known in the Lisp community for decades – that an XML markup and a concise syntax should be co-designed with a pair of converters permitting smooth transitions between the two. This paper is based on the pair (RuleML,POSL), whose evolving components have been supported by online converters [http://www.ruleml.org/submission/ruleml-shortation.html#section_6]. These being in place, we can look at the following design issues.

Knowledge representation (KR) languages have been developed, with limited time and interaction, to cover the following Semantic Web design space: Object-centered resource instance descriptions via binary properties (RDF), taxonomies over resource classes and properties (RDFS), description logic with class-forming operations and class/property axioms (OWL), as well as derivation, integrity, transformation, and reaction rules (RuleML). At the bottom of, or combined with, these languages, different kinds of (binary, triple, n-ary) ground facts have been used besides database tuples for representing instances. On top of, or again combined with, these languages, query languages have been defined. Integrations of various of these languages have been developed, including the combination of object-centered descriptions and rules (N3, OO RuleML) as well as description logic and rules (DLP, SWRL). These language integrations can help with information integration on the Web such as mapping object-centered representations to positional ones. The POSL research has explored this design space and attempted to introduce orthogonal ('decoupled') dimensions for systematic Semantic Web language development. The orthogonal design has allowed to incorporate most of the above notions in such a way that they can be used and revised independently from each other.

Two language families that predated the (Semantic) Web, yet have been very useful for it, are positional languages based on Horn logic such as Prolog and slotted languages with object-centered instance and class descriptions plus rules as in F-Logic [KL89]. Both have concise ASCII syntaxes, elegant semantics, and decent computational properties. Since these positional and slotted styles are often needed conjointly in the XML&RDF Web, they have been integrated in POSL. Prolog and F-Logic can be given additional XML syntaxes and adapted to the Semantic Web by 'webizing' key language elements via URIs as well as permitting modular and distributed knowledge bases. In POSL, the integrated

XML syntax is OO RuleML and integrated webizing is done in the style of N3 for all language elements.

Based on projects at NRC, we will exemplify POSL notions using an e-Business scenario. Without the POSL layer available above the XML level, it would not have been possible to complete, in the allotted time, the development cycles of recent Web applications such as RACOFI Music [ABB+03] [http://racofi.elg.ca/about.html] and the New Brunswick Business Knowledge Base [MB04] [http://www.ruleml.org/usecases/nbbizkb].

## 2 Facts, Queries, Complex Terms, and Rest Arguments

Positional, slotted, as well as a combined form of KR, each have advantages for various tasks in the Semantic Web, hence are supported in POSL in an integrated manner.

Beginning with our e-Business scenario, a 4-ary relation `shipment` can represent the shipping of some cargo at a price from a source to a destination. The corresponding `shipment` relationships (atoms) can be represented in all notations discussed.

**Positional notations** have been used, e.g., in everyday life, for ordered sequences of possibly repeating objects, in mathematics (hence in physics, chemistry, etc.), for n-tuples and for the arguments to n-ary functions, in logics, for the arguments to n-ary relations, in programming, database, and KR languages, for n-ary functions and relations, in XML, for child elements within a parent element, as well as in RDF, for `Seq`uence containers with ordered `rdf:li` children.

For example, the `shipment` relation can use the cargo, price, source, and destination directly as arguments, in that order. Corresponding `shipment` atoms can be represented as in relational database tuples, Datalog facts, etc.

For this, POSL uses a Prolog-like syntax, e.g. obtaining the following two ground facts (constants may be symbols, with a lower-case or upper-case first letter, or numbers):

```
shipment(PC,47.5,BostonMoS,LondonSciM).
shipment(PDA,9.5,LondonSciM,BostonMoS).
```

**Slotted notations** have been used, e.g., in everyday life, for unordered sets of attribute-value pairs, in mathematics, for labeled graphs, in frame logics, for molecular formulas, in programming, database, and KR languages, for records and object-centered relations, in XML, for attributes within start tags, as well as in RDF, for resource descriptions via properties.

For example, the above positional 4-ary `shipment` relation can also be conceived in a slotted manner, where *slot names* such as `cargo` identify the roles of the arguments so that their order becomes irrelevant. Corresponding `shipment` atoms can then be represented as object-oriented database nodes, frame logic facts, etc. pairing the slot names such as `cargo` with their *slot fillers* such as `PC`.

