8th International Semantic Web Conference (ISWC 2009)

25-29 October 2009, Virginia, USA

SWESE 2009 :

5th International Workshop on Semantic Web Enabled Software Engineering

ISWC 2009 Workshop
26 October 2009, Virginia, USA

Edited by: *Elisa F. Kendall*
Jeff Z. Pan
Marwan Sabbouh
Ljiljana Stojanovic
Yuting Zhao

Workshop Organisers

Elisa F. Kendall, Sandpiper Software, Inc. US.
Jeff Z. Pan, University of Aberdeen, UK.
Marwan Sabbouh, MITRE Corporation. US.
Ljiljana Stojanovic, FZI, Germany.
Yuting Zhao, University of Aberdeen, UK.

Program Committee

Sponsorshop

The workshop is being held in cooperation with a prominent network of excellence and is meant to act as a focal point for joint interests and future collaborations. SWESE2009 is sponsored by the MOST project (EU ICT 2008-216691).

MOSTPROJECT

Introduction

The advent of the World Wide Web has led many corporations to web-enable their business applications and to the adoption of web service standards in middleware platforms. Marking a turning point in the evolution of the Web, the Semantic Web is expected to provide more benefits to software engineering. Over the past five years there have been a number of attempts to bring together languages and tools, such as the Unified Modelling Language (UML), originally developed for Software Engineering, with Semantic Web languages such as RDF and OWL. The Semantic Web Best Practice and Deployment Working Group (SWBPD) in W3C included a Software Engineering Task Force (SETF) to investigate potential benefits. A related international standardisation activity is OMG's Ontology Definition Metamodel (ODM), which was formally adopted in October 2006, and finalized in December 2008. Another interesting question is how to use ontology to improve guidance and traceability in software development.

It has been argued that the advantages of Semantic Web Technologies in software engineering include reusability and extensibility of data models, improvements in data quality, and discovery and automated execution of workflows. According to SETF's note A Semantic Web Primer for Object-Oriented Software Developers, the Semantic Web can serve as a platform on which domain models can be created, shared and reused.

However, are there other potential benefits in the use of Semantic Web concepts in the field of Software Engineering? Could the Web-based, semantically rich formality of OWL be combined with emerging model driven development tools such as the Eclipse Modelling Framework to provide some badly needed improvements in both the process and product of software development activities? What is it about the amalgamation of OWL, UML and MDA methodology that could make a difference? Certainly, there appear to be a number of strong arguments in favour of this approach but consensus on the best way forward, if there is indeed a way forward at all has not yet formed. This workshop seeks to build on prior events that have begun to explore and evaluate this important area.

Table of Contents

Validating Process Refinement with Ontologies*

Yuan Ren[1], Gerd Groener[2], Jens Lemcke[3], Tirdad Rahmani[3], Andreas Friesen[3], Yuting Zhao[1], Jeff Z. Pan[1] and Steffen Staab[2]

[1]University of Aberdeen, [2]University of Koblenz-Landau, [3]SAP AG

Abstract. A crucial task in process management is the validation of process refinements. A process refinement is a process description in a more fine-grained representation. The refinement is with respect to either an abstract model or a component's principle behaviour model. We define process refinement based on the execution set semantics. Predecessor and successor relations of the activities are described in an ontology in which the refinement can be validated by concept satisfiability checking.

1 Introduction

With the growing interest about applying semantic web technologies on business process modelling, many frameworks and ontological models have been proposed to facilitate a more unified semantic representation [5, 6].

In model-driven software development, process models are usually created and refined on different levels of abstraction. A generic process describes the core functionality of an application. A refinement is a transformation of a process into a more specific process description which is developed for a more concrete application and based on more detailed process behaviour knowledge. In this procedure, the refined process should refer to the intended behaviour of the abstract process and satisfies behaviour constraints. To check and ensure the consistency of refinement becomes a crucial issue in process management. Currently, such consistency check is mainly done manually and few methods have been investigated to help automation. Hence the validation is error-prone, time-consuming and increases the costs during the development cycle.

In this paper, we use execution set semantics to describe two types of process refinements and present an ontological approach to represent and check them. We first apply topological transformations to reduce the refitment checking w.r.t. execution set semantics into checking of predecessors and successors of process elements. Then we encode process models into OWL DL ontologies. Finally we show that the refinement checking on the process models can be accomplished by concept ussatisfiability checking in the ontology. We implemented our approach and conducted performance evaluation on a set of randomly generated process models. Experiment results showed that, 80% of the refinement validation tasks

* This work has been supported by the European Project Marrying Ontologies and Software Technologies (EU ICT 2008-216691).

can be performed within 1s, which is significantly faster than manually consistency checking and the correctness of the validation is guaranteed.

The rest of the paper is organised as follows: in Sec.2 we define the problem of process refinement with its graphical syntax, semantics and mathematical foundation. The representation and validation of processes with the corresponding execution constraints is demonstrated in Sec.3. In Sec.4 we present the evaluations and in Sec.5 we review related works and conclude the paper.

2 Preliminary

In this section, we introduce preliminary knowledge about process models, process refinement w.r.t. execution set semantics and DL-based ontologies.

Syntax of Process Models A process model—or short: process—is a non-simple directed graph $\mathrm{P} = \langle \mathrm{E}, \mathrm{V} \rangle$ without multiple edges between two vertices. As a graphical representation, we use the business process modelling notation (BPMN: `http://www.bpmn.org/`) due to its wide industry adoption. However, we consider a normal form of process models for the sake of this paper as opposed to the full set of partly redundant constructs in BPMN.

In our definition, vertices (V) include activities, gateways ($\mathrm{A}, \mathrm{G} \subseteq \mathrm{V}$), and the specific vertices start and end event ($v_0, v_{\text{end}} \in \mathrm{V}$). Fig. 1a shows a BPMN diagram which consists of two activities between the start and end events.

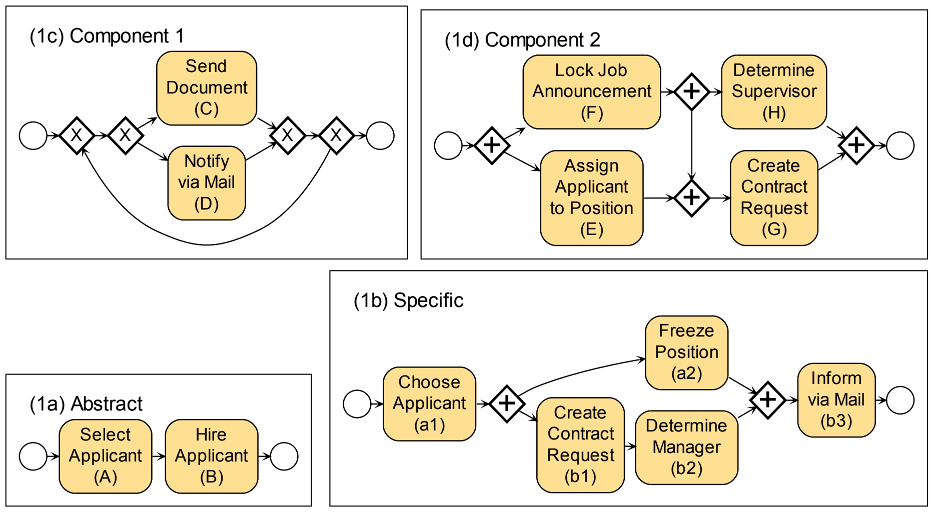


Fig. 1. Wrong process refinement

A gateway is either opening or closing ($\mathrm{G}^{\mathrm{O}}, \mathrm{G}^{\mathrm{C}} \subseteq \mathrm{G}$), and either exclusive or parallel ($\mathrm{G}^{\otimes}, \mathrm{G}^{\oplus} \subseteq \mathrm{G}$). The process models (c) and (d) in Fig. 1 contain exclusive and parallel gateways, respectively. We call a process *normal* if it does not contain parallel gateways ($\mathrm{G}^{\oplus} = \emptyset$)—as, for example, process model (c).

The edge set (E) is a binary relation on V. We define the predecessor and the successor functions of each $v_1 \in \mathrm{V}$ as follows: $\mathrm{pre}(v_1) := \{v_2 \in \mathrm{V} \mid (v_2, v_1) \in \mathrm{E}\}$, $\mathrm{suc}(v_1) := \{v_3 \in \mathrm{V} \mid (v_1, v_3) \in \mathrm{E}\}$. The start (end) event has no predecessor (successor): $|\mathrm{pre}(v_0)| = |\mathrm{suc}(v_{\mathrm{end}})| = 0$ and exactly one successor (predecessor): $|\mathrm{suc}(v_0)| = |\mathrm{pre}(v_{\mathrm{end}})| = 1$. Each open gateway $o \in \mathrm{G}^{\mathrm{O}}$ (close gateway $c \in \mathrm{G}^{\mathrm{C}}$) has exactly one predecessor (successor): $|\mathrm{pre}(o)| = |\mathrm{suc}(c)| = 1$. Each activity $a \in \mathrm{A}$ has exactly one predecessor and successor: $|\mathrm{pre}(a)| = |\mathrm{suc}(a)| = 1$. We can then construct gateway-free predecessor and successor sets as follows:

$$PS(v_1) := \{v_2 \in \mathrm{A} \mid v_2 \in \mathrm{pre}(v_1) \textit{ or } \exists u \in \mathrm{G} \textit{ s.t. } u \in \mathrm{pre}(v_1) \textit{ and } v_2 \in PS(u)\}$$
$$SS(v_1) := \{v_3 \in \mathrm{A} \mid v_3 \in \mathrm{suc}(v_1) \textit{ or } \exists u \in \mathrm{G} \textit{ s.t. } u \in \mathrm{suc}(v_1) \textit{ and } v_3 \in SS(u)\}$$

These two definitions make gateways "transparent" to ordering relations. For example in Fig.1b, $SS(a1) = \{b1, a2\}$, in Fig.1c, $PS(C) = \{C, D\}$.

Execution Set Semantics of Process Models We define the semantics of a process model using the execution set semantics [18]. An execution is a *proper* sequence of activities ($a_i \in \mathrm{A}$): $[a_1 a_2 \ldots a_n]$. A proper sequence is obtained by simulating token flow through a process model. A token is associated to exactly one vertex or edge. Initially, there is exactly one token, associated to the start event. Tokens can be created and consumed following the rules below. Whenever a token is created in an activity, the activity is appended to the sequence. Exactly one of the following actions is performed at a time:

– For creating a token in an activity or in the end event $v_1 \in \mathrm{A} \cup \{v_{\mathrm{end}}\}$, exactly one token must be consumed from the incoming edge $(v_2, v_1) \in \mathrm{E}$.
– Exactly one token must be removed from an activity or from the start event $v_1 \in \mathrm{A} \cup \{v_0\}$ in order to create one token in the leaving edge $(v_1, v_2) \in \mathrm{E}$.

– For creating a token in a parallel close gateway $g \in (\mathrm{G}^{\oplus} \cap \mathrm{G}^{\mathrm{C}})$, exactly one token must be consumed from every incoming edge $(v, g) \in \mathrm{E}$.
– For creating a token in an exclusive close gateway $g \in (\mathrm{G}^{\otimes} \cap \mathrm{G}^{\mathrm{C}})$, exactly one token must be consumed from exactly one incoming edge $(v, g) \in \mathrm{E}$.
– Exactly one token must be removed from a close gateway $g \in \mathrm{G}^{\mathrm{C}}$ in order to create one token in the leaving edge $(g, v) \in \mathrm{E}$.

– For creating a token in an open gateway $g \in \mathrm{G}^{\mathrm{O}}$, exactly one token must be consumed from the incoming edge $(v, g) \in \mathrm{E}$.
– Exactly one token must be removed from a parallel open gateway $g \in (\mathrm{G}^{\oplus} \cap \mathrm{G}^{\mathrm{O}})$ in order to create one token in each leaving edge $(g, v) \in \mathrm{E}$.
– Exactly one token must be removed from an exclusive open gateway $g \in (\mathrm{G}^{\otimes} \cap \mathrm{G}^{\mathrm{O}})$ in order to create one token in exactly one leaving edge $(g, v) \in \mathrm{E}$.

If none of the above actions can be performed, simulation has ended. The result is a proper sequence of activities—an execution. It is to be noted that each execution is finite. However, there may be an infinite number of executions for a process model. The execution set of a process model P, denoted by ES_P, is the (possibly infinite) set of all proper sequences of the process model.

For example, $ES_{1\mathrm{a}}$ for process (a) in Fig. 1 is $\{[\mathsf{AB}]\}$: first A, then B (for brevity, we refer to an activity by its short name, which appears in the diagrams in parenthesis). Process (b) contains parallel gateways (⊕) to express that some activities can be performed in any order: $ES_{1\mathrm{b}} = \{[\mathsf{a_1a_2b_1b_2b_3}], [\mathsf{a_1b_1a_2b_2b_3}], [\mathsf{a_1b_1b_2a_2b_3}]\}$. Exclusive gateways (⊗) are used in process (c) both to choose from the two activities and to form a loop: $ES_{1\mathrm{c}} = \{[\mathsf{C}], [\mathsf{D}], [\mathsf{CC}], [\mathsf{CD}], [\mathsf{DC}], [\mathsf{DD}], \ldots\}$. Process (d) shows that gateways can also occur in a non-block-wise manner: $ES_{1\mathrm{d}} = \{[\mathsf{EFGH}], [\mathsf{EFHG}], [\mathsf{FHEG}], [\mathsf{FEGH}], [\mathsf{FEHG}]\}$.

Correct Process Refinement For refinement validation we have to distinguish between horizontal and vertical refinement. A horizontal refinement is a transformation from an abstract to a more specific model which contains the decomposition of activities. A vertical refinement is a transformation from a principle behaviour model of a component to a concrete process model for an application. The validation have to account for both refinements.

