OWL Web Ontology Language as a Scripting Language for Smart Space Applications

Espen Suenson, Johan Lilius, and Iván Porres

Department of Information Technologies
Åbo Akademi University
Turku, Finland
givenname.surname@abo.fi

Abstract. We describe a scripting language for smart space applications based on the OWL Web Ontology Language. The design goals of the scripting language are: I. A syntax that easily expresses common script applications for smart spaces. II. Based on OWL to enable synergy with semantic web technologies. III. Ease of implementation by using existing OWL reasoners.
We motivate the design of the scripting language and give some examples of how to use it. Furthermore, we will give a formal definition of the syntax and semantics of the scripting language based on the OWL 2 definition.

Keywords: OWL; smart space; ontology; scripting; semantic web

1 Introduction

The Semantic Web is based on the architecture of the World Wide Web. However, there is also a wish to permit greater interoperability between portable devices, and in particular to permit portable devices to discover and exchange information about their immediate physical environments, that is, their context. A Smart Space is an idea of connecting such devices in a local way similar to the Semantic Web. [9] In this paper we define a smart space as a local network about the size of a room consisting of smart phones, tablet PCs and the like, that is, devices that are fairly computationally powerful.

Smart-M3 is a communication infrastructure for smart devices conceived by Nokia. In Smart-M3, computational devices (smart phones and others) communicate within a limited physical space through a central database, the System Information Broker (SIB). The communication is in the form of RDF. [3] A program that communicates with the SIB is called a Knowledge Processor (KP). An application may consist of several KPs acting together, KPs residing on other devices and programs not communicating directly with the SIB. The KPs do not communicate with each other since this is done via the SIB.

In this paper we describe a scripting language for simple smart space applications. The language targets the Smart-M3 platform. The first design goal of our scripting language is that it should make it easier to make smart space applications. To satisfy this design goal our scripting language must make it easy to express simple tasks, see the examples of section 3.

The second design goal is that the scripting language is based on OWL. [7] Hence we will refer to the scripting language as OWL Script. By building on well deliberated standards we hope to save some of the work of inventing new standards and to enable greater interoperability between standards.

The third design goal is that OWL Script should be able to be implemented without expending the effort to create a compiler or interpreter from ground. The use of OWL means that it is possible to use existing OWL reasoning engines as the computational core of OWL Script.

The main research question that we are trying to answer is how to make a practically oriented scripting layer that combines the ideas of smart spaces and the Semantic Web technologies. 'Practical' is meant to be understood both from from the point view of the scripting programmer as well as of the scripting language implementer.

2 Related work

Luukkala and Niemelä have proposed to use an Answer Set Programming implementation as the basis for rule based computation on top of the Smart-M3 platform. [4] Their approach provides for rule based constraint programming in a way that is computationally more powerful than using OWL. However, in their approach variables can only hold single individuals. We believe that our approach where variables refer to sets provides for more intuitive scripts with regard to smart spaces.

Oliver and Honkola suggests using the WQL query language of Smart-M3 to implement scripts similar to those considered in this paper. [5] However, they do not describe the scripting language in detail.

As the individual scripts in our approach have a form similar to 'IF a THEN b', OWL Script, if viewed as a collection of rules, bears some resemblance to a production rule system (with implementations such as Drools). However, the focus of our work is not on the rules as a system, but rather on the individual 'rule' level and its expression in OWL. Each script or 'rule' will execute independently.

The Rule Interchange Format [8], RuleML [2] and REWERSE I1 Rule Markup Language are rule formats that can be executed. However, the purpose of these formats is mainly to serve as interchange formats between different rule notations, hence they do not address specifically a syntax suitable for smart space scripting or implementation via OWL reasoners.

A number of OWL reasoning engines exist which could conceivably be used as back ends in implementing the proposed scripting languages. To name some: Pellet, Hermit, the Jena framework, Bossam and Racer.

3 Programming with OWL Script

Shown in table 1 is an example KP in OWL Script. It checks whether a named user is listed as busy in the SIB, and whether there's a call to his phone. If so, the phone's voicemail is activated by inserting a command in the SIB that will presumably be read by the phone at a later time and acted upon.