For this, POSL uses an F-Logic-inspired syntax, now obtaining these facts ("*name*`->`*filler*" slots are separated by a "`;`" infix, which indicates unorderedness):

```
shipment(cargo->PC;price->47.5;source->BostonMoS;dest->LondonSciM).
shipment(cargo->PDA;price->9.5;source->LondonSciM;dest->BostonMoS).
```

**Positional-slotted notations** have also been used combined, e.g., in Lisp, for obtaining the benefits of both KR methods.

For example, the above 4-ary `shipment` relations can also be split and recombined in a positional-slotted manner, where a positional part is followed by a slotted part. The first two arguments, `cargo` and `price`, are rather self-explaining in the positional notation, but the last two arguments, `source` and `dest`ination, could be confused, hence are given here explicit slot names in the combined positional-slotted notation.

For this, POSL uses a Prolog/F-Logic-combining syntax, obtaining these facts (the "`,`" infix has precedence over the "`;`" infix):

```
shipment(PC,47.5;source->BostonMoS;dest->LondonSciM).
shipment(PDA,9.5;source->LondonSciM;dest->BostonMoS).
```

Existing relations such as `shipment` may require later enhancement by further information such as `start` and `duration`. Instead of extending the argument sequences of all affected positional or positional-slotted atoms by further proper or null-value arguments in the ordered, positional manner, it is usually preferable to add only proper arguments in the unordered, slotted manner.

**Complex terms and plexes** go beyond the Datalog language considered so far. All three notations are also possible for any complex term (*cterm*), e.g. describing a pair of stakeholders as follows (using "`[...]`" for argument grouping): the positional `stakepair[PeterMiller,SpeedShip]`, the slotted `stakepair[owner->PeterMiller;shipper->SpeedShip]`, and the positional-slotted `stakepair[PeterMiller;shipper->SpeedShip]`. Similarly, a *plex* is regarded as the special case of a constructorless cterm, e.g. changing our stakeholder pairs thus: the Prolog-like list `[PeterMiller,SpeedShip]`, the F-Logic-inspired term `[owner->PeterMiller;shipper->SpeedShip]`, and the combination `[PeterMiller;shipper->SpeedShip]`.

**Non-ground formulas** contain at least one variable argument, interpreted as being universally quantified. They are allowed for all three notations. However, variables are not permitted as slot names since there would no longer be a unique most general unifier, so non-determinism would already arise during the unification phase of resolution. Variables can be named or anonymous. Named variables are prefixed by a "`?`"; the anonymous variable is written as a stand-alone "`?`". For example, facts and queries can be either ground or non-ground.

**Rest arguments** are permitted in atoms, one for positional arguments and one for slotted arguments. Positional arguments are separated from a positional rest by a "`|`"; slotted arguments are separated from a slotted rest by a "`!`". In both cases the rest itself is normally a variable. In particular, the anonymous variable can be used as a positional or slotted "don't care" rest. A slotted "don't care" rest makes an option out of F-Logic's convention: to tolerate arbitrary *excess slots* in either formula (e.g., a query), with slot names not used in slots of the other formula (e.g., a fact), during unification.

For example, for the earlier slotted `PC-shipment` fact, the query

```
shipment(cargo->?what;price->?;source->BostonMoS;dest->?goal)
```

succeeds, binding `?what` to `PC` and `?goal` to `LondonSciM`. However, the query

```
shipment(owner->?who;cargo->?;price->?;source->BostonMoS;dest->?)
```

fails because of its excess slot named `owner`. Similarly, the query

```
shipment(cargo->?what;source->BostonMoS;dest->?goal)
```

fails because of the fact's excess slot named `price`. On the other hand, the query with the slotted "rest doesn't care" combination "`!?`"

```
shipment(cargo->?what;source->BostonMoS;dest->?goal!?)
```

again succeeds with the initial bindings, since "`!?`" anonymously unifies the `price` slot (independent of where it occurs in the fact).

Conversely, the earlier fact would tolerate excess query slots such as in the above `owner` query after making it non-ground via an anonymous rest:

```
shipment(cargo->PC;price->47.5;source->BostonMoS;dest->LondonSciM!?).
```

If the query also contains an anonymous rest, both it and the fact can contain excess slots, as in

```
shipment(owner->?who;cargo->?what;source->BostonMoS;dest->?goal!?)
```

which succeeds with the initial bindings, since the anonymous query rest unifies the fact's `price` slot and the anonymous fact rest unifies the query's `owner` slot, leaving us agnostic about who the owner is.