Fig. 1 shows a refinement horizontally from abstract to specific while vertically complying with the components' principle behaviour. In our example scenario, Fig. 1a is drawn by a line of business manager to sketch a new hiring process. Fig. 1b is drawn by a process architect who incrementally implements the sketched process. Fig. 1c and d are the principle behaviour models of different components.

To facilitate horizontal validation, the process architect has to declare which activities of Fig. 1b implement which activity of Fig. 1a: $\mathrm{hori}(\mathsf{a_1}) = \mathrm{hori}(\mathsf{a_2}) = \mathsf{A}$, $\mathrm{hori}(\mathsf{b_1}) = \mathrm{hori}(\mathsf{b_2}) = \mathrm{hori}(\mathsf{b_3}) = \mathsf{B}$. For vertical validation, the process architect needs to link activities of Fig. 1b to service endpoints given in Fig. 1c and d: $\mathrm{vert}(\mathsf{a_1}) = \mathsf{E}$, $\mathrm{vert}(\mathsf{a_2}) = \mathsf{F}$, $\mathrm{vert}(\mathsf{b_1}) = \mathsf{G}$, $\mathrm{vert}(\mathsf{b_2}) = \mathsf{H}$, $\mathrm{vert}(\mathsf{b_3}) = \mathsf{D}$.

Correct horizontal refinement. We say that a process Q is a correct *horizontal* refinement of a process P if $ES_Q \subseteq ES_P$ after the following transformations.

1. **Renaming.** Replace all activities in each execution of ES_Q by their originators (function hori()). Renaming the execution set $\{[\mathsf{a_1a_2b_1b_2b_3}], [\mathsf{a_1b_1a_2b_2b_3}], [\mathsf{a_1b_1b_2a_2b_3}]\}$ of Fig. 1b yields $\{[\mathsf{AABBB}], [\mathsf{ABABB}], [\mathsf{ABBAB}]\}$.
2. **Decomposition.** Replace all sequences of equal activities by a single activity in each execution of ES_Q. For Fig. 1b this yields $\{[\mathsf{AB}], [\mathsf{ABAB}]\}$.

As $\{[\mathsf{AB}]\} \not\supseteq \{[\mathsf{AB}], [\mathsf{ABAB}]\}$, Fig. 1b is a wrong horizontal refinement of Fig. 1a. The cause is the potentially inverted order of AB by $\mathsf{b_1a_2}$ or $\mathsf{b_2a_2}$ in Fig. 1b.

Correct vertical refinement. We say that a process Q is a correct *vertical* refinement of a process P if $ES_Q \subseteq ES_P$ after the following transformations.

1. **Renaming.** Replace all activities in each execution of ES_Q by their grounds (function vert()). Renaming the execution set $\{[\mathsf{a_1a_2b_1b_2b_3}], [\mathsf{a_1b_1a_2b_2b_3}], [\mathsf{a_1b_1b_2a_2b_3}]\}$ of Fig. 1b yields $\{[\mathsf{EFGHD}], [\mathsf{EGFHD}], [\mathsf{EGHFD}]\}$.

2. **Reduction.** Remove all activities in each execution of ES_Q that do not appear in P. For our example, reduction with respect to Fig. 1c yields $\{[\mathsf{D}]\}$. Reduction with respect to Fig. 1d yields $\{[\mathsf{EFGH}], [\mathsf{EGFH}], [\mathsf{EGHF}]\}$.

Fig. 1b is a correct vertical refinement of Fig. 1c because $\{[\mathsf{C}], [\mathsf{D}], [\mathsf{CC}], [\mathsf{CD}], [\mathsf{DC}], [\mathsf{DD}], \ldots\} \supseteq \{[\mathsf{D}]\}$ and a wrong vertical refinement of Fig. 1d because $\{[\mathsf{EFGH}], [\mathsf{EFHG}], [\mathsf{FHEG}], [\mathsf{FEGH}], [\mathsf{FEHG}]\} \not\supseteq \{[\mathsf{EFGH}], [\mathsf{EGFH}], [\mathsf{EGHF}]\}$. The cause for the wrong refinement is the potentially inverted execution of FG by $\mathsf{b_1a_2}$ in Fig. 1b.

As enumerating the execution sets for validation is infeasible, our solution works with descriptions in ontology instead of using the execution sets themselves.

Description Logics and Ontologies DL-based ontologies have been widely applied as knowledge formalism for the semantic web. An ontology usually consists of a terminology box (TBox) and an assertion box (ABox). In TBox the domain is described by concepts and roles with DL constructs. In this paper, we use DL $\mathcal{ALC}$. Its concepts are inductively defined by following constructs:

$$\top, \bot, A, \neg C, C \sqcap D, C \sqcup D, \exists r.C, \forall r.D$$

where $\top$ is the super concept of all concepts; $\bot$ means nothing; A is an atomic concept; C and D are general concepts and r is an atomic role. In DL, the subsumption between two concepts C and D is depicted as $C \sqsubseteq D$. If two concepts mutually subsume each other, they are equivalent, depicted by $C \equiv D$. When a concept can not be instantiated in any model, i.e., $C \sqsubseteq \bot$, it is unsatisfiable. Two concepts are disjoint if $C \sqsubseteq \neg D$. In this paper we write $Disjoint(C_1, C_2, \ldots, C_n)$ to denote that all these concepts disjoint with one another.

3 Validation with Ontologies

In this section, we present our solution of validating process refinement in detail. We first eliminate all the parallel gateways in a process, then translate such a process into ontologies based on the predecessor and successor sets of activities, finally we show that the refinement checking can be reduced to concept unsatisfiability checking

3.1 Process Transformation

As we can see from ES_{1c}, the execution ordering relations between successors of some $g \in G^O$ are implicit in the original process. For example, $b1$ and $a2$ does not have any explicit edge, the semantics of parallel gateway still implies that $b1a2$ or $a2b1$ must appear in some execution. In order to make such relations explicit, we eliminate all the parallel gateways while retain the execution set. Our strategy is to generate exclusive gateways to represent the executions.

Given a process P, its normal $n(P)$ can be obtained as follows:

1. Repeatedly replace each penning-parallel gateway g by an opening-exclusive gateway e. For each $v \in suc(e)$, construct a new penning-parallel gateway g' with $pre(g') = v$, $suc(g') = suc(v) \cup suc(e) \setminus \{v\}$ and then make $suc(v) = g'$.
2. Remove all the edges from an opening- to a closing-parallel gateway.
3. If an opening-gateway has only one successor, remove the gateway
4. If an closing-gateway has only one predecessor, remove the gateway

In step 1 direct successors of parallel gateways are "pulled" out of the gateway. Here a loop block is considered as a single successor. In this procedure, a parallel gateway with n successors is transformed into n parallel gateways with n successors but one of the successive sequence is shortened by one successor. Due to the finite length of these sequences, this replacement always terminates. Step 2 then reduces the number of successors for these remaining parallel gateways by removing "empty" edges. Step 3 and 4 finally remove the gateways. When a gateway is removed, its predecessors and successors should be directly connected.

It's obvious that this normalisation will always result in a normal process. An example of normalisation of Fig.1b and Fig.1d can be seen in Fig.2.

The size of $n(P)$ can be exponentially large w.r.t. P in worst case: suppose P contains only a pair of parallel gateways with n sequences of one activity, then $n(P)$ will contains a pair of exclusive gateways with $n!$ sequences of n activities.

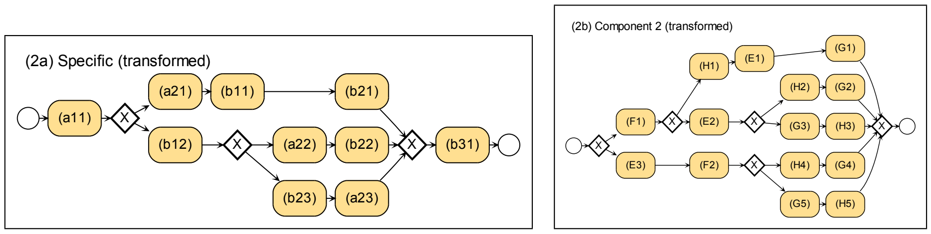


Fig. 2. Transformation to execution diagrams for Fig. 1b and d

In normalisation, some activities will be duplicated in the process. These duplications have different predecessors (successors). We distinguish them by an additional numerical subscript. We depict such a transformed process $n(P)$ with distinguished activities by $P^\star$. Obviously, $ES_{P^\star}$ is the same as $ES_{n(P)}$ after replacing all the distinguished activities by their original names. Thus, relation between two execution sets can be characterised by the following theorem:

Theorem 1. *Given two processes $P = \langle \mathrm{E}_P, \mathrm{V}_P \rangle$ and $Q = \langle \mathrm{E}_Q, \mathrm{V}_Q \rangle$, $ES_Q \subseteq ES_P$* iff *$\forall a_i \in \mathrm{A}_{Q^\star}$, there exists some $a_j \in \mathrm{A}_{P^\star}$ such that $PS_{Q^\star}(a_i) \subseteq PS_{P^\star}(a_j)$ and $SS_{Q^\star}(a_i) \subseteq SS_{P^\star}(a_j)$.*

Proof. (1) For the $\rightarrow$ direction the lhs $ES_Q \subseteq ES_P$ holds. We demonstrate the subsumption for PS. For an arbitrary activity $a_i \in \mathrm{A}_{Q^\star}$, the activity $a_i{}'$ is the corresponding activity before normalisation (i.e. without additional subscripts).

The activity ${a_j}'$ is the originator or ground activity of ${a_i}'$ in P after renaming and $a_j \in \mathrm{A}_{P^\star}$ is the the corresponding activity after normalization of P. From the prerequisite it directly follows that the predecessors of $a_i \in \mathrm{A}_{Q^\star}$ are predecessors of a_j in $Q^\star$. The subsumption of the successor set is demonstrated likewise. (2) To prove the other direction we assume that the rhs holds. Consider an execution $s \in ES_Q$ we demonstrate that $s \in ES_P$. For each activity ${a_i}'$ of an arbitrary execution $s \in ES_Q$ the corresponding activity $a_i \in \mathrm{A}_{Q^\star}$ is received after normalization. From the rhs it follows that there exists an activity $a_j \in \mathrm{A}_{P^\star}$ so that each predecessor of a_i is also a predecessor of a_j in $P^\star$ and likewise for the successors of a_i. After activity renaming and demonstrating for all activities of each execution of ES_Q the inclusion of the lhs follows.

Therefore, we reduce the process refinement w.r.t. execution set semantics into the subsumption checking of finite predecessor and successor sets. We then show that the transformation operations of execution sets can be equivalently performed on the its process model and the predecessor and successor sets:

– **Reduction** on the process diagrams has the same effect on the execution sets. That means, given a component model P and a process model Q, if we reduce Q into Q' by removing all the activities that do not appear in P, and connect their predecessors and successors directly, the resulting $ES_{Q'}$ will be the same as the reduced ES_Q with respect to P.
– **Renaming** can also be directly performed on the process diagram, i.e. $ES_P[a \rightarrow A] = ES_{P[a \rightarrow A]}$. Thus, the renaming can be performed on the predecessor and successor sets as well, i.e. $PS_P(x)[a \rightarrow A] = PS_{P[a \rightarrow A]}(x)$ $(SS_P(x)[a \rightarrow A] = SS_{P[a \rightarrow A]}(x))$.
– **Decomposition** can be done on the predecessor and successor sets as well. Theorem 1 shows that the subsumption of execution sets can be reduced to subsumption of predecessor and successor sets. Decomposition means, an activity x can go from not only predecessors of x, but also another appearance of x, and can go to not only successors of x, but also another appearance of x. Any sequence of x in the execution will be decomposed.

Thus, for horizontal refinement, we can first obtain the predecessor and successor sets of activities, and then perform the **Renaming** and **Decomposition** on these sets, and then check the validity. For vertical refinement, we can first perform the **Reduction** on processes, then obtain the predecessor and successor sets and perform the **Renaming** on these sets, and finally check the validity.

In this paper, we perform **Reduction** directly on the a process P and obtain the predecessor and successor sets from $P^\star$, then encode **Renaming** and **Decomposition** into ontology and check the validity with reasoning.

3.2 Refinement representation

In this section we represent the predecessor and successor sets of activities with ontologies. In such ontologies, activities are represented by concepts. The predecessors/successors relations are described by two roles *from* and *to*, respectively.

On instance level, these two roles should be inverse role of each other. However this is not necessary in our solution. Composition of activities in horizontal refinement is described by role *compose*. Grounding of activities in vertical refinement is described by role *groundedTo*. To facilitate the ontology construction, four operators are defined for pre- and post- refinement process:

Definition 1. *: Given S a predecessors or successors set, we define four operators for translations as follows:*
Pre-refinement-from operator $\boldsymbol{Pr}_{from}(S) = \forall from. \bigsqcup_{x \in S} x$
Pre-refinement-to operator $\boldsymbol{Pr}_{to}(S) = \forall to. \bigsqcup_{y \in S} y$
Post-refinement-from operator $\boldsymbol{Ps}_{from}(S) = \bigsqcap_{x \in S} \exists from.x$
Post-refinement-to operator $\boldsymbol{Ps}_{to}(S) = \bigsqcap_{y \in S} \exists to.y$

The effect of the above operators in refinement checking can be characterised by the following theorem:

Theorem 2. $PS_Q(a) \subseteq PS_P(a)$ iff
Disjoint$(x|x \in \mathrm{A}_P \cup \mathrm{A}_Q)$ *infers that* $\boldsymbol{Pr}_{from}(PS_P(a)) \sqcap \boldsymbol{Ps}_{from}(PS_Q(a))$ *is satisfiable.*
$SS_Q(a) \subseteq SS_P(a)$ iff
Disjoint$(x|x \in \mathrm{A}_P \cup \mathrm{A}_Q)$ *infers that* $\boldsymbol{Pr}_{to}(SS_P(a)) \sqcap \boldsymbol{Ps}_{to}(SS_Q(a))$ *is satisfiable.*

For sake of a shorter presentation, we only prove the first part of the theorem. The proof for the second part is appropriate to the first part.