Table 1. OWL Script programming example

```
with  user = /User and /Id == "peter.smith@abo.fi"
      phone = /Phone /Owner user
      busyUser = user and /Busy
      ringingPhone = phone and /IncomingCall

when  busyUser, ringingPhone

then  insert /ActivateVoicemail( ringingPhone )
```

The program has three clauses. The 'with' clause defines an ontology in the KP that we can think of as a declaring local variables. Any variable that has the '/' access modifier prefixed will be fetched from the SIB. Variables without access modifiers are local to the KP. All variables are sets, so even though we assume that there's only one user with a given 'Id', the variables 'phone' and 'ringingPhone' could easily be conceived to have multiple members.

The 'when' clause decides if the KP will take any action. In this case, if both 'busyUser' and 'ringingPhone' are nonempty sets, the 'then' clause will be carried out. The 'then' clause specifies the action that will be taken if the KP fires. In our example a unary predicate will be inserted into the SIB over the set 'ringingPhone'.

Table 2. OWL Script programming example

```
when  /SIB_Location and /EmployeeRestaurant,
      dev/Time > "11:00:00" ^^xsd:time,
      dev/Time < "13:30:00" ^^xsd:time

then  insert /AtLunch( user )
```

In table 2 we see an example that lists the user as busy in the SIB if he is present in a restaurant and it is around lunch time. The example demonstrates communication with the hosting device through the use of the 'dev/' access modifier. We also see that some things are outside the scope of OWL Script and must be provided by the device, such as the current time.

4 Syntax

The syntax of OWL Script is defined by extending the syntax of OWL 2 Functional Syntax and adding the syntactical categories for those things that are exclusive to OWL Script.

Table 3. Extension of the OWL 2 syntax

AccessModifier	:=	'/' | 'dev/'
NonlocalIdentifier	:=	**AccessModifier IRI**
Quantifier	:=	**nonNegativeInteger** |
		'min' **nonNegativeInteger** |
		'max' **nonNegativeInteger** |
		'only'
Class	+=	**NonlocalIdentifier**
ObjectProperty	+=	**NonlocalIdentifier**
DataProperty	+=	**NonlocalIdentifier**
DataRange	+=	('==' | '>' | '<') **Literal**
ClassExpression	+=	'(' **ClassExpression** ')' |
		ClassExpression 'and' **ClassExpression** |
		ClassExpression 'or' **ClassExpression** |
		ClassExpression ',' **ClassExpression** |
		[**Quantifier**] **PropertyExpression**
PropertyExpression	:=	**ClassExpression ObjectPropertyExpression ClassExpression** |
		'?' **ObjectPropertyExpression ClassExpression** |
		ClassExpression ObjectPropertyExpression '?' |
		DataPropertyExpression DataRange
ClassAxiom	+=	**ClassExpression** '=' **ClassExpression**

The extension of the OWL syntax is based on the OWL 2 Functional Syntax definition. [7] The extensions are shown in table 3 in extended BNF notation. The '+=' symbol that is used in some places instead of the normal ':=' means that the syntactic categories of OWL 2 are augmented with the productions shown, that is, those productions can be used in addition to those already defined in the standard. Syntactic categories that are not defined here are defined in the OWL 2 standard.

The access modifiers '/' and 'dev/' are the only extensions of the basic OWL 2 syntax that cannot be expressed in ordinary OWL 2. The rest of the extensions in table 3 can be regarded as syntactic sugar for OWL 2 expressions. Of course, the constructs of table 4 cannot be expressed in OWL.

Table 4. OWL Script syntax

RDFExpression	:=	**NonlocalIdentifier** '(' **ClassExpression** ')' |
		NonlocalIdentifier '(' **ClassExpression** ',' **ClassExpression** ')' |
		NonlocalIdentifier '(' **ClassExpression** ',' **Literal** ')'
Action	:=	'new' **IRI** |
		'insert' **RDFExpression** |
		'remove' **RDFExpression**
KnowledgeProcessor	:=	['with' { **Axiom** }] 'when' **ClassExpression** 'then' { **Action** }

Disregarding the access modifiers, the body of the 'with' and 'when' clauses is standard OWL with added syntactic shortcuts. The 'then' clause builds a set of RDF triples and directs insertion or removal of triples. The full syntax for KPs is shown in extended BNF notation in table 4.

It should be noted that the presented syntax is ambiguous, but that the ambiguity can be resolved by the proper use of parentheses.