If anonymous rest slots are employed in all formulas, the effect of F-Logic's implicit rest variables is obtained. The more precise "`!`"-free slotted formulas can enforce more restricted unifications where needed.

In general, "`|`" and "`!`" rests can follow after zero or more fixed positional and slotted arguments, and can unify the zero or more remaining arguments. Before being bound to a variable, such an arbitrarily long rest $e_1, \ldots, e_Z$ or $s_1 \rightarrow f_1; \ldots; s_Z \rightarrow f_Z$ is made into a single complex term, namely a plex `[`$e_1, \ldots,$`e`$_Z$`]` or `[`$s_1$`->`$f_1; \ldots;$`s`$_Z$`->`$f_Z$`]`, respectively.

Including both kinds of rest variables, the most general kind of positional-slotted atoms/cterms have these forms:

```
r(s1->f1;...;sL->fL;e1,...,eM|Ve;sL+1->fL+1;...;sN->fN!Vf)
c[s1->f1;...;sL->fL;e1,...,eM|Ve;sL+1->fL+1;...;sN->fN!Vf]
```

The **semantics** of POSL clause sets will be based on slotted (and positional-slotted) extensions of the positional (here, LP [Llo87]) notions of clause instantiation and ground equality (for the model-theoretic semantics) as well as unification (for the proof-theoretic semantics).

Since slot names are assumed here to be non-variable symbols, *slotted instantiation* can recursively walk through the fillers of slots, substituting dereferenced values from the substitution (environment) for any variables encountered.

Since POSL uses no implicit rest variables, *slotted ground equality* can recursively compare two ground atoms or cterms after lexicographic sorting – w.r.t. the slot names – of the slots encountered.

Since POSL uses at most (one positional and) one slotted rest variable on each level of an atom or complex term, *slotted unification* can perform sorting as in the above slotted ground equality, use the above slotted instantiation of variables, and otherwise proceed left-to-right as for positional unification, but pairing up identical slot names before recursively unifying their fillers, while collecting excess slots on each level in the plex value of the corresponding slotted rest variable.

## 3 Rules and Variable Typing

On top of the positional and slotted facts, and in the same integrated manner, POSL offers Horn-like rules for inferential tasks in the Semantic Web. Facts are interpretable as clauses that are degenerated (premiseless) rules, which in POSL can be naturally extended to clauses that are full-blown (premiseful) rules.

Extending the e-Business scenario, a ternary relation `reciship` can represent reciprocal shippings of unspecified cargos at a total cost between two sites. A Datalog rule infers this conclusion from three premises, two `shipment` atoms and an `add` atom. The `shipment` relation was defined in section 2 and the `add` relation is based on a SWRL built-in satisfied here iff the first argument is equal to the sum of the second and third arguments.

**Positional rules** are the usual Horn rules, in POSL written using a Prolog-like syntax, but with "`?`"-prefixed variables as in, e.g., Jess, N3, and SCL. In the `reciship` example, the following Datalog rule is obtained (the "`:-`" infix, for "⇐", has lowest precedence):

```
reciship(?cost,?A,?B) :-
  shipment(?,?cost1,?A,?B),
  shipment(?,?cost2,?B,?A),
  add(?cost,?cost1,?cost2).
```

The query `reciship(?total,BostonMoS,LondonSciM)` uses the rule to recursively query the corresponding `shipment` facts and the `add` built-in, binding `?total` to `57.0`. This query could be chained to from the body of another positional rule with head `reciBosLon(?total)`.

