Semantic Web Model and Reasoning Based on F-logic

1 Tao He, 2 Liping Li, 3 Huazhong Li, 4 Jiangang Chen

Abstract

In order to model context in pervasive computing environments, and support logic-based context reasoning, a core ontology modeling framework for OWL-S (Ontology Web Language for Services) and a reasoning mechanism using F-logic are proposed in this paper. It analyzes the construction of OWL-S model type, gives F-logic axioms, derivation rules and methods to check the consistency and satisfiability of context information. The mechanism and method can verify some global attributes of Semantic Web system by model checking based on first-order logic, provide formal model for OWL-S, and expand its expression ability and verification ability.

Keywords: Web Service, F-logic, Software Requirement, Model Checking, Verification
2. Formalism of Software Requirement Meta-framework based on Semantic Web

OWL-S provides a core ontology framework for Web services of software requirement framework meta-model to describe the specification of properties and ability of Web services. It has unambiguous and machine-translation capabilities. This paper maps the static part of the OWL-S to F-logic for semantic interoperability, and presents a formalized semantics modeling method of OWL-S model theory, which represents global property and framework of software requirement framework meta-model as F-logic formula. This method can use model checking based on the first-order logical to verify some global properties of the OWL-S service system.

For a given domain, assuming that all the required domain elements obtains from resources Resource. As a result, the basic elements of software requirement meta-framework (Role, Goal, Process and Service) can be defined as follows:

\[
\text{softwareMetamodel} = \{ \text{Role} \rightarrow \text{String}, \text{Goal} \rightarrow \text{String}, \text{Process} \rightarrow \text{Resource}, \text{Service} \rightarrow \text{Resource} \}.
\]

Firstly, formalize relationship between each layer. Role R is responsible for their corresponding role goal; executor actors prefer its individual goal.

Process can directly or jointly achieve functional goals. They are also helpful in achieve the non-function goal.

The meanings of following formula are as below,

- the first constraint expresses that executor \( a \) may undertake many role \( R \);
- the second constraint expresses that \( rp \) cannot be either role goal or individual goal at the same time;
- the third constraint expressed executor \( a \) prefer personal goal \( PG \), and role \( r \) can undertake its goal of role \( RoleGoal \).

\[
\begin{align*}
\forall a: \text{Actor} & \exists R: \text{Role} \text{ plays@}_a \Rightarrow R \land R[\text{number}\rightarrow\{1, 2, \ldots, n\}] \iff \\
& (\text{actor::Resource} \land \text{role::Resource} \land \text{plays@}_a \Rightarrow \text{Role} \land \text{bePlays@}_R \Rightarrow \text{Actor}) \\
& \land rp \rightarrow \{1 \iff rp: \text{RoleGoal} \land rp: \text{PersonalGoal}.
\end{align*}
\]

\[
\forall a: \text{Actor}, r: \text{Role}, \exists PG: \text{PersonalGoal}, RG: \text{RoleGoal}
\]

\[
\begin{align*}
& (\text{prefer@}_a \Rightarrow PG \land PG[\text{number}\rightarrow\{1, 2, \ldots, n\}]) \lor \\
& (\text{takesCharge@}_r \Rightarrow RG \land RG[\text{number}\rightarrow\{1, 2, \ldots, n\}])
\end{align*}
\]

Where \( rp \) is a personal goal, \( RG \) is the role goal, \( a \) is the executor, \( r \) is a role.

The following formula shows that there is at least a process \( P \) can achieve functional objectives \( f_g \), and contribute to non-functional goal \( nfg \).