Proof. (1) We demonstrate the $\rightarrow$ direction with a proof by contraposition. The disjointness of activities holds. Supposed the rhs is unsatisfiable, i.e. $\mathbf{Pr}_{from}(PS_P(a)) \sqcap \mathbf{Ps}_{from}(PS_Q(a))$ is unsatisfiable. Obviously, both concept definitions on its own are satisfiable, since $\mathbf{Pr}_{from}(PS_P(a))$ is just a definition with one all-quantified role followed by a union of (disjoint) concepts. The concept definition behind this expression is $\forall from. \bigsqcup_{x \in PS_P(a)} x$ which restricts the range of $from$ to all concepts (activities) of $PS_P(a)$. $\mathbf{Ps}_{from}(PS_Q(a))$ is a concept intersection which only consists of existential quantifiers and the same $from$ role. This definition is also satisfiable. Therefore the unsatisfiability is caused by the intersection of both definitions. In $\mathbf{Ps}_{from}(PS_Q(a))$ the same role $from$ is used and the range is restricted by $\mathbf{Pr}_{from}(PS_P(a))$. Therefore the contradiction is caused by one activity $b \in PS_Q(a)$ which is not in $PS_P(a)$, but this is a contradiction to the precondition $PS_Q(a) \subseteq PS_P(a)$.
(2) The $\leftarrow$ direction can be proved similarly by contraposition.

Now we can represent horizontal and vertical refinements by ontologies:

Horizontal Refinement For conciseness of presentation, we always have a pre-refinement process P and a post-refinement process Q and we refine one activity z of P in this step. z may have multiple appearances z_j in $P^\star$. For each z_j we define $component\_z_j \equiv \exists compose.z_j$. Simultaneous refinement of multiple activities can be done in a similar manner of single refinement. Then we construct an ontology $\mathcal{O}_{P \rightarrow Q}$ with following axioms:

1. for each activity $a_i \in \mathrm{A}_{Q^\star}$ and $hori(a) = z$
   $a_i \sqsubseteq \bigsqcup \exists compose.z_j$
   These axioms represent the composition of activities with concept subsumption, which realise **Renaming** in horizontal refinement. For example, $b_{31} \sqsubseteq \exists compose.B$ and $a_{11} \sqsubseteq \exists compose.A$.
2. for each $a_i \in \mathrm{A}_{Q^\star}$ where a is not refined from z
   $a_i \sqsubseteq \mathbf{Pr}_{from}(PS_{P^\star}(a_i))[z_j \rightarrow component\_z_j]$,
   $a_i \sqsubseteq \mathbf{Pr}_{to}(SS_{P^\star}(a_i))[z_j \rightarrow componennt\_z_j]$,
   These axioms represent the predecessor and successor sets of all the unrefined activities in the pre-refinement process. Because in the post-refinement process, any activity refined from z_j will be considered as a subconcept of $\bigsqcup component\_z_j$, we replace the appearance of each z_j by corresponding $component\_z_j$. For example, $Start \sqsubseteq \forall to.component\_A$.
3. for each $z_j \in \mathrm{A}_{P^\star}$,
   $component\_z_j \sqsubseteq \mathbf{Pr}_{from}(PS_{P^\star}(z_j) \cup \{component\_z_j\})[z_j \rightarrow componennt\_z_j]$,
   $component\_z_j \sqsubseteq \mathbf{Pr}_{to}(SS_{P^\star}(z_j) \cup \{component\_z_j\})[z_j \rightarrow componennt\_z_j]$,
   These axioms represent the predecessor and successor sets of all the refined activities in the pre-refinement process. Due to the mechanism of **Decomposing**, we add corresponding $component\_z_j$ to their predecessor and successor sets, and replace the z_j with $component\_z_j$ for the same reason as before. For example, $component\_A \sqsubseteq \forall from.(Start \sqcup component\_A)$.
4. for each $a_i \in \mathrm{A}_{Q^\star}$,
   $a_i \sqsubseteq \mathbf{Ps}_{from}(PS_{Q^\star}(a_i))$,
   $a_i \sqsubseteq \mathbf{Ps}_{to}(SS_{Q^\star}(a_i))$,
   These axioms represent the predecessor and successor sets of all the activities in the post-refinement process. For example, $a_{22} \sqsubseteq \exists from.b_{12}$, $b_{23} \sqsubseteq \exists to.a_{23}$.
5. $Disjoint(a_i | a_i \in Q$ and $Hori(a) = z)$
   These axioms represent the uniqueness of all the sibling activities refined from the same z_j. For example, $Disjoint(a_{11}, a_{21}, a_{22}, a_{23})$
6. $Disjoint($ all the activity in P, and all the $component\_z_j)$.
   This axiom represents the uniqueness of all the activities before refinement. For example, $Disjoint(Start, End, A, B, component\_A, component\_B)$.

With the above axioms, ontology $\mathcal{O}_{P \rightarrow Q}$ is a representation of the horizontal refinement from P to Q by describing the predecessor and successor sets of corresponding activities with axioms.

Vertical Refinement Similar as horizontal refinement, suppose we have principle behaviour model P and a concrete process model Q, which has already been reduced w.r.t P to eliminate ungrounded activities. Any activity in Q can be grounded to some activity in P. Thus, after reduction, $\forall a \in \mathrm{A}_P, \exists b \in \mathrm{A}_Q$ that b is grounded to a, and vice versa. Therefore for each $x_j \in \mathrm{A}_{P^\star}$, we define $grounded\_x_j \equiv \exists groundedTo.x_j$.Then we construct an ontology $\mathcal{O}_{P \rightarrow Q}$ with following axioms:

1. for each activity $a_i \in \mathrm{A}_{Q^\star}$ and $vert(a) = x$
   $a_i \sqsubseteq \bigsqcup \exists groundedTo.x_j$

These axioms represent the grounding of activities by concept subsumption, which realise the **Renaming** in vertical refinement. For example, $a_{11} \sqsubseteq \exists groundedTo.E$, $b_{11} \sqsubseteq \exists groundedTo.F$.

2. for each $a_i \in \mathrm{A}_{P^\star}$
$grounded\_a_i \sqsubseteq \mathbf{Pr}_{from}(PS_{P^\star}(a_i))[x_j \to grounded\_x_j]$,
$grounded\_a_i \sqsubseteq \mathbf{Pr}_{to}(SS_{P^\star}(a))[x_j \to grounded\_x_j]$,
These axioms represent the predecessor and successor sets of all the activities in the pre-refinement process. Due the mechanism of **Renaming** we replace all the $x_j \in \mathrm{A}_{P^\star}$ by $grounded\_x_j$. Because **Decomposition** is not needed in vertical refinement, we stick to the original predecessor and successor sets. These axioms become the constraints on the activities in $Q^\star$.
3. for each $a_i \in \mathrm{A}_{Q^\star}$,
$a_i \sqsubseteq \mathbf{Ps}_{from}(PS_{Q^\star}(a_i))$,
$a_i \sqsubseteq \mathbf{Ps}_{to}(SS_{Q^\star}(a_i))$,
These axioms represent the predecessor and successor sets of all the activities in the post-refinement process. Notice that the ungrounded activities have been removed from the process.
4. for each $x \in \mathrm{A}_P$,
$Disjoint(a_i | a_i \in Q^\star$ and $vert(a) = x)$
These axioms represent the uniqueness of all the sibling activities refined from the same x.
5. $Disjoint(Start, End,$ all the $grounded\_x_j)$.
This axiom represents the uniqueness of all the activities before refinement. For example, $Disjoint(Start, End, grounded\_C, grounded\_D)$.

With above axioms, ontology $\mathcal{O}_{P\to Q}$ is a representation of the refinement from P to Q by describing the predecessor and successor sets of corresponding activities with axioms.

In both horizontal and vertical refinement, the number of axioms are linear w.r.t. the size of $P^\star$ and $Q^\star$. The language is $\mathcal{ALC}$.

3.3 Concept satisfiability checking

In ontology $\mathcal{O}_{P\to Q}$, all the activities in $Q^\star$ satisfy the ordering relations in $P^\star$ by satisfying the universal restrictions and satisfy the ordering relations in $Q^\star$ by satisfying existential restrictions. Given the uniqueness of concepts, the inconsistency between $P^\star$ and $Q^\star$ will lead to unsatisfiability of particular concepts. The relation between the ontology $\mathcal{O}_{P\to Q}$ and the validity of the refinement from P to Q is characterised by the following theorems:

Theorem 3. *An execution path containing activity a in Q is invalid in the refinement from P to Q,* iff *there is some $a_i \in Q^\star$ such that $\mathcal{O}_{P\to Q} \models a_i \sqsubseteq \bot$.*

Proof. For each a in Q the ontology $\mathcal{O}_{P\to Q}$ contains the axioms $a \sqsubseteq \mathbf{Ps}_{from}(PS_Q(a))$ and $a \sqsubseteq \mathbf{Ps}_{to}(SS_Q(a))$. The axioms $a \sqsubseteq \mathbf{Pr}_{from}(PS_Q(a))$ and $a \sqsubseteq \mathbf{Pr}_{to}(SS_Q(a))$ are derived from the axioms (item 1,2). Depending on the refinement either the axioms $a \sqsubseteq \exists groundedTo.x_j$ and $grounded\_x_j \equiv$

$\exists groundedTo.x_j$ or $a \sqsubseteq \exists compose.z_j$ and $component\_z_j \equiv \exists compose.z_j$ are in the ontology. (1) For the $\rightarrow$ direction the lhs holds, we demonstrate that a is unsatisfiable. Since a is invalid either $PS_Q(a) \not\subseteq PS_P(a)$ or $SS_Q(a) \not\subseteq SS_P(a)$. From Theorem 2 it follows that either $\mathbf{Pr}_{from}(PS_P(a)) \sqcap \mathbf{Ps}_{from}(PS_Q(a))$ or $\mathbf{Pr}_{to}(SS_P(a)) \sqcap \mathbf{Ps}_{to}(SS_Q(a))$ is unsatisfiable and therefore a is unsatisfiable since a is subsumed. (2) The $\leftarrow$ direction is proved by contraposition. Given a is unsatisfiable in $\mathcal{O}_{P \rightarrow Q}$. Assumed a is valid in the refinement then $PS_Q(a) \subseteq PS_P(a)$ and $SS_Q(a) \subseteq SS_P(a)$ holds. From Theorem 2 the satisfiability of $\mathbf{Pr}_{to}(SS_P(a))$, $\mathbf{Pr}_{from}(PS_P(a))$, $\mathbf{Ps}_{from}(PS_Q(a))$ and $\mathbf{Ps}_{to}(SS_Q(a))$ follows which leads to a contradiction to the satisfiability of a.

This theorem has two implications:

1. The validity of a refinement can be checked by the satisfiability of all the name concepts in an ontology;
2. The activities represented by unsatisfiable concepts in the ontology are the source of the invalid refinement.

we check the satisfiability of the concepts to validate the process refinement. Every unsatisfiable concept is either an invalid refinement or related to an invalid refinement.

With the help of reasoning, we can easily see that Fig.1b is an invalid horizontal refinement w.r.t. Fig.1a: $a_{22} \sqsubseteq \exists from.b_{12}$, also $a_{22} \sqsubseteq \exists compose.A$ thus $a_{22} \sqsubseteq \forall from.(Start \sqcup component\_A)$. However, b_{12} disjoints with both $Start$ and $component\_A$ therefore a_{22} is unsatisfiable. Similarly, we can detect that b_{12}, b_{23} and a_{23} are unsatisfiable. This implies the invalid routes in Fig.2a and further the invalid refinement of Fig.1b. Also, the vertical refinement w.r.t. Fig.1d is wrong while the vertical refinement w.r.t. Fig.1c is correct.

According to the underlying logic, reasoning complexity is ExpTime.

Helped by our analysis, the process architect remodels their process (Fig. 3). Now, the execution set of Fig. 3 is $\{[\mathsf{a_1a_2b_1b_2b_3}], [\mathsf{a_1a_2b_2b_1b_3}], [\mathsf{a_1a_2b_1b_3b_2}]\}$. Renaming of Fig. 3's execution set with respect to Fig. 1a yields $\{[\mathsf{AABBB}]\}$. After decomposition, we conclude that Fig. 3 correctly horizontally refines the process in Fig. 1a because $\{[\mathsf{AB}]\} \supseteq \{[\mathsf{AB}]\}$. As for validating vertical refinement with the component models, renaming yields $\{[\mathsf{EFGHD}], [\mathsf{EFHGD}], [\mathsf{EFGDH}]\}$. After reduction with respect to Fig. 1c and Fig. 1d, we conclude that Fig. 3 correctly grounds on Fig. 1c and Fig. 1d because $\{[\mathsf{C}], [\mathsf{D}], [\mathsf{CC}], [\mathsf{CD}], [\mathsf{DC}], [\mathsf{DD}], \ldots\} \supseteq \{[\mathsf{D}]\}$ and $\{[\mathsf{EFGH}], [\mathsf{EFHG}], [\mathsf{FHEG}], [\mathsf{FEGH}], [\mathsf{FEHG}]\} \supseteq \{[\mathsf{EFGH}], [\mathsf{EFHG}]\}$.