With types `?:Float` and `?:Address` for the head as well as additional `Product` and `Float` types for the extra anonymous and `?cost`-'indexed' body variables, the above `reciship` rule becomes fully typed as follows (we use the classical "`:`" infix between a variable and its type, which then also holds for all of its other occurrences in that clause):

```
reciship(?cost:Float,?A:Address,?B:Address) :-
  shipment(?:Product,?cost1:Float,?A,?B),
  shipment(?:Product,?cost2:Float,?B,?A),
  add(?cost,?cost1,?cost2).
```

**Slotted rules** are much like in F-Logic. The `reciship` relation is redefined here in a slotted manner with slot names `price`, `site1`, and `site2`, where two conventionally 'indexed' `site` slots are used. Analogously, the positional `add` relation is assumed here to have adjoined slot names `sum`, `addend1`, and `addend2`:

```
reciship(price->?cost;site1->?A;site2->?B) :-
  shipment(cargo->?;price->?cost1;source->?A;dest->?B),
  shipment(cargo->?;price->?cost2;source->?B;dest->?A),
  add(sum->?cost;addend1->?cost1;addend2->?cost2).
```

Notice that the slot name `price` now occurs both in the relation `shipment`, for an elementary cost, and in the relation `reciship`, for an aggregated cost. Similarly, while the `dest` slot in the `shipment` relation has type `?:Address`, a slot with the same name in a `flight` relation would have type `?:AirportCode`. Such 'overloading' is caused by slot names, except when 'webized' (cf. section 4), being local to their relations much like property restrictions are local to their class descriptions in OWL [DS04].

Now, `reciship(site1->BostonMoS;price->?total;site2->LondonSciM)` through `reciship(price->?total;site1->BostonMoS;site2->LondonSciM)`, the lexicographically sorted normal form, queries the slotted rule, which itself queries corresponding clauses, again binding `?total` to `57.0`. This query could be chained to from the body of another slotted rule with head `reciBosLon(price->?total)`.

This rule can again use variable typing:

```
reciship(price->?cost:Float;site1->?A:Address;site2->?B:Address)
  :-
  shipment
     (cargo->?:Product;price->?cost1:Float;source->?A;dest->?B),
  shipment
     (cargo->?:Product;price->?cost2:Float;source->?B;dest->?A),
  add(sum->?cost;addend1->?cost1;addend2->?cost2).
```

**Positional-slotted rules** use positional or slotted relations *as* the conclusion or any of the premises, or use positional-slotted relations *within* the conclusion or any of the premises. For instance, to avoid the 'indexed slot' conventions/assumptions in the slotted rule above, a positional-slotted rule can be positional in the conclusion and the `add` premise, and can be slotted in the `shipment` premises:

```
reciship(?cost,?A,?B) :-
  shipment(cargo->?;price->?cost1;source->?A;dest->?B),
  shipment(cargo->?;price->?cost2;source->?B;dest->?A),
  add(?cost,?cost1,?cost2).
```

Moving further towards our positional origin of the spectrum, the `shipment` premises can also be positional-slotted, as indicated in section 2. Alternatively, the conclusion can be positional-slotted as follows (the right-hand side is the normal form, "positional before slotted", of the left-hand side):

```
reciship(price->?cost;?A,?B) :-     reciship(?A,?B;price->?cost) :-
  ... .                               ... .
```

The query `reciship(price->?total;BostonMoS,LondonSciM)`, normalizing to `reciship(BostonMoS,LondonSciM;price->?total)`, again binds `?total` to `57.0`. This query could be the body of another positional-slotted rule with (one-)positional head `reciBosLon(?total)`.

The **semantics** of slotted and positional-slotted clause sets can be defined on top of the semantic basis for atoms and complex terms in section 2. Since the level of clauses is the same in all three notations, the treatment in section 2 naturally extends to slotted (and positional-slotted) generalizations of positional (LP [Llo87]) clauses. The further semantic treatment via Herbrand models and resolution proof theory directly follows the one for the positional notation [Llo87]. The semantics of typing (sorts) could be given directly but can also be reduced to the unsorted case in a well-known manner. All occurrences of a sorted variable are replaced by their unsorted counterparts plus a body-side application of a sort-corresponding unary predicate to that variable (sorted facts thus become unsorted rules). Moreover, the definition of the unary predicate reflects the subsumption relations of the sort taxonomy via rules.

The **implementation** of POSL for slotted and positional-slotted clauses has been following the semantics via an extension of the Java-based jDREW interpreter [Spe02] called OO jDREW [http://www.jdrew.org/oojdrew]. The implementation of typing has been performed directly (without the above reduction) for various sorted Prolog systems before RDFS and OWL became available as Web-based taxonomy languages. We have adapted sorted indexing techniques for Prolog to RDFS and to the OO jDREW interpreter for POSL.