\[
\begin{align*}
\text{Goal} & \Rightarrow \text{function,} \\
\text{NonFunctionalGoal} & \Rightarrow \text{degree}.
\end{align*}
\]

\[
\begin{align*}
\forall f_g: \text{FunctionalGoal}, nfg: \text{NonFunctionalGoal}, \exists P: \text{Process}, \text{achieves@}_P \Rightarrow \\
& \text{FunctionalGoal} \land P[\text{number}\rightarrow\{1, 2, \ldots, n\}] \land \\
& \text{contributes@}_P \Rightarrow \text{NonFunctionalGoal} \land P[\text{number}\rightarrow\{1, 2, \ldots, n\}] \\
\iff & \text{achieves@}_P \Rightarrow \text{FunctionalGoal} \land \\
& \text{contributes@}_P \Rightarrow \text{NonFunctionalGoal} \land \\
& \text{beAchieves@}_f \Rightarrow \text{Process} \land \\
& \text{beContributes@}_n \Rightarrow \text{Process}.
\end{align*}
\]
Where $P$ is a process; $fg$ is a functional target instance; $nfg$ is non-functional objectives. The below formula means Web services can identify the process, and a process can be identified by at least one service.

$$\forall P: \text{Process}, \exists S: \text{Service} \quad \text{achieves} @ P \rightarrow \text{Service} \wedge S[\text{number}=>\{1 , 2 , \ldots , n\}]$$

$\leftarrow$ realizes@S $\rightarrow$ Process $\wedge$

beRealized@P $=>$ Service.

Where $P$ is process instance, $S$ is service instance. Above formula shows that: if one service can identify the process $P$ at least, the process $P$ can achieve at least one service.

The following formula mean a goal is a function goal or the non-function goal, and each function goal is composed of operation, object and manner:

$\text{FunctionGoal} [ \text{Operation :: Resource} =>> \text{OperationFunctionGoal} , \text{Object:: Resource} =>> \text{ObjectFunctionGoal} , \text{Manner:: Resource} =>> \text{MannerFunctionGoal} ]$.

$\forall fg: \text{FunctionalGoal} , m: \text{Manner}$ EXISTS ob:Object, op:Operation

$\leftarrow$ hasOperation@fg $=>$ op $\wedge$ hasObject@fg $=>$ ob $\wedge$ hasManner@fg $=>> Manner$

$\leftarrow$ hasOperation@fg $=>$ Operation $\wedge$

hasObject@fg $=>$ Object $\wedge$

hasManner@fg $=>> Manner$ $\wedge$

beOperated@op $=>$ FunctionalGoal $\wedge$

beObjected@ob $=>$ FunctionalGoal $\wedge$

beManner@m $=>> \text{FunctionalGoal}.$

Where, $fg$ is a functional target, $m$ is the way, $ob$ is the object, $op$ is the operation. This restraint indicates: there are some object $ob$ and operates $op$ to realize function goal $fg$, and each way realizes the function goal $fg$ which has its style $m$.

The following formula mean a non-function goal may be a quantity goal or a quality goal.

$\text{NonfunctionalGoal} [ \text{QuantitativeGoal} => \text{number} , \text{QualitativeGoal} => \text{quality} ]$.

A goal can be decomposed into a set of sub-goals, and the decomposition relations can be divided into Mandatory, Optional, Alternative, OR.

At the same time, the constraints of depend, or exclude exist in the target. Relationship hasGoal is used to indicate the ultimate goal set an actor has, after target refinement and reification.

The following formula indicates that a goal can be variable decomposed to some target under certain constraints.

$\text{VariabilityDecomposite}@ \text{GoalA} =>> \text{GoalB} \wedge \text{Constraint}@ \text{GoalC} =>> \text{GoalD}.$

$\text{VariabilityDecomposition} [ \text{Mandatory} =>> \text{Goal} , \text{Optional} =>> \text{Goal} , \text{Alternative} =>> \text{Goal} , \text{OR} =>> \text{Goal} ]$.

Following $g$, $g'$ are the goal example, $G$, $G1$ are the goal collection, $a$ are executors,

1. FORALL a:Actor, $G, G1: \text{Goal EXISTS g':G1}$

$(G1: G \leftarrow g: G) \wedge (g: G \leftarrow g': G) \wedge g': G \wedge \neg(g: G1) \leftarrow \text{hasGoal}@a => G \wedge \text{Mandatory}@g => G1.$

And vice versa.