4 Evaluation

We have implemented the transformation of BPMN process models and refinement information to OWL-DL ontology. In addition to the transformation, we implemented a generator which creates random, arbitrarily complex refinement scenarios. Flow correctness is ensured by constructing the process models out of block-wise patterns that can be nested. The generator is parameterized by

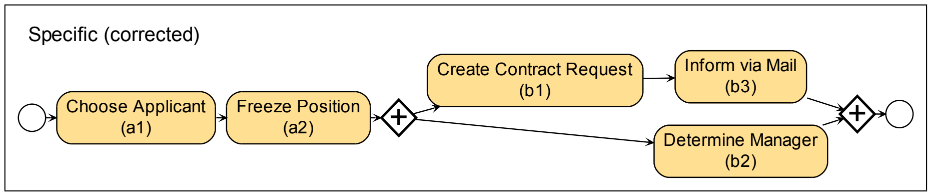


Fig. 3. Adapted specific process for correct refinement

the maximum branching factor B, the maximum length L of a pattern instance, the maximum depth of nesting N, and by the probability for loops, parallel, or exclusive flow. Most realistic appearing process diagrams were created with $B = 3$, $L = 6$, $N = 3$, and with a mixture of loops, parallelism, and exclusive flow to the ratio of $2 : 1 : 2$.

With the given parameters, we generated 1239 refinement scenarios (197 correct, 1042 wrong) with the average and maximum number of activities in the generic and refined models printed on the left-hand side below.

	Generic Activities	Specific Activities	Total Activities	Transf. Time	OWL-DL Axioms	Reasoning Time
Average	5.79	17.4	23.2	4ms	154	2.8s
Maximum	30	53	69	0.4s	1159	3.4min

The generated scenarios were used to evaluate the refinement analysis on a laptop with a 2 GHz dual core CPU, 2 GB of RAM using Java v1.6 and Pellet 2.0.0. Two factors contribute to the overall complexity of the analysis:

1. **Transformation to OWL-DL.** As we pointed out earlier, theoretically, the complexity of the transformation can be exponential in the worst case. However, our experiments show that in many practical cases, the size of the OWL-DL knowledge base—measured by the number of axioms—remains relatively small. In particular, for 80% (90%) of the scenarios, the number of axioms was below 220 (400) (see Fig. 4). Some unusual nesting of parallel flow causes the exceptions in the diagram that have a higher number of generated axioms. Remarkably, the appearance of such cases seems to uniquely distribute over the scenarios independently of the size of the original processes due to the artificial nature of the generated scenarios.
2. **OWL-DL Reasoning.** The theoretical complexity of OWL-DL reasoning is exponential as well. However, our evaluation runs in Fig. 5 suggest that for the practical cases evaluated, reasoning time grows less than exponentially (less than a straight line on a logarithmic scale) compared to the number of axioms in the OWL-DL knowledge base. We separately plot the reasoning times of the correct and wrong refinements because classifying a consistent knowledge base is more expensive in general.

When comparing absolute times, reasoning consumes about two orders of magnitude more time than transformation as can be seen from the right-hand

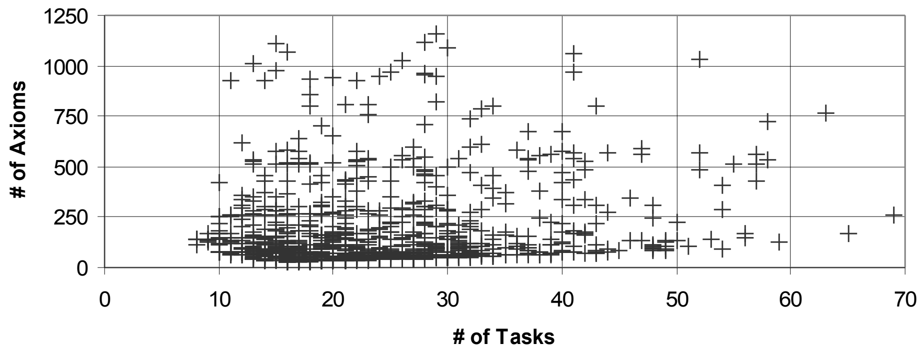


Fig. 4. Axioms for activities

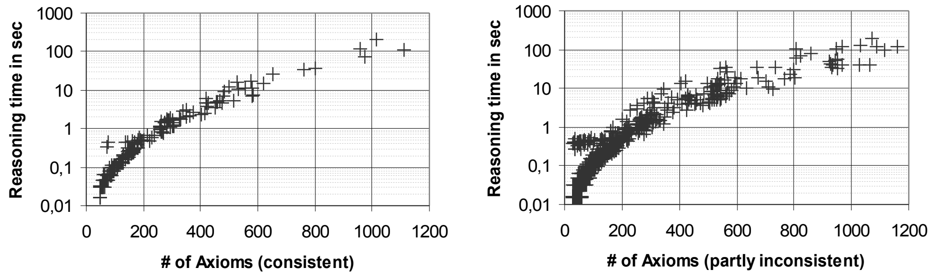


Fig. 5. Reasoning time for axioms on logarithmic scale

side of the table above. This determines our future research to seek improvements in the reasoning rather than in the transformation.

As for the complete run time, the above table indicates that the refinement analysis of an average scenario—with a generic process of about 6 activities and a refining process of about 17 activities—would take about 3 seconds. We consider this a simpler, yet realistic problem size.

In one of the larger evaluated scenarios, 15 generic activities were refined to 48 specific activities (for comparison: our running example contains 5 specific activities). The 63 activities in total (= 15+48) were transformed to 402 activities due to many parallel flows in that scenario. Analysis of the 765 generated axioms was performed in 18 seconds.

In the most complicated scenario of our evaluation, where a large knowledge base had to be constructed due to the heavy use of parallel gateways, total analysis time remained below 4 minutes. Although this is definitely too much for providing a real-time refinement check to process modelers, analysis took less than 1 second for 80% ($\leq$ 220 axioms) and less than 10 seconds for 90% ($\leq$ 400 axioms) of the examined practical cases. Compared to the manual efforts a human is required today, our approach provides a significant improvement. Furthermore, the check performed by our approach is—in contrast to the manual approach—guaranteed to be correct and thus helps to avoid costly follow-up process design errors.

5 Related Works and Conclusion

There are many existing works related to our study. Some of them [17, 14] come from the business process modelling and management community which investigated process property checking with model checkers.

Researchers in system transition and communication systems [13, 11, 10, 12] also developed behaviour algebra to analyse the bisimulation, i.e. matching between processes. In some of the works, execution set semantics are also applied [18]. However, these models do not validate refinement with activity compositions.

Other models use mathematical formalisms to describe concurrent system behaviour. [3] describes concurrent system behaviour with operational semantics and denotational semantics. But the analyzed equivalence between process models does not distinguish between deterministic and non-deterministic choices.

Semantic web community contribute to this topic by providing first semantic annotations for process models such as service behaviour and interaction [4, 16, 15] and later automatic process generation tools [7]. However, these approaches do neither consider process refinement nor a DL based validation of relationships.

In [8] actions and services, which are a composition of actions are described in DL. Actions contain pre- and post-conditions. The focus is on a generic description of service functionality. As inference problems, the realizability of a service, subsumption relation between services and service effects checking is analyzed. Services are described similarly with DL in [2]. The reasoning tasks are checking of pre- and post-conditions of services. The main focus of this work is the reasoning complexity.

The DL $\mathcal{DLR}$ is extended with temporal operators in [1] for temporal conceptual modelling. In this extension, query containment for specified (temporal) properties is analyzed. In [9] the DL $\mathcal{ALC}$ is extended with the temporal logics LTL and CTL. Still, neither of them considers process modelling and refinements.

Our contribution is this paper includes:

1. Devising a general approach to represent and reason with process models containing parallel and exclusive gateways;
2. Applying graph-based topological approach with DL reasoning to provide automatic solution of process refinement checking;
3. Implementing and evaluating a prototype that performs process transformation and refinement checking as proposed.

In the future, there are several potential extension of this work. We will continue our implementation and evaluation to support larger and more complex process models. We will also try to extend the process representation with more expressive power. Another interesting topic is whether the process transformation itself can be automatically inferred by reasoning. We also want to integrate our refinement representation with other business process modelling ontologies.

References

1. A. Artale, E. Franconi, F. Wolter, and M. Zakharyaschev. A Temporal Description Logic for Reasoning over Conceptual Schemas and Queries. *Lecture notes in computer science*, pages 98–110, 2002.
2. F. Baader, C. Lutz, M. Milicic, U. Sattler, and F. Wolter. A Description Logic Based Approach to Reasoning about Web Services. In *Proceedings of the WWW 2005 Workshop on Web Service Semantics (WSS2005)*, Chiba City, Japan, 2005.
3. A.J. Cowie. *The Modelling of Temporal Properties in a Process Algebra Framework*. PhD thesis, University of South Australia, 1999.
4. Markus Fronk and Jens Lemcke. Expressing semantic Web service behavior with description logics. In *Semantics for Business Process Management Workshop at ESWC*, 2006.
5. M. Hepp, F. Leymann, C. Bussler, J. Domingue, A. Wahler, and D. Fensel. Semantic business process management: Using semantic web services for business process management. *Proc. of the IEEE ICEBE*, 2005.
6. M. Hepp and D. Roman. An Ontology Framework for Semantic Business Process Management. In *Proc. of 8th Internalional Conference Wirtschaftsinformatik*, 20007.
7. Joerg Hoffmann, Ingo Weber, T. Kaczmarek James Scicluna, and Anupriya Ankolekar. Combining Scalability and Expressivity in the Automatic Composition of Semantic Web Services. In *Proceedings of the 8th International Conference on Web Engineering (ICWE 2008)*, 7 2008.
8. C. Lutz and U. Sattler. A Proposal for Describing Services with DLs. In *Proceedings of the 2002 International Workshop on Description Logics*, 2002.
9. C. Lutz, F. Wolter, and M. Zakharyaschev. Temporal description logics: A survey. In *Temporal Representation and Reasoning, 2008. TIME'08. 15th International Symposium on*, pages 3–14, 2008.
10. R. Milner. *A Calculus of Communicating Systems*. Springer LNCS, 1980.
11. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
12. R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes. *Information and Computation*, pages 41–77, 1992.
13. Davide Sangiorgi. Bisimulation for Higher-Order Process Calculi. *Information and Computation*, 131:141–178, 1996.
14. WMP van der Aalst, HT de Beer, and BF van Dongen. Process Mining and Verification of Properties: An Approach based on Temporal Logic. *LNCS*, 3761:130–147, 2005.
15. I. Weber, J. Hoffmann, and J. Mendling. Semantic Business Process Validation. In *Proc. of International workshop on Semantic Business Process Management*, 2008.
16. I. Weber, Joerg Hoffmann, and Jan Mendling. Beyond Soundness: On the Correctness of Executable Process Models. In *Proc. of European Conference on Web Services (ECOWS)*, 2008.
17. P.Y.H. Wong and J. Gibbons. A process-algebraic approach to workflow specification and refinement. *Lecture Notes in Computer Science*, 4829:51, 2007.
18. George M. Wyner and Jintae Lee. Defining specialization for process models. In *Organizing Business Knowledge: The MIT Process Handbook*, chapter 5, pages 131–174. MIT Press, 2003.

Implementing Semantic Web applications: reference architecture and challenges

Benjamin Heitmann, Sheila Kinsella, Conor Hayes, and Stefan Decker

firstname.lastname@deri.org
Digital Enterprise Research Institute
National University of Ireland, Galway
Galway, Ireland

Abstract. To date, Semantic Web research has tended to focus on data modelling challenges, at the expense of software architecture and engineering issues. Our empirical analysis shows that implementing Semantic Web technologies creates challenges which can affect the whole application. Standard solutions and best practices for Semantic Web technologies are just emerging. The lack of these has been an obstacle for implementing and deploying applications which exploit Semantic Web technologies for real world use cases.

In this paper we conduct an empirical survey of Semantic Web applications. We use this empirical data to propose a reference architecture for Semantic Web applications, and to identify the four main challenges for implementing the most common functionality related to Semantic Web technologies from a software engineering perspective: (i) the issues involved in integrating noisy and heterogeneous data, (ii) the mismatch of data models and APIs between components, (iii) immature and belated best practices and standards, and (iv) the distribution of application logic across components. We describe two orthogonal approaches for mitigating these challenges: (a) simplifying the application architecture by delegating generic functionality to external service providers, and (b) assembling and customising of components provided by software frameworks for rapid development of complete applications.

1 Introduction

Semantic Web technologies simplify knowledge-intensive applications, by enabling a Web of interoperable and machine-readable data [1] based on formal and explicit descriptions of the structure and semantics of the data [2].

Existing research on deploying Semantic Web technologies has tended to focus on data modelling, and software architecture and engineering issues have been comparatively neglected. Benefits such as simplification of information retrieval [3], information extraction [4] and data integration [5] have been well researched.

However, in order to encourage wide scale adoption of Semantic Web technologies, the whole life cycle of Semantic Web data needs to be assessed in terms of efforts and pay back of the application development. According to [6] this life cycle includes: the initial phase of *ontology development*, followed by *planning* how to use the data, *creation* of new data or *refining* of existing data, then persistent *archiving* of data, and finally *publication* and external *access* of data. Creation, refining, archiving and publication may all be performed at runtime by the application, and as such involve aspects of software engineering and software architecture in addition to data modelling aspects.

While the challenges of ontology development have been analysed based on empirical data of ontology development projects [7], to our best knowledge no empirical analysis of the challenges involved in the implementation of creating, refining, archiving and publication of data based on Semantic Web technologies exists.

We have performed an empirical survey of 98 Semantic Web applications (Section 2), which allows us to identify the most common shared components and the challenges in implementing these components. Together, these components constitute a reference architecture for Semantic Web applications. The survey shows that implementing Semantic Web technologies creates challenges which can affect the whole application. Standard solutions and best practices for Semantic Web technologies are just emerging. The lack of these has been an obstacle for implementing and deploying applications which exploit Semantic Web technologies for real world use cases.