Three **applications** of POSL have used its OO jDREW implementation for product-seaking/advertising trees in the tree-similarity-based AgentMatcher system [BBY], music filtering rules in the collaborative system RACOFI Music [ABB+03], and business-analysis rules in the New Brunswick Business Knowledge Base [MB04].

## 4 Webizing Individuals, Relations, Slots, and Types

The POSL language elements of individuals, constructors, and relations can be webized, and generally can be given URIs. Since it concerns language elements wherever they occur, POSL webizing is orthogonal to the 'positional'/'slotted' distinction. Different occurrences of the same language element can thus be disambiguated by giving them different URIs.

Also, we distinguish two kinds of character sequences that have the form of URIs in the POSL KR language: An active URI, meant to identify a resource (the usual case), is enclosed in a pair of angular brackets, `<...>`, following IETF's generic URI syntax [http://gbiv.com/protocols/uri/rev-2002/rfc2396bis.html] and N3 [http://www.w3.org/2000/10/swap/Primer]; a passive URI, meant to stand for itself as a string (the unusual case), is enclosed in a pair of double quotes, `"..."`, exactly as other strings in POSL or in other languages. XML namespace prefixes and local names as well as general QNames can then be expressed via variables bound to active URIs (although XML applications like XSLT and RDF use `"..."` or even `'...'` for what is here called active URIs).

A symbolic POSL language element occurrence can be associated with an active URI via symbol-URI juxtaposition, generalizing a wide-spread convention for user-email association as in `"Fred Bird"<mailto:sales@sphip.com>`. A POSL element such as the string "Fred Bird" can also be replaced by a URI, as in `<mailto:sales@sphip.com>`.

**Webized individuals** employ active URIs in place of, or in addition to, individual-constant symbols. For example, `SpeedShip` can be associated with an active URI for the intended speed shipping company's homepage `<http://sphip.com>` to obtain the following webized individual:

```
SpeedShip<http://sphip.com>
```

Our 4-ary positional `shipment` fact from section 2 can now be extended by a shipping company as the first argument of a 5-ary fact using one of three options.

(1) The individual symbol `SpeedShip` itself can be used, as we did with `BostonMoS` etc. before webizing:

```
shipment(SpeedShip,PC,47.5,BostonMoS,LondonSciM).
```

(2) The active URI can be employed in place of the individual symbol, as practiced in RDF, N3, and other Web KR languages:

```
shipment(<http://sphip.com>,PC,47.5,BostonMoS,LondonSciM).
```

(3) The webized individual symbol can be employed, as defined in RuleML:

```
shipment(SpeedShip<http://sphip.com>,PC,47.5,BostonMoS,LondonSciM).
```

The same options exist for slotted facts, as exemplified with the most general option (3), enriched by webized `BostonMoS` and `LondonSciM` individuals:

```
shipment(shipper->SpeedShip<http://sphip.com>;
         cargo->PC;
         price->47.5;
         source->BostonMoS<http://www.mos.org/info/contact.html>;
         dest->LondonSciM<http://www.sciencemuseum.org.uk/
                          visitors/location.asp>).
```

In passing note that the new positional first argument caused all former arguments to shift by one, while the new slotted argument was added without affecting the interpretations of any existing positional or slotted arguments (thus supporting distributed development).

**Webized relations** employ active URIs in place of, or in addition to, symbolic relation names. For example, the 4-ary and 5-ary positional `shipment` relations can be uniquely distinguished via URIs pointing to different signatures:

```
shipment<http://transport.org/rels/pos/shipment#4>
shipment<http://transport.org/rels/pos/shipment#5>
```

These webized relations can now be used unambiguously as follows (the URIs with the fragmentid's `#shipment4` and `#shipment5` would be sufficient here):

```
shipment<http://transport.org/rels/pos/shipment#4>
                          (PDA,9.5,LondonSciM,BostonMoS).
shipment<http://transport.org/rels/pos/shipment#5>
                          (SpeedShip,PC,47.5,BostonMoS,LondonSciM).
```

Similarly, 4-ary, 5-ary, and varying-arity slotted `shipment` relations could be distinguished via URIs pointing to different signatures or, for the latter case, to RDFS-like `subPropertyOf` information (varying arity is represented by an "`X`"):

```
<http://transport.org/rels/slot/shipment#4>
<http://transport.org/rels/slot/shipment#5>
<http://transport.org/rels/slot/shipment#X>
```

Sample uses will be demonstrated in section 5.