2. FORALL a:Actor , $G, G1: \text{Goal EXISTS g':G1}$

$(g: G \leftarrow (g': G1 \wedge g': G)) \wedge \neg (g: G1) \leftarrow \text{hasGoal}@a => G \wedge \text{Optional}@g => G1.$

And vice versa.

3. FORALL a:Actor , $G, G1: \text{Goal EXISTS g':G1}$

$(g': G1 \wedge g: G \leftarrow g: G) \wedge (g \leftarrow g': G1 \wedge g': G) \wedge \neg (g: G1) \leftarrow \text{hasGoal}@a: => G \wedge \text{Alternative}@g => G1.$

And vice versa.
FOR ALL a:Actor, G,G1:Goal EXISTS g':G1

\[ \text{hasGoal@ } a \Rightarrow G \land \text{OR}@g \Rightarrow G1 \leftarrow (g' \land g' \land g : G) \land (g : G \leftarrow g' \land g' : G) \land \neg (g : G1) \].

And vice versa.

FOR ALL a:Actor, G,G1:Goal EXISTS g':G1

\[ (G1 \leftarrow g) \land \neg (g : G1) \leftarrow \text{hasGoal@ } a \Rightarrow G \land \text{Depend}@g \Rightarrow G1 \land \text{Depend}:\text{Constraint} \land \text{Exclude}:\text{Constraint}. \]

And vice versa.

The rule 1 explains that executor \( a \) has goal set \( G \), and can force implement goal set \( G1 \) through goal \( g \), from this may infer \( G1 \) is the subset of \( G \), \( g \) belongs to \( G \) and \( g \) does not belong to \( G1 \). Vice versa this rule, namely this rule's counter inference is also correct.

3. F-Logic Inference of Meta-Model

In order to verify the OWL-S ontology, the consistency, satisfiability, implication, and equivalence, examples of the F-logic models must be verified. When it comes to consistency of the model, check whether knowledge is meaningful, whether there are some model explanation I to O, whether there are some the possible explanations I to C, whether problems can be reduced to the consistency and so on. Regardless of whether the two classes are represented by the same instance set, we must check the equivalence of model with reasoning. For verifying the model, we must inspect whether the individual \( i \) is an example of class \( C \), and retrieve the individual set of illustration \( C \).

The F-logic reasoning logical task we consider is a standard reasoning task, usually being considered in the text:

- satisfiability checking: The duty is to inspect whether the knowledge foundation of the F-logic is consistent (to have a model);
- class satisfiability checking: The expansion of class \( A \) is non-empty, for example, \( A \) may have at least one instance;
- classification inspection of class: The duty is to check whether a given class \( C \) is expansion subset of another class \( D \), for example, \( C \) is subclass of \( D \);
- example inspection: The duty is to check whether to follow a specific description logic knowledge foundation. In this foundation a given individual is a member of a specified class.
- Connection’s inquiry answer: The duty is when assigning a given variable, to check whether a connection inquiry is logical deduction of a description logic knowledge foundation.

3.1. F-logic rule

Regular F-logic procedure \( P \) is composed of the form of rules such as the following:

\[ \text{h} \leftarrow \text{b}_1, \ldots, \text{b}_m, \text{not } \text{c}_1, \ldots, \text{not } \text{c}_n, \]

Where \( h \), \( b_1 \), \( \ldots \), \( b_m \), \( c_1 \), \( \ldots \), \( c_n \) are the free equivalent atoms or molecules. \( h \) is the head atom of \( r \), \( B^+(r) = \{ b_1, \ldots, b_m \} \) is the positive body atom set of \( r \), and \( B^-(r) = \{ c_1, \ldots, c_n \} \) is the negative body atom set of \( r \). If \( B^-(r) = \emptyset \), then \( r \) is positive. If each variable in \( r \) appears in \( B \), \( r \) is safe. If each rule \( r \in P \) is positive (is correspondingly safe), then \( P \) is positive (is correspondingly safe). Moreover, each F-logic program contains the following rules. These rules make semantic of molecular axiomatizable:

\[ x::z \leftarrow x::y, y::z \]
\[ x::z \leftarrow x::y, y::z, x::x \]
The first rule above axiomatizes the transmission of subclass relations; the second rule axiomatizes the inheritance of class member; the third rule axiomatizes the fact that each class is a subclass of its own.