Based on the survey, we identify the four main challenges (Section 3) for implementing Semantic Web applications: (i) the issues involved in integrating noisy and heterogeneous data, (ii) the mismatch of data models and APIs between components, (iii) immature and belated best practices and standards, and (iv) the distribution of application logic across components. Identifying these challenges allows better assessment of the costs associated with adopting Semantic Web technologies within enterprises, and forms the basis for designing better software frameworks and software architecture for exploiting the emerging Web of Data.

Towards this goal, we present two approaches for mitigating the identified challenges (Section 4) from a software engineering perspective. The first approach proposes an architectural solution by delegating generic components to external service providers thus simplifying the application. An orthogonal approach is to provide better software engineering support with components provided by software frameworks for rapid assembling and customising of complete applications. Finally, we list related research (Section 5) and discuss future research (Section 6).

The main contributions of this paper are (1) an empirical analysis of the state of the art regarding the implementation of the most common components of Semantic Web applications, (2) a reference architecture for Semantic Web applications based on the empirical analysis, (3) identifying the main challenges in implementing these components which are introduced by Semantic Web technologies, and (4) two approaches to mitigate these challenges from a software engineering perspective.

2 Empirical analysis of Semantic Web applications

As our goal is to identify the main challenges introduced by implementing Semantic Web technologies, we have performed an empirical analysis of the most common capabilities specific to Semantic Web applications. In Section 2.1, we provide a classification for Semantic Web applications in order to differentiate them from other applications on the World Wide Web. Section 2.2 outlines the methodology of the survey. The results of our survey follow in Section 2.3. First, we present a description of components which abstract the most common functionality related to Semantic Web technologies. Secondly, we provide statistics about the variations amongst the implementations of the components.

2.1 Classifying Semantic Web applications and the Web of Data

The most basic requirement for a Semantic Web application is the use of RDF for the metadata used by the application. This can be derived from the fundamental role of RDF in the "layer cake" of Semantic Web standards [8]. Additionally a set of formal vocabularies should be used to capture the application domain, and `SPARQL` should be used as data query language, according to [9, definition 2.2]. All surveyed applications meet these requirements, except for applications using programmatic access to RDF data for efficiency reasons.

The Linked Data principles define how to publish RDF data, so that RDF data sets can be inter-linked [10] to form a Web of Data. The Linking Open Data community project(`http://linkeddata.org`) provides most of the currently available linked data.

2.2 Methodology of the survey

The survey of current Semantic Web applications has been performed in two parts, consisting of an architectural analysis and a questionnaire about the application functionality.

Architectural analysis The applications from two key demonstration challenges in the Semantic Web domain have been analysed to identify the most common functionality of Semantic Web applications: the "Semantic Web challenge"(`http://challenge.semanticweb.org/`), organised as part of the International Semantic Web Conference from 2003 to 2008, and the "Scripting for the Semantic Web challenge"(`http://www.semanticscripting.org`), organised as part of the European Semantic Web Conference from 2006 to 2008. Duplicate submissions have been eliminated, resulting in a total number of 98 surveyed applications.

The result of the architectural analysis is a list of components which provide an abstraction of the most common functionality which is required to implement Semantic Web standards. The components have been extracted from the architecture diagrams and the textual descriptions of the application architecture and implementation, depending on availability in the submitted paper. The components provide a common way to decompose the surveyed applications, so that components with similar functionality from different applications can be compared. This allows us to e.g. identify the need for data updating standards in section 3.3, as most applications have a user interface, but only a minority of applications allow creation of new data by the user.

Application functionality questionnaire Additionally a questionnaire was used to collect details about the implementation of the applications. The questionnaire contains 27 properties associated with 7 areas of functionality. The results from the questionnaire provide statistics about the range of variations in which the functionality of the common components has been implemented.

The questionnaire covers these areas of functionality: (1) implementation of Semantic Web standards, (2) support for data sources, (3) support for formal vocabularies that are heterogeneous and have diverse ownership, (4) implementation of data integration and alignment, (5) support for structured, semi-structured, unstructured or multimedia data, (6) support for authoring and editing of data, and (7) support for external data sources and the open-world assumption.

Only the applications from the "Semantic Web challenge" 2003 to 2006, and the "Scripting for the Semantic Web challenge" 2005 to 2007 were analysed with the questionnaire. The authors of the papers describing the applications where asked to validate and correct the details about their applications. Of the 50 applications analysed with the questionnaire, 74% validated their data.

2.3 Survey results

Taken together, the two parts of the survey can be combined to provide an overview of the state of the art in implementing the required functionality for Semantic Web technologies. The architectural analysis provides a list of the most common components, and the questionnaire provides statistical data about the different variations of implementing each component.

Table 1 shows the seven most common components, and lists the number of applications implementing a specific component by year.

year	number of applications	data interface	persistence storage	user interface	integration service	search service	authoring interface	crawler
2003	10	100%	80%	90%	90%	80%	20%	50%
2004	16	100%	94%	100%	50%	88%	38%	25%
2005	6	100%	100%	100%	83%	83%	33%	33%
2006	19	100%	95%	89%	63%	68%	37%	16%
2007	24	100%	92%	96%	88%	88%	33%	54%
2008	23	100%	87%	83%	70%	78%	26%	30%
total	98	100%	91%	92%	72%	81%	32%	35%

Table 1. Percentage of surveyed applications implementing the 7 most common components, per year and in total

2.4 Reference architecture for Semantic Web applications

The surveyed applications share a significant amount of functionality regarding common capabilities of Semantic Web applications. We abstract from the differences between individual applications and distinguish seven main components, which together constitute a reference architecture for Semantic Web applications by describing high-level concepts and terminology, without fixing interfaces [11, page 242].

The (i) **data interface** provides an abstraction over remote and local data sources, the (ii) **persistent storage** stores data and run time state, and the (iii) **user interface** provides access for the user. (i) to (iii) have each been implemented by **more than 90%** of surveyed applications. The (iv) **integration service** provides a unified view on heterogeneous data, and the (v) **search service** allows searching in data. (iv) and (v) have each been implemented by **70% to 80%** of surveyed applications. The (vi) **crawler** discovers and retrieves remote data, and the (vii) **authoring interface** allows creating new data and editing existing data. (vi) and (vii) have each been implemented by **30% to 40%** of surveyed applications.

In the following we describe the functionality of each component in detail and provide statistical data for the range of variations amongst the implementations of the surveyed applications. The full results of the architectural analysis are available on-line(`http://semwebapp-components.dabbledb.com/`), as are the details of the questionnaire results(`http://www.activerdf.org/survey/`).

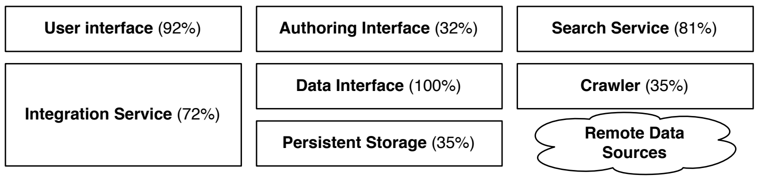


Fig. 1. The reference architecture for Semantic Web applications, with the percentage of surveyed applications implementing the component

Data Interface: *Also known as data adapter or data access provider.* Provides the **interface** needed by the application logic to **access local or remote data sources**, with the distinction based on either physical remoteness or administrative and organisational remoteness. Separation from the persistence layer is motivated by the function of the data interface as an **abstraction layer** regarding the implementation, number and distribution of persistence layers. 100% of the applications have a data interface.

Component variations: Accessing local data is implemented via programmatic access through RDF libraries by at least 50% of the applications. Only 24% use a query language for accessing local or remote data sources, but only half of these applications use the SPARQL standard. Multiple data sources with different ownership are used by 90% of applications, 70% support external data provided by the user and 60% can export their data or make it reusable as a source for other applications, by e.g. providing a SPARQL end-point. 76% of the applications support updating their data during application runtime.

Persistent Storage: *Also known as persistence layer or triple store.* Provides persistent **storage for data and run time state** of the application, it is accessed via the data interface. In practice many triple stores and RDF libraries provide both a data interface and persistent storage, but there are cases where the components are de-coupled, e.g. if the application has no local data storage, and only uses SPARQL to access remote data. 91% have a persistent storage.

Component variations: Possible supported standards include but are not limited to data representation languages (XML, RDF), meta-modelling languages (OWL, RDFS) and query languages (SQL, SPARQL). RDF is explicitly mentioned by 86% of applications, OWL is supported by 48%, RDFS by 22%. Inferencing or reasoning on the stored data is explicitly mentioned by 58% of the applications. Storage of any combination of structured, semi-structured, unstructured data or (binary) files can be implemented, with different levels of features or optimisation for the different data types. 58% implement support for unstructured text and 48% support mixing of structured and unstructured data in some way.

User Interface: *Also known as portal interface or view.* Provides a **human accessible interface** for using the application and **viewing the data**. Does not provide any capabilities for modifying or creating new data. 92% have a user interface, as some applications do not provide a human usable interface.

Component variations: The navigation can be based on data or metadata, such as a dynamic menu or faceted navigation. The presentation may be in a generic format, e.g. in a table, or it may use a domain specific visualisation, e.g. on a map (10%). 16% present images to the user and 6% explicitly mention support for audio content in the user interface. 28% support multiple languages in the user interface, thus catering to a multilingual audience.

Integration Service: *Also known as integration, aggregation, mediation, extraction layer or service.* Provides means for **addressing structural, syntactic or semantic heterogeneity** of data, caused by accessing data from multiple data sources using diverse kinds of format, schema or structure. The desired result is a **homogeneous view on all data** for the application. The integration service often needs to implement domain or application specific logic for the data integration. 72% of the applications have an integration service.

Component variations: Integration of heterogeneous data is supported by 90% of the applications, and 90% support data from sources with different ownership. Data from distributed data sources is supported by 72%. These three properties are orthogonal, as it would be e.g. possible to support just SIOC data [12] which is not heterogeneous, but which is aggregated from personal websites, so that the data sources are distributed and under different ownership.

Mapping or alignment between different schema may be automatic (12%), but most applications (80%) require some form of human intervention for the integration. Reasoning and inferencing can be used for the integration (58%). Integration may be performed once if data stays static, or continuously if new data gets added.

Search service: *Also known as query engine or query interface.* Provides the ability to **perform searches** on the data based on the content, structure or domain specific features of the data. **Interfaces for humans, machine agents or both** can be provided. 81% provide a search service.

Component variations: Besides search on features of the data structure or semantics, generic full text search (58%) or a search on unstructured and structured data at the same time (48%) can be provided. The interface for machine agents may be provided by e.g. a SPARQL, web service or REST endpoint.

Crawler: *Also known as harvester, scutter or spider.* Required if data needs to be found and accessed in a domain specific way before it can be integrated. Implements **automatic discovery and retrieval of data**. 35% implement a crawler. Some applications have an integration service, but do not need a crawler, e.g. because they only use local RDF data, but need to perform object consolidation [13].

Component variations: Support of different discovery and access mechanisms, like HTTP, HTTPS, RSS. Natural language processing or expression matching to parse search results or other web pages can be employed. The crawler can be active once if data is assumed to be static or continuous (76%) if new data needs to be discovered.

Authoring interface: Allows the user to **enter new data, edit existing data, and import or export data**. This component depends on the user interface component, and enhances it with capabilities for modifying and writing data. Separation between the user interface and the authoring interface reflects the low number of applications (32%) implementing write access to data.

Component variations: The annotation task can be supported by a dynamic interface based on schema, content or structure of data. Direct editing of data using standards such as e.g. RDF, RDF Schema, OWL or XML can be supported. Input of weakly structured text, using e.g. wiki formatting can be implemented. Suggestions for the user can be based on vocabulary or the structure of the data.

3 The main challenges for implementing Semantic Web technologies

The list of common components and the data about the variations in implementing these components allow us to identify the main challenges for Semantic Web application development: (i) the issues involved in integrating noisy and heterogeneous data, (ii) the mismatch of data models and APIs between components, (iii) immature and belated best practices and standards, and (iv) the distribution of application logic across components. In the following we detail these challenges and subsequently explain their impact on an example application.

3.1 Integrating noisy and heterogeneous data

An objective of RDF is to facilitate data integration [5] and aggregation [4]. However, even if all data sources were to use RDF as their data model, there would still exist potential integration issues due to different access mechanisms, noisy and erroneous data, and inconsistent usage of vocabularies and instance URIs between sources. Therefore, depending on how noisy and disconnected the data is, some amount of pre-processing may be required before using the data in an application.

Our survey shows that implementing integration of noisy and heterogeneous data can contribute the biggest part of the application functionality required for utilising Semantic Web technologies. The majority (72%) of surveyed applications implement an integration service. However manual intervention as part of the integration is necessary for 80% of applications. This means that prior to integration, either data is manually edited or data from the different sources is inspected in order to create custom rules or code. Only 20% explicitly mention fully automatic integration using e.g. heuristics or natural language processing. 76% allow updating of the data after the initial integration, and reasoning and inferencing is used for 58% of integration services.

Semantic Web data may be accessible by multiple different methods: large dumps to be downloaded, individual (possibly dynamically-generated) documents which need to be crawled, or via SPARQL endpoints to which queries must be issued. In order to ease the acquisition of published data, site suppliers can provide a semantic sitemap [14] on their website, so that crawling agents know where to find related RDF data. There are also a set of best practice guidelines [10] for publishing and interlinking pieces of data on the Semantic Web. The creation of ontologies is beyond the scope of this paper but has been discussed in previous literature [7].