**Webized slots** employ active URIs in place of, as pioneered by RDF, or in addition to, symbolic slot names. For example, the `shipment` slots can be drawn from URIs containing fragmentid's with the original slot names, except for the `charge` fragmentid, for which the local slot name `price` is kept:

```
shipment(<http://transport.org/slots/shipment#shipper>->SpeedShip;
         <http://transport.org/slots/shipment#cargo>->PC;
         price<http://ebizguide.org/slots#charge>->47.5;
         <http://trajectory.org/slots/movement#source>->BostonMoS;
         <http://trajectory.org/slots/movement#dest>->LondonSciM).
```

**Webized types** use a URI reference to an RDFS or OWL class. For example, the `Product` type can be associated with a URI for the corresponding OWL class:

```
Product<http://www.daml.org/services/owl-s/1.0/
        ProfileHierarchy.owl#Product>
```

Using this for typing the anonymous variable of our positional rule in section 3, a primitive from XML Schema Datatypes for its `cost`-like variables, and a webized `Address` type, we obtain the following Web-typed rule:

```
recishipt(?cost:Float<http://www.w3.org/TR/2001/
                      REC-xmlschema-2-20010502/#float>,
          ?A:<http://ebizguide.org/types#Address>,
          ?B:<http://ebizguide.org/types#Address>) :-
  shipment(?:Product<http://www.daml.org/services/owl-s/1.0/
                      ProfileHierarchy.owl#Product>,
           ?cost1:Float<http://www.w3.org/TR/2001/
                        REC-xmlschema-2-20010502/#float>,
           ?A,?B),
  ... .
```

## 5 Webized Atoms for Representing RDF Knowledge

Webizing is also possible for atoms, providing for their URI grounding by associating an atom with an active URI. Fact atoms can use an OID (a symbolic name or a URI possibly prefixed by a symbolic name) as a special 'zeroth' argument separated from further arguments by an up-arrow infix "^": $relation(oid\hat{}arg_1...arg_N)$.

For example, the earlier 4-ary positional and slotted facts can now be grounded using variously webized versions of names like `s1` and `s2`:

```
shipment(s1^PC,47.5,BostonMoS,LondonSciM).
shipment(<http://sphip.com/event#s2>^PDA,9.5,LondonSciM,BostonMoS).

shipment(s1<http://sphip.com/event#s1>^
         cargo->PC;
         price->47.5;
         source->BostonMoS;
         dest->LondonSciM).
shipment(<http://sphip.com/event#s2>^
         ...).
```

In the same way, a rule head atom is webized; rule body atoms can be webized similarly, where a constant queries a specific fact and a free or anonymous variable allows for any unifying fact.

For example, the earlier positional and slotted rules can now be grounded using versions of a name `r1`, querying the `s1` facts, and retrieving the `oid` of the `s2` facts:

```
recishipt(<http://sphip.com/rule#r1>^?oid,?cost,?A,?B) :-
  shipment(s1^?,?cost1,?A,?B),
  shipment(?oid^?,?cost2,?B,?A),
  add(?cost,?cost1,?cost2).

recishipt(r1^back->?oid;price->?cost;site1->?A;site2->?B) :-
  shipment(s1<http://sphip.com/event#s1>^
```

```
          cargo->?;price->?cost1;source->?A;dest->?B),
  shipment(?oid^cargo->?;price->?cost2;source->?B;dest->?A),
  add(sum->?cost;addend1->?cost1;addend2->?cost2).
```

Notice that the variable `?oid` occurs in two roles: before a `shipment` "^", for binding an OID, and after the `recishipment` "^", for outputting it as data.