F-logic symbol $A$ is a superset of function and predicate symbols appearing in $P$. Supposes $L^{F}_P$ express F-logic language based on $\Sigma_P$. Suppose $\Sigma_P$ contains at least a 0-ary function symbols or unique 0-ary predicate symbol.

$B_H$, Herbrand foundation of $L^{F}_P$, is basic set of atomic formula and molecular of $L^{F}_P$. The subset of $B_H$ is called the Herbrand explanation. A basic logic program $P$ expressed as gr(P), is union set of all possible basic instances of $P$. Each rule $r \in P$ is obtained in $\Sigma_P$ through substitution of an elemental term by rule $r$.

Supposes $P$ is a procedure. $M$, a Herbrand interpretation of $P$, is a model of $P$. If for each rule, $r \in \text{gr}(P)$, $B'(r) \subseteq M$ means $H(r) \cap M \neq \emptyset$. A Herbrand model $M$ is minimal if and only if for every model has $M' \subseteq M$, $M'=M$.

According to [8], a simple logic program $P$ of a Herbrand interpretation $M$, is expressed as $P_M^{h}$. It is obtained from the $\text{gr}(P)$ by removing the following:

(i) each rule $r$ with $B \cap (r) \cap M \neq \emptyset$; and

(ii) delete the non-c (not c) from each body of the remaining rules $r$ with $c \in B'(r)$. If $M$ is a minimal Herbrand model of $P_M^{h}$, then $M$ is a stable model of $P$.

If $P$ is a positive logic program, then for each rule, the corresponding Horn F-logic theory $\Phi$ replace arrow $\leftarrow$ with $\cap$, and replace each "", "" with $\land$ in the rule body. [8]

3.2. F-logic inference of metamodel

\textbf{Theorem 1} Let $P$ be a positive logic program and $\Phi$ is a corresponding Horn F-logic theory, then $P$ has a corresponding model $M$, and for each basic atom or molecule $a, a \in M$ if and only if $\Phi|=a$.

Definition 2 (flat item) a basic flat item is a constant or a variable. An item is flat if and only if it is a basic flat item or a function item $f@(t_1, ... , t_n)$, where $f$ is a $n$-function symbol and $t_1, ... , t_n$ is basic flat item. An atom $P@(t_1, ... , t_n)$ is flat if and only if all $t_1, ... , t_n$ items are flat. A text is flat if and only if its atoms are flat; a clause is flat if and only if it is all the text is flat.

For example, if $c$ is a constant and $a$ is a 0-ary function symbol, then $P@(x, c, a), P@(f@x; f@c)$ and $f@x \square g@c$ is flat, but $P@(f@a, f@c)$ and $f@x \square g@c$ is not flat.

Random clause set may be transformed into flat clause set by “extracting” the clauses unneeded. It requires applying the following conversion rules to a given set of clauses to achieve as much as possible.

(1) If $s_i$ is not a basic flat item, where $x$ is a new variable, then use

$P@[t_1, ... , f@(s_1, ... , x, ... , s_m), ... , t_n] \rightarrow$ ( ) OR H$\leftarrow B$ , $x \square s_i$

to replace a clause like below.

$P@[t_1, ... , f@(s_1, ... , s_i, ... , s_m), ... , t_n] \rightarrow$ ( ) OR H$\leftarrow B$.

(2) If $s_i$ is not a basic flat item, where $x$ is a new variable, then use

H$\leftarrow B$ , $P@[t_1, ... , f@(s_1, ... , x, ... , s_m), ... , t_n] \rightarrow$ ( ) , $x \square s_i$

to replace a clause like below.

H$\leftarrow B$ , $P@[t_1, ... , f@(s_1, ... , x, ... , s_m), ... , t_n] \rightarrow$ ( ) .

Obviously, for any set of clauses, use a uniquely determined set of a flat clause (equivalent to renaming of variables and the sort of body atoms) to terminate its flattening.