Previous research performed using the Swoogle system [15] shows that there is a spectrum for RDF data, ranging from well structured data using stable and slowly changing vocabularies and ontologies to noisy and dynamic data with undefined terms and formal errors. The study tracked size changes in three versions of 183k RDF documents, and found changes had occurred in 60% of these documents. It also found that 2.2% of terms had no definitions and some had both class and property meta-usage. The Swoogle study also showed that the size distribution of Semantic Web documents is highly skewed, with many small documents and much fewer very large documents. Similarly, a study [16] using the WATSON infrastructure concludes that the Semantic Web is composed of many small, lightweight ontologies and fewer large, heavyweight ontologies. Assuming that within an ontology there is generally consistent use of vocabularies and instance URIs, the large number of smaller lightweight ontologies will present more problems for data integration.

The Semantic Web bug tracker(`http://bugs.semanticweb.org/`) is an initiative to improve the quality of the Web of data by tracking issues which could introduce inaccuracies in systems consuming the data. Previous work [17, 13] by the authors has also shown up various inconsistencies in RDF data on the Web. Some examples of frequent errors occurring in Semantic Web data observed from these sources are:

– **Use of non-standard terms:** There is frequent usage of classes and properties which are not defined in the official specifications. A study of ontology usage [17] shows over 1m definitions of instances of the class `foaf:chatEvent`, which does not exist in the official FOAF specifications(`http://xmlns.com/foaf/0.1/`).
– **Incorrect usage of vocabularies:**
Publishers frequently use terms from vocabularies in ways which they were not intended and which may introduce unexpected results after reasoning. For example, dbpedia, which publishes structured data extracted from Wikipedia, uses the property `foaf:img` to link resources of all types to associated images. This is problematic because according to the FOAF specifications, the property `foaf:img` has a domain of `foaf:Person`. This means that confusingly, a reasoning system could infer that all Dbpedia resources with images are of type `foaf:Person`.
– **Multiple URIs for the same objects:** The ability to uniquely identify arbitrary resources via URIs is an important factor in data integration. However there is little agreement between sources on which URIs to use for a particular resources. This is a problem as it may result in potentially useful information about a resource being missed. Reasoning on inverse functional properties (IFPs) can alleviate this to some extent. An evaluation of an object consolidation algorithm [13] showed that 2.4 million instances could be consolidated into 400k instances. However noise in IFP statements can cause even more problems. [13] notes that in filling out online profiles, users who do not wish to reveal their instant messaging usernames will fill in an alternative value such as “none”. As a result, 85k of the users supplying these non-unique usernames were incorrectly merged.

The majority of applications rely on data integration, but in order to implement it, expensive human intervention is necessary and knowledge about reasoning and inferencing needs to be acquired by the software engineers. Up to three components can be required for integrating data (the integration service, crawler and often the search service), which contribute to the requirements introduced by Semantic Web technologies to the application.

3.2 Mismatch of data models and APIs between components

Within the components of the surveyed Semantic Web applications there were two frequently occurring mismatches from a software engineering perspective: either they internally used different data models or the APIs between components were mismatched, both of which pose important challenges to implementing Semantic Web technologies.

The graph based data model of RDF provides the foundation for knowledge representation on the Semantic Web [1], however programmatically accessing RDF data from a component requires mapping of an RDF graph or subset (in the case of a SPARQL query result) to the data model used by the component [18].

Most of the surveyed applications (92%) were implemented using object oriented languages, and many of the surveyed applications stored data in relational databases. Web applications which are based on relational databases only have to manage the mismatch between object oriented and relational data. Semantic Web applications however have to additionally handle the graph data model of RDF.

Web applications utilise object relational mappers (ORM) such as Hibernate(`www.hibernate.org`) for Java or ActiveRecord(`http://ar.rubyonrails.org/`) for Ruby to transparently map between the data models, and similar approaches for mapping RDF data have been developed such as ActiveRDF [18] for Ruby or SuRF for Python(`http://pypi.python.org/pypi/SuRF`). Without such a mapper, the developer has to provide an abstraction layer on top of the RDF data model himself.

3.3 Missing or belated conventions and standards

In order to benefit from Semantic Web technologies, new paradigms such as the graph based data model of RDF and its open-world semantics need to be understood. On the other hand, many concepts and ideas to which Web application developers are accustomed are hard to translate to the stack of Semantic Web technologies. Approaches for providing conventions and standards to ease the shift towards Semantic Web technologies have often been designed with a considerable delay after the standardisation of RDF in 1999. Providing more and authoritative recommendations is an important factor for increasing adoption of Semantic Web technologies by enterprises.

All of the surveyed applications consume RDF data of some form, 70% allow accessing or importing of user provided external data, and 60% can export data or are reusable as a source for another application. However as discussed in section 3.1, there are many different export and access mechanisms for RDF data, from putting an RDF dump on a web server, embedding links to RDF data in HTML or providing a SPARQL endpoint.

Authoritative recommendations for making RDF accessible over the Web were not available until 2006, when Tim-Berners Lee published a design note(`http://www.w3.org/DesignIssues/LinkedData.html`) which established the Linked Data principles. RDFa specifies how to embed RDF graphs in XHTML documents, and has been in development since 2004. GRDDL (from 2007) specifies how to enable automatic conversion of HTML documents to RDF data.

The basic database interaction pattern of a Web application is "create, read, update and delete"(CRUD) [19], but Semantic Web applications can operate on both local and remote data sources so that updating and deleting depends on the provenance of the data [19]. The survey shows that there is a remarkable difference between the number of surveyed applications which provide a user interface (90%) and the number of applications which allow entering or editing data (30%).

Several approaches for enabling CRUD are currently under development, such as the Update extension for SPARQL, which provides the ability to add, update, and delete RDF data remotely. RDF forms and RDF pushback provide an architecture for structured data input, remote updating of data and conversion of RDF data to legacy data formats. However, at the time of writing the W3C was not involved in these efforts.

3.4 Distribution of application logic across components

The components of a Semantic Web application implement different areas of functionality which are required by Semantic Web technologies, however the components need to be controlled by the application logic in order to use the components for the application domain. For many of the components identified by the survey, the application logic is not expressed as code but as part of queries, rules and formal vocabularies.

58% of the surveyed application use inferencing and reasoning, which often encode some form of domain and application logic, 80% explicitly mention using a formal vocabulary, and 24% make use of an RDF query language. This results in the application logic being distributed across the different components.

The distribution of application logic is a well known problem for Web applications built on top of relational databases [20], and current web frameworks such as Ruby on Rails or the Google Web Toolkit(`http://code.google.com/webtoolkit/`) allow the application developer to control the application interface and the persistent storage of data programmatically through Java or Ruby, without resorting to e.g. JavaScript for the interface and SQL for the data storage. However approaches for centralising the application logic of Semantic Web applications still have to be developed.

3.5 The impact of the challenges on an example application

The challenges of implementing Semantic Web standards become apparent even for small applications. Figure 2 shows the architecture of an application from the authors previous work, the *SIOC explorer* [21]. It aggregates content from weblogs and forums exposing their posts and comments as RDF data using the SIOC vocabulary [12]. The **application logic** and most parts of the application are implemented using the Ruby scripting language and the Ruby on Rails(`http://rubyonrails.org/`) web application framework. The **user interface** allows faceted browsing of the SIOC data and is implemented through the BrowseRDF Ruby component [19]. The **data interface** is provided by ActiveRDF[18], which is an object-oriented Ruby API for accessing RDF data. It is used to access the **integration service**: The data interface is also used to access the **persistent storage** of RDF data using the Redland library(`http://librdf.org/`). Other application data is persistent to a MySQL relational database. The **crawler** is implemented through several Unix command line utilities which are controlled by Ruby. The *SIOC explorer* does not implement a search service or an authoring interface.

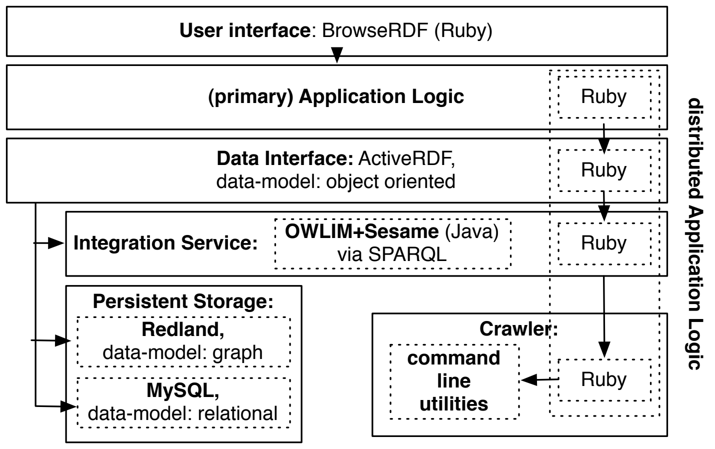

User interface: BrowseRDF (Ruby)
(primary) Application Logic
Ruby
Data Interface: ActiveRDF, data-model: object oriented
Ruby
Integration Service:
OWLIM+Sesame (Java) via SPARQL
Ruby
Persistent Storage:
Redland, data-model: graph
MySQL, data-model: relational
Crawler:
command line utilities
Ruby
distributed Application Logic

Fig. 2. Result of the architectural and functional analysis of the *SIOC explorer*

All four identified implementation challenges affect the *SIOC explorer*: (1) Even though all data sources use RDF and the SIOC vocabulary, the **data** is **noisy** enough to require two steps of **data integration**. The OWLIM extension of Sesame provides generic object consolidation, and integration specific to SIOC data is implemented as Ruby code. (2) The **components** are **mismatched**, regarding both the **data models**(object oriented, relational and graph based) and the **programming language APIs** (Ruby and Java). This requires mapping RDF to Ruby objects (ActiveRDF) and mapping relational data to Ruby objects (ActiveRecord). Sesame has no Ruby API, so SPARQL is used to access Sesame, resulting in slow performance for large numbers of concurrent read operations. (3) **Unclear standards and best practices** affect the crawler implementation, as different SIOC exporters require different methods to discover and aggregate the SIOC data, as RDFa and GRDDL were not in wide use when the SIOC explorer was developed in 2007. (4) The **application logic is distributed** across the primary application logic component, the data interface, the rules of the integration service and the code which controls the crawler.

4 Mitigating the software engineering challenges

We propose two approaches for mitigating these challenges. The first approach proposes an architectural solution by delegating generic components to external service providers thus simplifying the application. The second approach is to provide better software engineering support with components provided by software frameworks for rapid assembling and customising of complete applications. Both approaches use modularisation to delegate the implementation of some Semantic Web capabilities to components which are provided either by an external service or by a software framework.

4.1 Delegating generic components to external providers

The majority (72%) of surveyed applications implement an integration service, and in section 3.1 we have discussed the issues involved in integrating noisy data even if all sources support RDF. One possible approach to mitigate the identified

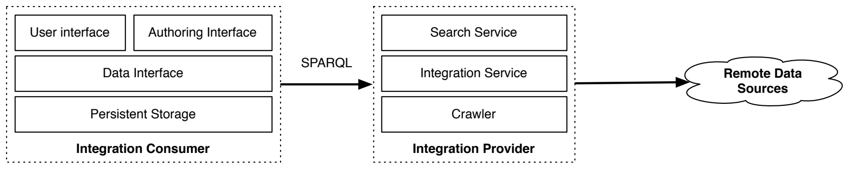


Fig. 3. Simplified Semantic Web application architecture after delegating data discovery, aggregation and integration to an integration provider

challenges is the delegation of generic data discovery, data aggregation and data integration to external providers.

In this way, external **integration providers** can provide the functionality of the integration service, search service and crawler. They provide high value services such as data aggregation and integration, which can be exploited by **integration consumers**. If the cost of discovering, aggregating and integration data on a Web scale is to high for some domains, the integration consumers can benefit from economies of scale when using external integration provider. Figure 3 shows the resulting simplified application architecture. Current search engines already provide APIs which can be utilised to search the whole web or just one specific site, however they lack the capabilities of Semantic Web technologies.

Delegating data integration to external providers can eliminate the need for (1) integration of semantic data if the application only needs generic services, like object consolidation based on inverse functional properties. As new (2) standards and best practices for discovering, publishing and updating of data become available, these only need to be implemented by the integration provider without affecting the **integration consumer**. The (3) mismatch of components is not affected by this, if SPARQL is efficient enough for the application. If domain specific integration of data is required, then different integration providers could provide specialised integration services for individual domains, just like generic and domain specific search engines exist today. However, (4) distributing application logic remains a challenge, as specialised and domain specific queries can contain important parts of the application logic.

The *SIOC explorer* can benefit from this approach by delegating the crawler which aggregates SIOC data from the different weblogs and forums, to an external integration provider such as Sindice [22], which continuously discovers and aggregates SIOC data and performs generic integration services such as object consolidation. This removes two components from the architecture, and allows the application to be purely implemented in Ruby thus eliminating API mismatches. All SIOC data would need to be accessed from Sindice via SPARQL, after which it can be stored in a local persistent store.

4.2 Assembling complete applications from components

While not explicitly described, most surveyed applications are at least partially created on a case-by-case basis: not just the application specific logic is implemented by a software engineer, but also at least one other component of the application. Very often the *user interface*, *integration service* or the *crawler* are custom made for the specific application. The survey shows that most applications are implemented with more then one programming language, which indicates that most applications are assembled from components with API mismatches.

Software frameworks provide the infrastructure for applications in the form of templates, components and libraries. The provision of software frameworks for implementing Semantic Web technologies has the potential to address all of the identified issues to a certain degree: (1) generic data integration can be provided through libraries provided by the framework. (2) the mismatch of components can be addressed if the framework provides all the components which are necessary to assemble a Semantic Web application, and if all parts of the framework provide APIs for the same programming languages. (3) New standards and best practices, e.g. for data discovery or publication, can be implemented as part of the framework, thus alleviating the need for the application programmer to implement them. (4) Distribution of application logic can be addressed through the framework by providing a central point for implementing the application logic, which can control and customise all of the components.