**RDF descriptions** can now be conceived as grounded slotted facts using null relations.

Assuming that our 5-ary slotted fact in section 4, by virtue of the `shipper` and other slots, can only be a shipping atom, we can omit the relation name `shipment` for the following RDF:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://transport.org/slots/shipment#"
  xmlns:p="http://ebizguide.org/slots#"
  xmlns:m="http://trajectory.org/slots/movement#">
  <rdf:Description about="http://sphip.com/event#s1">
    <s:shipper rdf:resource="http://sphip.com"/>
    <s:cargo>PC</s:cargo>
    <p:charge>47.5</p:charge>
    <m:source rdf:resource="http://www.mos.org/info/contact.html"/>
    <m:dest rdf:resource="http://www.sciencemuseum...location.asp"/>
  </rdf:Description>
</rdf:RDF>
```

This can be represented as the following POSL fact, using webizing also for slots and individuals (but not using its analog to namespace prefixes, explained in the online POSL document [http://www.ruleml.org/submission/ruleml-shortation.html]):

```
(<http://sphip.com/event#s1>^
 <http://transport.org/slots/shipment#shipper>->
                       <http://sphip.com>;
 <http://transport.org/slots/shipment#cargo>->PC;
 <http://ebizguide.org/slots#charge>->47.5;
 <http://trajectory.org/slots/movement#source>->
                       <http://www.mos.org/info/contact.html>;
 <http://trajectory.org/slots/movement#dest>->
                       <http://www.sciencemuseum...location.asp>).
```

Symbolic and webized individuals are represented in the same manner here, so that a symbolic name `PC` can later be replaced by a blank node or a URI for its product catalog entry, without changing anything about the enclosing slot.

The webized varying-arity relation name from section 4, `<http://transport.org/rels/slot/shipment#X>`, can be introduced into the above RDF description as an `rdf:type`:

```
<rdf:RDF
  ...
  <rdf:Description about="http://sphip.com/event#s1">
    <rdf:type
      rdf:resource="http://transport.org/rels/slot/shipment#X"/>
    ...
  </rdf:Description>
</rdf:RDF>
```

It is directly applicable to the above POSL fact:

```
<http://transport.org/rels/slot/shipment#X>
                        (<http://sphip.com/event#s1>^
                         ...).
```

This POSL fact could also be specialized using the webized 5-ary relation name `<http://transport.org/rels/slot/shipment#5>` from section 4.

**RDF rules** can then be directly defined in POSL to process such facts.

For example, the earlier slotted rule can be modified to query such facts, retrieving the OIDs and aggregated cost of any reciprocal shippings (the URIs are abridged to symbolic names):

```
reciship(r1^forth->?oid1;back->?oid2;
            price->?cost;site1->?A;site2->?B) :-
  (?oid1^shipper>->?;cargo->?;price->?cost1;source->?A;dest->?B),
  (?oid2^shipper>->?;cargo->?;price->?cost2;source->?B;dest->?A),
  add(sum->?cost;addend1->?cost1;addend2->?cost2).
```

Such rules can be employed within a semantic search engine operating on RDF/POSL-described metadata for high-precision results.

**RDF blank nodes** are used for OIDs local to the current document. For example, the earlier shipping description can be refined by referring to a local `cargo` description using the blank node identifier `PeterMillerPC` as follows:

```
<rdf:RDF
  ...
  <rdf:Description about="http://sphip.com/event#s1">
      ...
    <s:cargo rdf:nodeID="PeterMillerPC"/>
    ...
  </rdf:Description>
  <rdf:Description rdf:nodeID="PeterMillerPC">
    <p:value>2500.0</p:value>
    <p:weight>17.5</p:weight>
  </rdf:Description>
</rdf:RDF>
```

Following the RDF semantics [Hay04] [http://www.w3.org/TR/rdf-mt], this can be represented as the following module of two POSL facts connected by the existential variable `PeterMillerPC` (modules, like N3 and TRIPLE contexts, are enclosed using "{...}", and existential variables, global to clauses but local to modules, are prefixed by a "_" and usable both as slot fillers and OIDs):