5 Semantics

The semantics of an OWL ontology O is given in terms of an interpretation $\mathsf{I} = (\Delta_{\mathsf{I}}, \Delta_{\mathsf{D}}, \cdot^{\mathsf{C}}, \cdot^{\mathsf{OP}}, \cdot^{\mathsf{DP}}, \cdot^{\mathsf{I}}, \cdot^{\mathsf{DT}}, \cdot^{\mathsf{LT}}, \cdot^{\mathsf{FA}})$ where Δ_{I} is an object domain of individuals and Δ_{D} is a data domain of data values, $\cdot^{\mathsf{C}}$ is the class interpretation function, $\cdot^{\mathsf{OP}}$ is the object property interpretation function, $\cdot^{\mathsf{DP}}$ is the data property interpretation function, $\cdot^{\mathsf{I}}$ is the individual interpretation function, $\cdot^{\mathsf{DT}}$ is the datatype interpretation function, $\cdot^{\mathsf{LT}}$ is the literal interpretation function and $\cdot^{\mathsf{FA}}$ is the data facet interpretation function. [6]

To describe the contents of the SIB and hosting device triple stores we define additionally the interpretation functions $\cdot^{\mathsf{C}^{\mathsf{SIB}}}$, $\cdot^{\mathsf{OP}^{\mathsf{SIB}}}$, $\cdot^{\mathsf{DP}^{\mathsf{SIB}}}$, $\cdot^{\mathsf{C}^{\mathsf{dev}}}$, $\cdot^{\mathsf{OP}^{\mathsf{dev}}}$ and $\cdot^{\mathsf{DP}^{\mathsf{dev}}}$.

To describe the semantics of variables with access modifiers we require the following to hold given $/\mathsf{name}$ in O:

$$\begin{aligned}
&\text{if } /\mathsf{name} \in \mathsf{V_C} \text{ then} \\
&\qquad x \in (\mathsf{name})^{\mathsf{C}^{\mathsf{SIB}}} \Rightarrow x \in (/\mathsf{name})^{\mathsf{C}} \\
&\text{if } /\mathsf{name} \in \mathsf{V_{OP}} \text{ then} \\
&\qquad (x,y) \in (\mathsf{name})^{\mathsf{OP}^{\mathsf{SIB}}} \Rightarrow (x,y) \in (/\mathsf{name})^{\mathsf{OP}} \\
&\text{if } /\mathsf{name} \in \mathsf{V_{DP}} \text{ then} \\
&\qquad (x,y) \in (\mathsf{name})^{\mathsf{DP}^{\mathsf{SIB}}} \Rightarrow (x,y) \in (/\mathsf{name})^{\mathsf{DP}}
\end{aligned}$$

where $\mathsf{V_C}$, $\mathsf{V_{OP}}$ and $\mathsf{V_{DP}}$ are the vocabularies of classes, object properties and data properties in O.

We define exactly similar requirements to hold for the device interpretation functions $\cdot^{\mathsf{C}^{\mathsf{dev}}}$, $\cdot^{\mathsf{OP}^{\mathsf{dev}}}$ and $\cdot^{\mathsf{DP}^{\mathsf{dev}}}$. Note that the vocabulary membership conditions can be resolved syntactically.

Table 5. Extension of the datatype interpretation function

Data Range	**Interpretation** $\cdot^{\mathsf{DT}}$
`== lt`	$\{(\mathsf{lt})^{\mathsf{LT}}\}$
`> lt`	$(\mathsf{DT})^{\mathsf{DT}} \cap (\mathsf{xsd : minExclusive, lt})^{\mathsf{FA}}$
`< lt`	$(\mathsf{DT})^{\mathsf{DT}} \cap (\mathsf{xsd : maxExclusive, lt})^{\mathsf{FA}}$

where DT is the datatype of the literal `lt` if different from rdf:PlainLiteral; otherwise it is xsd:integer.

We need to define the semantics for the rest of the syntactic additions we have made to OWL. We do this by extending the domain of the interpretation functions $\cdot^{\mathsf{DT}}$ and $\cdot^{\mathsf{C}}$ from the OWL 2 Direct Semantics definition. [6] The extension of $\cdot^{\mathsf{DT}}$ is given in table 5. The extension of $\cdot^{\mathsf{C}}$ is given in table 6. We have added only one axiom form to the syntax, the condition that it imposes is given in table 7.