Similar benefits are already provided by modern Web application frameworks such as Ruby on Rails for Ruby, PHPCake for PHP and Django for Python. The Semantic Web Application Framework (SWAF) [19] provides a first step towards providing components for assembling and customising a complete Semantic Web application.

5 Related work

Our methodology is adapted from [23], which uses six cases studies of software systems as the basis for introducing the basic concepts of software architecture. This is used as the foundation for identifying the most important general challenges for the design of complex software systems which are constructed from many components. We adapt this approach for the field of Semantic Web technologies. The challenges which we identify are based on a survey of 98 Semantic Web applications.

Other empirical surveys about Semantic Web applications are publicly available, however they are not concerned with the software architecture and specific implementation details of concrete applications. Thus they provide no empirical basis for identifying the main challenges of implementing Semantic Web technologies. [24] presents the results of a survey of 627 Semantic Web researchers and practitioners done in January 2007. The questions from the survey cover the categories of demographics, tools, languages and ontologies. It tries to characterise the uptake of Semantic Web technologies and the types of uses cases for which they are deployed. Another similar survey of 161 researchers and 96 application-oriented participants was published online in 2009(`http://preview.tinyurl.com/semweb-company-austria-survey`).

[25] performs a survey and architectural analysis of 35 applications from the "Semantic Web challenges" in 2003, 2004 and 2005. The result is a prescriptive software architecture for Semantic Web applications described with UML. However, the results of the survey do not identify any software engineering challenges for implementing Semantic Web technologies.

While no other empirical analysis of the challenges of implementing Semantic Web applications exist, the ONTOCOM project [7] provides a detailed cost estimation model for ontology development projects. We do not provide a cost estimate model for software engineering of Semantic Web applications. However, our identification of the main challenges in implementing such applications provides the basis for future research on establishing such cost estimation models.

6 Conclusion

Semantic Web technologies enable new benefits such as semantically structured machine-readable data and the integration of data from multiple, heterogeneous sources. However, adopting new technologies adds effort resulting from implementing the new standards and their associated functionality. We have conducted an empirical survey of Semantic Web applications, which we have used to propose a reference architecture for Semantic Web applications, and for identifying the main challenges which are introduced by implementing Semantic Web technologies: the issues involved in integrating noisy and heterogeneous data, the mismatch of data models and APIs between components, immature and belated best practices and standards, and the distribution of application logic across components. These challenges have been an obstacle for the development of applications exploiting Semantic Web technologies. Two possible approaches for mitigating these challenges are: the simplification of the application architecture by delegating data integration to an external service provider, and assembling and customising of components provided by software frameworks.

The ecosystem of the emerging Web of Data will be based on integration providers and integration consumers. Integration providers provide access to data which has been discovered, aggregated and integrated in a generic way or which caters to a specific domain. Integration consumers will utilise these services to provide their users with benefits which are enabled by the Web of Data and by the integration providers. Data will be published and discovered according to community and industry best practices, which are increasingly implemented by ready-made components. The identified challenges and potential solutions enable future research to better assess the costs of adopting Semantic Web technologies within enterprises, and form the basis for designing better software frameworks and software architecture for exploiting the emerging Web of Data.

Acknowledgements: The work presented in this paper has been funded in part by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2).

References

1. Berners-Lee, T., Hendler, J.A., Lassila, O.: The Semantic Web. Scientific American **284**(5) (2001) 34–43
2. Decker, S., Melnik, S., Van Harmelen, F., Fensel, D., Klein, M., Broekstra, J., Erdmann, M., Horrocks, I.: The semantic web: The roles of XML and RDF. IEEE Internet computing **4**(5) (2000) 63–73
3. Abecker, A., van Elst, L.: Ontologies for knowledge management. In: Handbook on Ontologies in Information Systems. Springer (2004) 453–474
4. Davies, J., Fensel, D., van Harmelen, F., eds.: Towards the Semantic Web: Ontology-driven Knowledge Management. John Wiley and Sons (2002)
5. Fensel, D., Van Harmelen, F., Horrocks, I., McGuinness, D., Patel-Schneider, P.: OIL: An ontology infrastructure for the semantic web. IEEE intelligent systems **16**(2) (2001) 38–45
6. Möller, K.: A Lifecycle Model for Data on the Semantic Web, in progress. PhD thesis, National University of Ireland, Galway (2009)

7. Simperl, E., Popov, I., Burger, T.: ONTOCOM Revisited: Towards Accurate Cost Predictions for Ontology Development Projects. Proceedings of the International Semantic Web Conference (2009)
8. Gerber, A., van der Merwe, A., Barnard, A.: A Functional Semantic Web Architecture. Proceedings of the European Semantic Web Conference (2008)
9. Hausenblas, M.: Building Scalable and Smart Multimedia Applications on the Semantic Web. PhD thesis, Graz University of Technology (2008)
10. Bizer, C., Cyganiak, R., Heath, T.: How to Publish Linked Data on the Web. Technical report, FU Berlin (2007)
11. Endres, A., Rombach, D.: A Handbook of Software and Systems Engineering. Pearson Education (2003)
12. Breslin, J., Decker, S., Harth, A., Bojars, U.: Sioc: An approach to connect web-based communities. The International Journal of Web-Based Communities (2006)
13. Hogan, A., Harth, A., Decker, S.: Performing object consolidation on the semantic web data graph. In: Identity, Identifiers, Identification Workshop. (2007)
14. Cyganiak, R., Stenzhorn, H., Delbru, R., Decker, S., Tummarello, G.: Semantic sitemaps: Efficient and flexible access to datasets on the semantic web. In: European Semantic Web Conference. (2008)
15. Ding, L., Finin, T.: Characterizing the Semantic Web on the Web. Lecture Notes in Computer Science **4273** (2006) 242
16. d'Aquin, M., Baldassarre, C., Gridinoc, L., Angeletou, S., Sabou, M., Motta, E.: Characterizing Knowledge on the Semantic Web with Watson. In: Workshop on Evaluation of Ontologies and Ontology-based tools of the International. (2007)
17. Kinsella, S., Bojars, U., Harth, A., Breslin, J.G., Decker, S.: An interactive map of semantic web ontology usage. In: Information Visualisation, 2008. IV '08. 12th International Conference. (2008) 179–184
18. Oren, E., Heitmann, B., Decker, S.: ActiveRDF: embedding Semantic Web data into object-oriented languages. Journal of Web Semantics (2008)
19. Oren, E., Haller, A., Hauswirth, M., Heitmann, B., Decker, S., Mesnage, C.: A flexible integration framework for semantic web 2.0 applications. Software, IEEE (2007)
20. Leff, A., Rayfield, J.: Web-application development using the model/view/controller design pattern. Proceedings of the International Enterprise Distributed Object Computing Conference (2001) 118–127
21. Bōjars, U., Heitmann, B., Oren, E.: A Prototype to Explore Content and Context on Social Community Sites. In: Proceedings of the International Conference on Social Semantic Web (CSSW). (2007)
22. Oren, E., Delbru, R., Catasta, M., Cyganiak, R., Stenzhorn, H., Tummarello, G.: Sindice.com: A document-oriented lookup index for open linked data. International Journal of Metadata, Semantics and Ontologies (2008)
23. Garlan, D., Shaw, M.: An introduction to Software Architecture. Advances in Software Engineering and Knowledge Engineering **1** (1993) 1–40
24. Cardoso, J.: The Semantic Web Vision: Where Are We? IEEE Intelligent Systems **22** (2007) 84–88
25. Cunha, L.M., de Lucena, C.J.P.: Cluster The Semantic Web Challenges Applications: Architecture and Metadata Overview. Technical report, Pontificia Universidade Catolica do Rio de Janeiro (2006)

Fly-By-OWL: A Framework for Ontology Driven E-commerce Websites

Azhar Jassal, David Bell

Fluidity Research Group
Brunel University, Uxbridge, West London, UB8 3PH, UK
{Azhar.Jassal, David.Bell}@brunel.ac.uk

Abstract: Fly-By-OWL allows Web developers to create "ontology-driven" E-commerce websites that can harness live reasoning. It aims to make the Semantic Web's underlying technologies (ontologies and reasoning) relevant to Web developers. To demonstrate Fly-By-OWL, the "Semantic Pizza Store" is presented, an example store that uses the output of the pizza ontology tutorial as its knowledge base. By making use of inferences, products can be categorised dynamically and product customisation can rely upon consistency rather than hard-coded rules.

Keywords: Ontology, Semantic Web, OWL, Web Application, CGI, Dynamic Website, E-commerce, LAMP, Database, DBMS, Pizza

1. Introduction

The World Wide Web was the "killer app" that spread the internet to every home and office across the world. In the beginning, a website was just a collection of interlinking pages written in Hypertext Mark-up Language (HTML). The introduction of the Common Gateway Interface (CGI) specification in 1993 allowed a website to become both dynamic and interactive. From then on the design of a Web page could be separated from its content. Pages were no longer static; they were generated on-the-fly with content that was customised for each user's request. This spurred the advent of a whole range of Web applications, allowing everything from Web-based email (the original HoTMaiL) and e-commerce sites (Amazon) to become possible. While many technologies exist today that allow programming for the Web, from servlets to dynamic scripting languages, it was CGI that long ago led the way. With the design of a website separated from its content, relational databases found their place as the default backend of Web applications.

The extensive features provided by a fully fledged database management system (DBMS) made them the Web developer's default choice for not just data but all content handling. From the advent of the dynamic website to the present day, databases are used to hold everything needed to feed a Web application, from user details and visitor information, to content such as product information for an e-commerce store and the messages that somebody posts on their Blog. It is databases that hold the content that feeds today's Web. Yet, the Web itself has held limits that stem from its original design, by being based upon Hypertext Mark-up Language (HTML), nothing is said about what the data is for, i.e. about its semantics [1]. The biggest problem with the Web is that "information is dumb; the data contained in websites does not know what it is" [2]. To address this, the Semantic Web project was initiated in 2001 and aimed to bring structure to the meaningful content of Web pages [3].

The paper begins by analysing if Semantic Web technologies have found any utilisation within the backend of a present day Web application. Related work is discussed, noting the position of ontologies within those Web applications and their significance to its overall operation. Fly-By-OWL is then introduced; a framework that allows developers to create "ontology-driven" Web applications that can harness the potential of live reasoning. The first implementation of Fly-By-OWL focuses on an e-commerce context, by specifically allowing developers to create "ontology-driven" business-to-consumer (B2C) e-commerce websites. The e-commerce context is chosen to "emphasise upon practical application" which is necessary to work towards the widespread adoption of the Semantic Web [4]. The framework is then demonstrated with the "Semantic Pizza Store", which uses the ontology output from the Protégé pizza tutorial [5].

2. Finding a place for Semantics in Today's Web

It has been almost 20 years since the initial advent of the Web. According to the Netcraft Web Server Survey (July 2009), there are now 240 million hostnames running a publicly accessible Web server, compared to 24 million in October 2000 [6]. With 8 years passed since the commencement of the Semantic Web project, how many mainstream Web applications are now employing Semantic technologies? And where have developers, those responsible for serving relevance to real users, positioned them within their Web applications? O'Reily [7] wrote that "the Internet is the most striking example of a marketplace, full of economic opportunity, which has grown out of the open-source software community". As of July 2009, Apache, an open source Web server, holds the dominant 47% share of the market [6] and forms part of the larger LAMP stack for Web applications. LAMP is a free open-source software bundle which provides the principal components used to build a viable general purpose web server. Once mostly used for small-scale Web development, it has advanced its way into mainstream corporate software development, being used by Google and Yahoo to build search applications, while Lufthansa and the Sabre Travel Network used it to develop travel reservations systems [8].

In order to gather a snapshot of how developers are using Semantic technologies, six popular LAMP based projects with highly active user communities are analysed in Table 1, to determine whether any features of these packages utilise ontologies.

Support is categorised as either: standard (official feature), planned in roadmap (upcoming standard feature) or unofficial add-on (community maintained contribution).

Table 1. Ontology utilisation within six widely used LAMP based Web applications

Project	**Description**	**Standard**	**Roadmap**	**Add-on**
Drupal	Content Management System (CMS) used as a back-end for websites. Is followed by over 550,000 registered users and is used by thousands of websites. User contributed add-on: Drupal SIOC			✓
Joomla	Another CMS, used as a back-end to websites. Like Drupal, used by many prominent websites, followed by over 300,000 registered users. User contributed add-on: GoodRelation's for VirtueMart (a Joomla e-commerce extension)			✓
phpBB	Most widely used open-source bulletin board system in the world, followed by over 350,000 registered users. User contributed add-on: phpBB SIOC			✓
osCommerce	Open source out-of-the-box e-commerce solution with a community of 210,000 registered users. User contributed add-on: GoodRelation's			✓
vBulletin	Commercial bulletin board software used by thousands of websites, followed by a community of 200,000 registered users. User contributed add-on: vBulletin SIOC			✓
WordPress	Open source blog publishing application and CMS used by millions of websites. User contributed add-on: WordPress SIOC			✓

In an effort to find answers to the two questions posed beforehand, these six popular LAMP based Web applications were examined. The results show a clear picture, while all of the applications have a community contributed add-on that utilises ontologies in one form or another; the actual developers of the applications have no plans to use ontologies within a standard feature. This makes a simple fact apparent: while the Semantic Web does have an active research community that wishes to captivate the use of it's technologies within these projects, the technologies themselves have not proved their present real world benefit to developers. McBride [4] wrote that in order to step towards widespread adoption of Semantic Web technologies, we must "emphasise practical applications" and that we must start to "develop applications now". That was 7 years ago and while the Web continues to show unprecedented growth, the same unfortunately cannot be said about the Semantic Web. It has still not made any impact that can be felt by a real world Web user, i.e. common man.