\textbf{Definition 1} (Equivalence transformation) Let $P$ be a $\Sigma$-clause set. The $P$ equal transformation expression is $P^{eq}$. It is a clause set obtained by flattening $P$ and by adding the following clause:

$\leftarrow c \square d$ For any two different $\Sigma$-constants $c$ and $d$

$x \square x \leftarrow$
x = y → x = y
x = z → x = y, y = z
P[@(x1, ..., y, ..., xn)]→( ) ← P[@(x1, ..., y, ..., xn)]→( ) , y = x

Each n-predicate symbol P in Σ is different from d, and all i are 1 ≤ i ≤ n

When axiom c = d is neglected, the only difference between equal transformation and the axiom equal processing lies in using function substitute the axiom. For example, axioms such as A, for each n-ary Σ-function symbols and all i = 1, ..., n, these axioms can be omitted or abandoned because of being flattened.

Theorem 2 (correctness and completeness of equivalent transformation) Let P be a set of clauses, then P is UNA-E-satisfied if and only if P^# is satisfiable.

Proof: the proof of necessary conditions is not difficult. For the limited space this omitted its proof. Note that "extract" sub-item to keep E-satisfied. The essential condition (accuracy) is easy to prove. There are the similar explanations in [8-9], and it is easy to infer.

Definition 2 (Block transformation) Let P be a flat Σ-clause. Block transformation is expressed as P^#. It can be obtained by application of the following four steps:

Domain restraint: Replace each rule H =⇒ B of P with rule H =⇒ B, dom@x1 , ..., dom@xn, where \{x1, ..., xn\} is variable set occurs in H =⇒ B. There is k ≥ 0.

(2) Extracts the function item: replaces each clause like
H =⇒ B , \ P[@(t1, ..., f@(s1, ..., sm)) , ..., tm)→( ) \hspace{1cm} (1)

as far as possible in the result sub-clause, where f is a non-0 Σ-function symbol and has
H =⇒ B , \ P[@(t1, ..., x, ..., tm)→( ) , f@(s1, ..., sm)→refx \hspace{1cm} (2)

And finally to this step with clause:
\hspace{1cm} x→refy ← \hspace{1cm} (3)
\hspace{1cm} x→refy ← x→y \hspace{1cm} (4)

(3) limited domain search: add the following clause to the result clause set, for each variable c, each n-ary function symbol f and all i=1, ..., n, and for each m-ary Σ-predicate symbols P and all j=1, ..., m,
\hspace{1cm} dom@c ← \hspace{1cm} (5)
\hspace{1cm} dom@xj ← dom@(f@(x1, ..., xj)) \hspace{1cm} (6)
\hspace{1cm} dom@xj ← dom_candidate(@(f@(x1, ..., xj)) \hspace{1cm} (7)
\hspace{1cm} dom_candidate(@(f@(x1, ..., xj))) ← dom@xj, ..., dom@xn \hspace{1cm} (8)
\hspace{1cm} f@(x1, ..., xj) → subx1 ∨ ... ∨ f@(x1, ..., xj) → subxn ∨ dom(@(f@(x1, ..., xj)) \hspace{1cm} (9)
\hspace{1cm} x→c ← x→subc \hspace{1cm} (10)
\hspace{1cm} x→f@(x1, ..., xj) ∨ x→subx1 ∨ ... ∨ x→subxn ← x→subf@(x1, ..., xj) \hspace{1cm} (11)
P[@(x1, ..., xj, y, xj+1, ..., xn) ← xj→y , P[@(x1, ..., xn)] \hspace{1cm} (12)
(4) uniqueness of $\rightarrow$: add the following clause to the result clause set, for each $n$-ary $\Sigma$-function symbol $f$ and all $i = 1, \ldots, n$:  

\[\neg x \rightarrow y, \; \neg x \rightarrow z, \; y \neq z\]  

(14)  

To random two different $\Sigma$-constant $c$ and $d$  

\[c \neq d\]  

(16)  

The $f@((t_1, \ldots, t_n))$ is mapped to its child item to make $f@((t_1, \ldots, t_n))$ to be a new domain element. It is better termination behavior. After being tested, it is proved to be reliable. It set the model generation process to achieve this goal.