We emphasize that the extensions are merely syntactical, they do not extend the computational power of OWL. In other words, it is possible, though not always efficient, to implement the extensions in ordinary OWL. In particular, the syntactic form '`CE`$_1$ `,` `CE`$_2$' can be implemented as $(\exists\top.(\mathsf{CE}_1)^{\mathsf{C}}) \sqcap (\exists\top.(\mathsf{CE}_2)^{\mathsf{C}})$ (borrowing notation from [1]) and the form '`CE OPE ?`' can be implemented with the use of the OWL inversion operator ObjectInverseOf.

To define the semantics of the parts of OWL Script that doesn't directly extend OWL, we first define an interpretation function for RDF expressions, $\cdot^{\mathsf{RDF}}$. The definition of $\cdot^{\mathsf{RDF}}$ is given in table 8.

The semantics of the 'Action' syntactic category is as follows: An action of the form '`new IRI`' defines a new blank RDF node x such that $(\mathsf{IRI})^{\mathsf{C}} = \{x\}$. An action of the form '`insert RDFE`' indicates that the RDF triple set $(\mathsf{RDFE})^{\mathsf{RDF}}$ is to be written to the SIB or the hosting device triple store, depending on the access modifier. Similarly, '`remove RDFE`' indicates triple sets that are to be deleted.

The triple sets to be inserted or removed may be described by several actions. The complete sets are written atomically to the SIB and the hosting device, but the writing operations to the SIB and the device are not atomic with respect to each other. Conflicts between 'insert' and 'remove' actions are resolved in favor of insertion.

Table 6. Extension of the class expression interpretation function

Class Expression	**Interpretation** $\cdot^{\mathsf{C}}$
`( CE )`	$(\mathsf{CE})^{\mathsf{C}}$
`CE`$_1$ `and CE`$_2$	$(\mathsf{CE}_1)^{\mathsf{C}} \cap (\mathsf{CE}_2)^{\mathsf{C}}$
`CE`$_1$ `or CE`$_2$	$(\mathsf{CE}_1)^{\mathsf{C}} \cup (\mathsf{CE}_2)^{\mathsf{C}}$
`CE`$_1$ `, CE`$_2$	$\begin{cases} \delta \text{ if } \exists x, y : x \in (\mathsf{CE}_1)^{\mathsf{C}} \text{ and } y \in (\mathsf{CE}_2)^{\mathsf{C}} \\ \emptyset \text{ otherwise} \end{cases}$ where $\delta \subseteq \Delta_{\mathsf{I}}$ and $\delta \neq \emptyset$
`CE`$_1$ `OPE CE`$_2$	$\{x \mid \exists y : (x,y) \in (\mathsf{OPE})^{\mathsf{OP}} \text{ and } x \in (\mathsf{CE}_1)^{\mathsf{C}}, y \in (\mathsf{CE}_2)^{\mathsf{C}}\}$
`n CE`$_1$ `OPE CE`$_2$	$\{x \mid \#\{y \mid (x,y) \in (\mathsf{OPE})^{\mathsf{OP}} \text{ and } x \in (\mathsf{CE}_1)^{\mathsf{C}}, y \in (\mathsf{CE}_2)^{\mathsf{C}}\} = n\}$
`min n CE`$_1$ `OPE CE`$_2$	$\{x \mid \#\{y \mid (x,y) \in (\mathsf{OPE})^{\mathsf{OP}} \text{ and } x \in (\mathsf{CE}_1)^{\mathsf{C}}, y \in (\mathsf{CE}_2)^{\mathsf{C}}\} \geq n\}$
`max n CE`$_1$ `OPE CE`$_2$	$\{x \mid \#\{y \mid (x,y) \in (\mathsf{OPE})^{\mathsf{OP}} \text{ and } x \in (\mathsf{CE}_1)^{\mathsf{C}}, y \in (\mathsf{CE}_2)^{\mathsf{C}}\} \leq n\}$
`only CE`$_1$ `OPE CE`$_2$	$\{x \mid \forall y : (x,y) \in (\mathsf{OPE})^{\mathsf{OP}}, x \in (\mathsf{CE}_1)^{\mathsf{C}} \text{ implies } y \in (\mathsf{CE}_2)^{\mathsf{C}}\}$
`? OPE CE`	$\{x \mid \exists y : (x,y) \in (\mathsf{OPE})^{\mathsf{OP}} \text{ and } y \in (\mathsf{CE})^{\mathsf{C}}\}$
`CE OPE ?`	$\{y \mid \exists x : (x,y) \in (\mathsf{OPE})^{\mathsf{OP}} \text{ and } x \in (\mathsf{CE})^{\mathsf{C}}\}$
`DPE DR`	$\{x \mid \exists y : (x,y) \in (\mathsf{DPE})^{\mathsf{DP}} \text{ and } y \in (\mathsf{DR})^{\mathsf{DT}}\}$