```
{
  (<http://sphip.com/event#s1>^
   ...
   <http://transport.org/slots/shipment#cargo>->_PeterMillerPC;
   ...).
  (_PeterMillerPC^
   <http://ebizguide.org/slots#value>->2500.0;
   <http://ebizguide.org/slots#weight>->17.5).
}
```

Module-confined, clause-conjoining existential variables, introduced here for representing RDF blank nodes, can also be used for all other POSL notations discussed earlier, such as for positional relations.

## 6 Conclusions

This paper introduces a kernel of positional and slotted notions plus notations for KR on the Semantic Web. Several extensions can be found in the online POSL document [http://www.ruleml.org/submission/ruleml-shortation.html].

One recent POSL notion briefly mentioned is optional signature declarations, which can be helpful for knowledge base integration in Web-distributed development. Signatures enforce types for arguments of all clauses with given relation name, arity, and possibly slot names. These types can, as we have seen, refer to classes defined in a Web taxonomy language such as RDFS or OWL.

Current work concerns a general POSL treatment of slot cardinalities. While the F-Logic system FLORA-2 distinguishes single-valued from set-valued attributes, the description logic system OWL DL provides exact, min, and max cardinality restrictions. The original POSL design as presented in this paper employs single-valued slots. However, the available plex data structure introduced here can be restricted to represent finite sets, which correspond to multiple-valued slots. These can also be used as arguments to positional relations, and their connection to finite domains in LP systems is being investigated.

Only relations and their defining derivation rules have been presented here. However, functions defined by transformation rules can be added. These would also permit to regard slot names as functions applied, in suffix notation, to OIDs, as in the F-Logic semantics.

A topic of future research is the issue of extending OIDs towards a general notion of object identity. In general, to the left of the POSL "^" infix, there can be several ($M$) objects targeted by an ($N$-ary) operation: $operation(oid_1...oid_M\hat{}arg_1...arg_N)$. This can provide a bridge

from the declarative OO KR rules studied here to reaction rules, Web Services, and OOP. For example, with $M$=2 and $N$=1, the message `transfer(checking1,savings2^3500)` addresses equally focussed account objects `checking1` and `savings2` in a positional manner, using the single argument `3500` for the amount to be transferred in the 'from-to' direction. Besides such "`,`"-ordered receiver objects, also "`;`"-unordered ones can be used for parallel message broadcasting. For example, with $M$=2 and $N$=2, the message `equalize(checking1;checking2^min->1000;max->2000)` addresses equally focussed objects `checking1` and `checking2` in an unordered manner, using slotted arguments for the minimal amount, `1000`, and the maximal amount, `2000`, to be left in both accounts after a balancing transfer in either direction, if their amounts were unequal. In practice, such symbolic account names would be replaced by password-protected URIs.

The RuleML and SWRL groups, and the Semantic Web community at large, may provide further requirements and use cases for future POSL directions.

## References


[ABB+03] Michelle Anderson, Marcel Ball, Harold Boley, Stephen Greene, Nancy Howse, Daniel Lemire, and Sean McGrath. RACOFI: A Rule-Applying Collaborative Filtering System. In *Proc. Collaboration Agents: Autonomous Agents for Collaborative Environments*. Halifax, Canada, October 2003.

[BBY] Virendra C. Bhavsar, Harold Boley, and Lu Yang. A Weighted-Tree Similarity Algorithm for Multi-Agent Systems in e-Business Environments. In *Proc. Business Agents and the Semantic Web (BASeWEB) Workshop*, pages 53–72. NRC 45836, June 2003. To appear in: Computational Intelligence.

[Bol03] Harold Boley. Object-Oriented RuleML: User-Level Roles, URI-Grounded Clauses, and Order-Sorted Terms. In *Proc. Rules and Rule Markup Languages for the Semantic Web (RuleML-2003)*. LNCS 2876, Springer-Verlag, October 2003.

[DS04] Mike Dean and Guus Schreiber. OWL Web Ontology Language – Reference. W3C Recommendation, W3C, February 2004.

[Hay04] Patrick Hayes. RDF Semantics. W3C Recommendation, W3C, February 2004.

[KL89] Michael Kifer and Georg Lausen. F-Logic: A Higher-Order Language for Reasoning about Objects, Inheritance, and Scheme. In James Clifford, Bruce G. Lindsay, and David Maier, editors, *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pages 134–146, Portland, Oregon, 31 May–2 June 1989.

[Llo87] John W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, Heidelberg, New York, 1987.

[MB04] Anna Maclachlan and Harold Boley. Rules for Enhancing Facts Extracted from Business Data: The RuleML NBBizKB. Technical report, Submitted to Journal, February 2004.

[Spe02] Bruce Spencer. The Design of j-Drew: A Deductive Reasoning Engine for the Web. In *Joint CoLogNet Workshop on Component-based Software Development and Implementation Technology for Computational Logic Systems of LOPSTR '02*. Technical University of Madrid, Spain, September 2002.