Lemma 1 Let $T$ be a set of flattened clauses. If $P_{bl}$ is not satisfiable, then $P$ is not satisfiable.

Theorem 3 (correctness and completeness of block transformation) Let $P$ be a set of flat clauses, then $P$ is satisfiable if and only if $P_{bl}$ is satisfiable. 

Proof: the proof of necessity can be deduced from Lemma 1.  

Proof of sufficiency: Let $I_{bl}$ be a Herbrand $\Sigma$-model of $P_{bl}$. We proved that $I_{bl}$ determines a $\Sigma$-model $I$ (may be non-Herbrand) of $P$. The proof has two steps: First step we construct domain, constant, function symbol and predicate symbol explanation of $I$. Second step we proved that $I$ is a model of $P$. For limited space, concrete proof is omitted in this process.

Theorem 4 Let $P$ be a positive logic program, $\mathcal{K} P$ be the corresponding Horn F-logic theory, and then $P$ has a stable model $M$. For each basic atom or molecule $\alpha$, it has $\alpha \in M$ if and only if $\mathcal{K} P \models \alpha$.

Theorem 5 Let $\mathcal{K} \subseteq L$ be a F-logic theorem, and let $\kappa \in L$ be a formula, then $\mathcal{K} \models \kappa$ if and only if $\delta(\kappa) \models \Gamma$.  

Proof: it can be immediately driven from Theorem 3, and from the fact checking features $\mathcal{K} \models \kappa$ can deduce the satisfiability by formula ($\mathcal{K} \models \kappa$), and $\alpha \models \Gamma$ (with similar characteristics in the F-logic).

Theorem 6 mapping:  

$\Gamma$: \{F-formula$\} \rightarrow \{\text{well-formed formula of predicate computation}$\},  

$\Phi$: \{F-structure$\} \rightarrow \{\text{semantic structure of predicate computation}$\}.  

For any structure $M$ and any F-formula $\psi$, $M \models^f \psi$ if and only if $\phi(M) \models^c \Gamma(\psi)$. Here “$\models^f$” and “$\models^c$” express computing features of predicate logic in F-logic.  

F-OWL and Flora-2 is OWL inference engines based on F-logic, which use the above theorem to complete reasoning tasks of the model. Its inference will observe information formal semantic during the coding process in OWL, thus discovers the inconsistency in the OWL-S data, simultaneously reasoning new information from known information.

4. Conclusions

This paper describes the conversion from the ontology of semantic Web services language OWL-S to ontology expressed by F-logic. To ontology language based on predicate, this conversion keeps the properties. This paper not only studies formalization of the dynamic evolved requirement of OWL-S process model, but also studies formalized concept, axiom, restraint and uniform frame semantics of
the OWL-S static model. We portray the OWL-S static model with the F-logic formalization method, design an F-logic model to confirm the OWL-S ontology, and express its global property and frame as F-logic class and formula. It also gives the validation of axioms, inference rules, and theorems to verify the consistency and satisfiability of the model, to analyze the construction of OWL-S model type, and reasoning the tasks supported by OWL-S. This model and the method can use the first-order logical simulate to confirm some global property of the OWL-S service system. The proposed model and method can use model checking based on first-order logical to verify some global property of OWL-S service system. Next step we will use the result above to develop a composite automation verification tool to verify OWL-S requirement.

4. Acknowledgments

This work was supported in part by a grant from Shenzhen Science and Technology Program (JC201006020820A and JC201006020791A), Team of Scientific and Technological Innovation of Shenzhen Institute of Information Technology (CXTD2-002), Natural Science Foundation of Guangdong Province, China (S2011040000672), Guangdong Provincial Education and Science Project of the 11th "five-year plan"(2010jyk411), Research Fund of Shenzhen Institute of Information Technology (YB201008, JY2010003).

5. References