The quantified forms of the syntactic forms '`? OPE CE`', '`CE OPE ?`' and '`DPE DR`' have been omitted since they are similar to the forms shown. `CE` is a class expression, `OPE` is an object property expression, `DPE` is a data property expression and `DR` is a data range.

We can now define the semantics of a KP '`with O when CE then A`'. Let I be an interpretation that satisfies O under the additional SIB and device store requirements. If $(\mathsf{CE})^{\mathsf{C}} \neq \emptyset$ then the actions A are carried out and updates the SIB and device triple stores according to the definition above of action semantics.

6 Conclusion

The scripting language we present demonstrates that it is feasible to use OWL as a programming language in a limited domain setting. We believe that OWL Script is easy to use for examples similar to the ones we have given. As we work on incorporating further domains and examples, the language can be extended to accomodate these.

Table 7. Extension of the class expression axiom satisfaction conditions

Axiom	**Condition**
$\mathtt{CE_1 = CE_2}$	$(\mathsf{CE}_1)^{\mathsf{C}} = (\mathsf{CE}_2)^{\mathsf{C}}$

Table 8. RDF expression interpretation function

RDF Expression	**Interpretation** $\cdot^{\mathsf{RDF}}$
`NLI ( CE )`	$\{((x)^{\bar{\mathsf{I}}}, \mathsf{rdf : type}, \mathsf{IRI}) \mid x \in (\mathsf{CE})^{\mathsf{C}}\}$
`NLI ( CE₁ , CE₂ )`	$\{((x)^{\bar{\mathsf{I}}}, \mathsf{IRI}, (y)^{\bar{\mathsf{I}}}) \mid$ $x \in (\mathsf{CE}_1)^{\mathsf{C}}$ and $y \in (\mathsf{CE}_2)^{\mathsf{C}}\}$
`NLI ( CE , lt )`	$\{((x)^{\bar{\mathsf{I}}}, \mathsf{IRI}, \mathsf{lt}) \mid x \in (\mathsf{CE})^{\mathsf{C}}\}$

where IRI is the IRI of the non-local identifier `NLI` and $\cdot^{\bar{\mathsf{I}}}$ is the inverse of $\cdot^{\mathsf{I}}$.

The most important task of our future research is to make an implementation of OWL Script and see how well it performs in practice, in particular to gauge how suitable the syntax is to script programmers.

References

1. Baader, F., Nutt, W.: Basic description logics (2003)
2. Grosof, B.N., Gandhe, M.D., G, M.D., Finin, T.W.: Sweetjess: Inferencing in situated courteous ruleml via translation to and from jess rules. In: In: Proceedings of the ISWC 02 International Workshop on Rule Markup Languages for Business Rules on the Semantic Web, June 2002, Sardinia, Italy (2003)
3. Honkola, J., Laine, H., Brown, R., Tyrkko, O.: Smart-m3 information sharing platform. IEEE Symposium on Computers and Communications pp. 1041–1046 (2010)
4. Luukkala, V., Niemelä, I.: Enhancing a smart space with answer set programming. In: Proceedings of the 2010 international conference on Semantic web rules. pp. 89–103. RuleML'10, Springer-Verlag, Berlin, Heidelberg (2010)
5. Oliver, I., Honkola, J.: Personal semantic web through a space based computing environment. In: Proceedings of the 2nd International Conference on Semantic Computing (2008)
6. W3C recommendation: Owl 2 web ontology language direct semantics (2009), http://www.w3.org/TR/2009/REC-owl2-direct-semantics-20091027/
7. W3C recommendation: Owl 2 web ontology language structural specification and functional-style syntax (2009), http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/
8. W3C Working Group Note: Rif overview (2009), http://www.w3.org/TR/2010/NOTE-rif-overview-20100622/
9. Wang, X., Dong, J.S., Chin, C., Hettiarachchi, S., Zhang, D.: Semantic space: An infrastructure for smart spaces. IEEE Pervasive Computing 3, 32–39 (2004)