What these six popular Web applications share in common is that they are all "database-driven". The LAMP stack itself positions the database as a fundamental building block of a Web application, with the "M" in LAMP standing for MySQL, an open source DBMS. The initial advent of dynamic websites, where design could be separated from content, was a giant leap. With it, the DBMS became the default "content-handler", and so it seems what was made default with that great leap, has continued to remain default into the present day. The Web, above any other development platforms before it, is led by example. This can be blamed to being a by-product of its rapid growth. In 2004, the World Wide Web Consortium (W3C) selected a standard ontology language, OWL (Web Ontology Language). It is based upon Description Logic (DL) and exploits existing work on languages such as OIL and DAML+OIL [9]. With an ontology language chosen, it would have been expected for some Web applications to begin to adopt ontologies as their knowledge bases over a DBMS, but ontologies have still not become the backend to any noticeable real world Web application.

With Semantic technologies still not being accommodated for, is the Web of today any different from when CGI first made its advent? Perhaps the Web has been stuck in an endless loop; by using in essence the same building blocks from the advent of CGI that cannot accommodate semantics. Rob McCool, the author of "httpd" which later became Apache wrote that "without radical simplification, the Semantic Web will continue to see limited participation and few compelling applications" [10]. In order to move the field forward, this paper proposes an approach for creating "ontology driven" e-commerce websites that goes back to basics. It focuses upon being directly relevant to a general purpose Web developer. It allows novice Web developers to harness ontologies as their primary knowledge base, rather than a DBMS and make use of live reasoning. By allowing developers to "begin developing now" and by "emphasising practical applications" [4], it aims to progress adoption of the Semantic Web by making its underlying technologies more relevant to Web developers, as they build the applications that are relevant to real Web users.

3. Ontology utilisation within Web Applications

As can be seen in Table 1, the Semantic Web community has been active in contributing add-ons for popular Web applications that enable them to make some utilisation of Semantic technologies. An add-on or extension is an optional component, and installed by a user to add specific functionality that the developers did not deem significant enough to have to offer as standard. The Semantically-Interlinked Communities (SIOC) Initiative uses an ontology to represent "social data" in RDF [11]. By offering add-on's for four of the six Web applications listed in Table 1, large ontologies would be generated by exporting content. By offering its add-on "exporters" for various applications, SIOC envisions being able to interlink these "independent" and separated communities. The ontologies themselves do not play any role vital to the function of the overall Web applications, and serve a different purpose to the scope of the "ontology-driven" progression being proposed by this paper.

The remaining two Web applications listed in Table 1 make their ontology utilisation with GoodRelation's, a lightweight ontology for annotating e-commerce offerings [12]. GoodRelation's provides a vocabulary for describing the types of goods and the terms and conditions of items and services offered on the Web. It is an accepted vocabulary of Yahoo! SearchMonkey, to accommodate structured data for their search engine. The add-on's for the e-commerce Web applications (an extension in the case of Joomla) generate structured data following the GoodRelation's vocabulary from the product data already present within the Web applications back-end database. GoodRelation's provides a suitable vocabulary for the knowledge base of an "ontology-driven" e-commerce Web application. This paper focuses on a framework that allows the creation of such websites, where the Web developer is free to choose the most appropriate vocabulary for their knowledge base. While SOIC and GoodRelation's were projects that happened to contribute add-on's, there are also other projects that concern using ontologies within Web applications.

Stojanovic et al. [13] provides a reverse engineering approach to migrating data-intensive websites to the Semantic Web. By transforming a present day back-end relational database model into corresponding ontological structures, content is mapped from the database into an ontology. OntoWeaver, a website design framework, uses ontologies to drive the design and development of data-intensive websites [14]. OntoWebber is a model-driven ontology-based system architecture for creating data intensive Web sites and portals [15]. Of these projects, the most closely related to the scope of this paper is OntoWiki, an open source semantic wiki Web application which facilitates the visual representation of a knowledge base as information maps [16]. Within OntoWiki, Erfurt is being developed, a Semantic Web toolkit with a native reasoner (and DIG capabilities) for the PHP programming language. While the context is wiki's, the usage of ontologies as the primary knowledge base that drives the Web application makes it the most relevant to this paper. The simultaneous creation of a Semantic toolkit for PHP, which along with Perl and Python makes the "P" in LAMP, could be another vital element in enabling Semantic technologies to find a more prominent position within Web applications.

Bearing in mind the call for "rapid simplification" or the Semantic Web having to face limited partition [17], we present a framework that goes back to basics and focuses on being relevant to even casual Web developers. By bringing together the standard ontology language (OWL), semantic reasoners (e.g. FaCT++, Pellet) and ontology editors (e.g. Protégé), the Fly-By-OWL framework allows the most novice Web developer to create an "ontology-driven" e-commerce website. It allows Web developers to harness the artificial intelligence (AI) capabilities provided by reasoning in their websites. It introduces the "ontology-driven" concept in an e-commerce context to "emphasise upon practical application" [4]. With Fly-By-OWL, developers can manipulate a knowledge base (with live reasoning on-the-fly) just as their applications would have previously interacted with a DBMS. Fly-By-OWL aims to make the creation of "ontology-driven" websites a possibility for even casual Web developers and practically demonstrate the leap provided by using ontologies over present day means. To the best of our knowledge, the implementation will be the first to allow an everyday Web user to interact with an OWL knowledge base with live reasoning within some context, e.g. e-commerce. Allowing Web developers to harness the abilities of a reasoner and progressing from “database-driven” to “ontology-driven” will allow us to step closer to the Semantic Web.

4. Fly-By-OWL: Ontology Driven Websites

The Fly-By-OWL framework allows even a novice Web developer to create “ontology-driven” Web applications. The initial implementation of the framework focuses on using it to create e-commerce stores. With product information presently held in a database, a Web developer would write a catalogue/ shopping cart which would interact with a DBMS. Fly-By-OWL presents the Web developer with a data-model of an OWL knowledge base (with inferences) which can then be presented through HTML however the developer envisions. The knowledge base can be queried using Manchester Syntax [18]. Using ontologies allows limitations imposed by the present day “database-driven” approach to be overcome. For example, product’s can be categorised on-the-fly using equivalent classes in the ontology, by using reasoning and inferences which place products within appropriate categories. Product specifications can be customised without any hard coded rules as reasoning can indicate whether a class is now inconsistent.

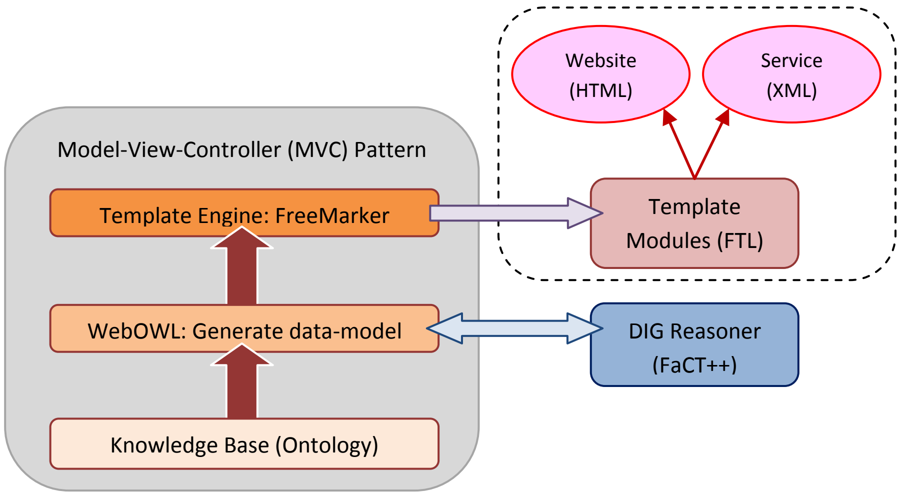

Website (HTML)
Service (XML)
Model-View-Controller (MVC) Pattern
Template Engine: FreeMarker
Template Modules (FTL)
WebOWL: Generate data-model
DIG Reasoner (FaCT++)
Knowledge Base (Ontology)

Figure 1: Fly-By-OWL Framework

The framework is compromised of three layers (see Fig. 1) and uses a Model-View-Controller (MVC) pattern to isolate “content”, in essence business logic (the knowledge base) from “design”. While the framework is able to produce output in either HTML (Web browser, for humans) or XML (Web services, for machines), the first implementation focuses on HTML, to “emphasise upon practical application” [4] within the context of e-commerce websites. Fly-By-OWL requires a live constant connection (over DIG) to a reasoner, which runs as the backend reasoning server.

4.1 Bottom Layer: Knowledge Base (Ontology)

The bottom layer of the framework compromises of the ontology itself. The knowledge base can be created in Protégé, an open source ontology editor. The implementation of the framework supports both OWL and OWL 2 ontologies. In a present day Web application, the knowledge base could be served by anything from a flat file up to a DBMS. By using an ontology, artificial intelligence capabilities can be harnessed by the Web developer that with semantics, are more rich in scope and ability when compared to present day means.

4.2 Middle Layer: Data-model Generator

The middle layer of the framework interacts with the knowledge base and a reasoner (over DIG) to generate a data-model. Whereas the behind-the-scenes of a current day dynamic website may use a DBMS as a database server, Fly-By-OWL uses FaCT++ as a reasoning server. Once inferences have been made, the data-model is generated following a set specification. A custom query in Manchester Syntax can be passed into this layer. If one is received, it is made equivalent to a temporary class which is added to the knowledge base that holds results after reasoning.

4.3 Top Layer: Template Engine

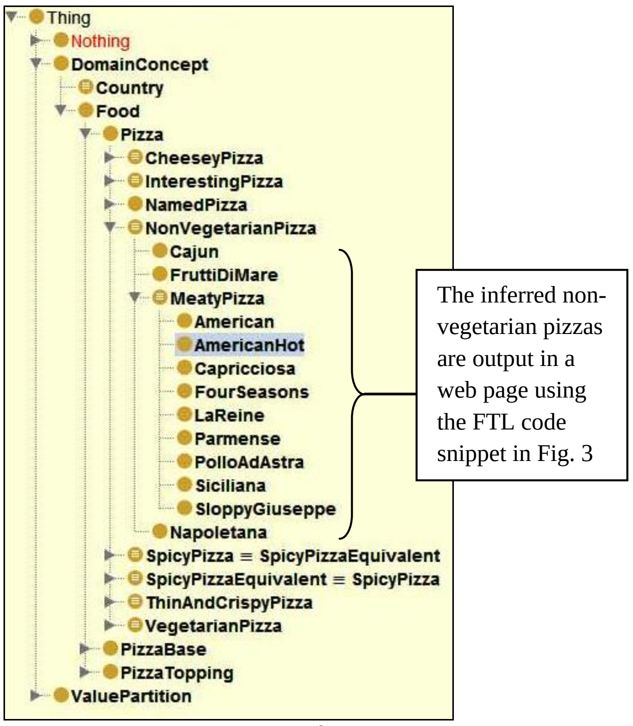
The top layer of the framework uses FreeMarker [19] as the template engine and is able to produce output for either a website (HTML) or Web service (XML). The output is customised by scripting templates in either HTML or XML with the FreeMarker Template Language (FTL). FTL allows the user to manipulate and fetch elements from the data-model. The Web developer can place these elements however they envision through the use of templates. Template “modules” are created that each present both different information and functionality to a user.

4.4 Template modules

Template modules allow a Web developer to create a Web application with Fly-By-OWL just as they would with any other dynamic scripting language (e.g. PHP, ASP, and JSP). For example, the template module “index” may contain an initial welcome page to the website, whereas the template module “products” may list items and the module “customise” may provide the functionality to enable a user to customise a products properties (with reasoning used to check for consistency). Using modules makes creating a website with Fly-By-OWL no different from how Web applications have always been created with dynamic scripting languages. Pages hold individual functionality, and once interlinked create a complete Web application. Some examples of these individual pages can be: “home” (home page), “catalogue” (product catalogue), “cart” (visitors shopping cart), “checkout” (start of checkout), “complete” (end of checkout).

4.5 Fly-By-OWL in practice

To illustrate how Fly-By-OWL works in practice, Fig. 2 displays the inferred class hierarchy of the ontology output from the Protégé pizza tutorial [5]. Here, the “NonVegetarianPizza” class shows its inferred results; pizzas that contain a meat or seafood topping. From an e-business perspective, a Web developer may want to use the inferred results of this class as a non-vegetarian category on a pizza ordering e-commerce website.



The inferred non-vegetarian pizzas are output in a web page using the FTL code snippet in Fig. 3

Figure 2: The pizza ontology tutorial (inferred class hierarchy)

By using FTL, “NonVegetarianPizza” can be found in the data model. Iterating through its contents will allow a Web developer to display the pizzas that contain a meat or seafood through a “dynamic category”. The category is dynamic as the pizzas within it have not been specified by humans, but are placed there due to their properties and through inference. An excerpt from a template which makes use of the inferred results of the “NonVegetarianPizza” class, written in HTML with FTL scripting can be seen in Fig. 3. This is a code snippet from the “pizzas” page of the “Semantic Pizza Store” (discussed further in section 1.4). It demonstrates how to output the inferred subclasses of “NonVegetarianPizza”. Following the data-model specification, elements can be retrieved from the knowledge base and output to the user. The code snippet in Fig. 3 iterates through the data-model at “NonVegetarianPizza”, displaying a box for each pizza, made from a table that contains its name and toppings (inferred pure classes). FTL provides all of the common calls a Web programmer would require, such as ‘if’ and ‘else’ clauses, ‘for’ and ‘while’ loops, among many other functions that allow the manipulation of the data-model.

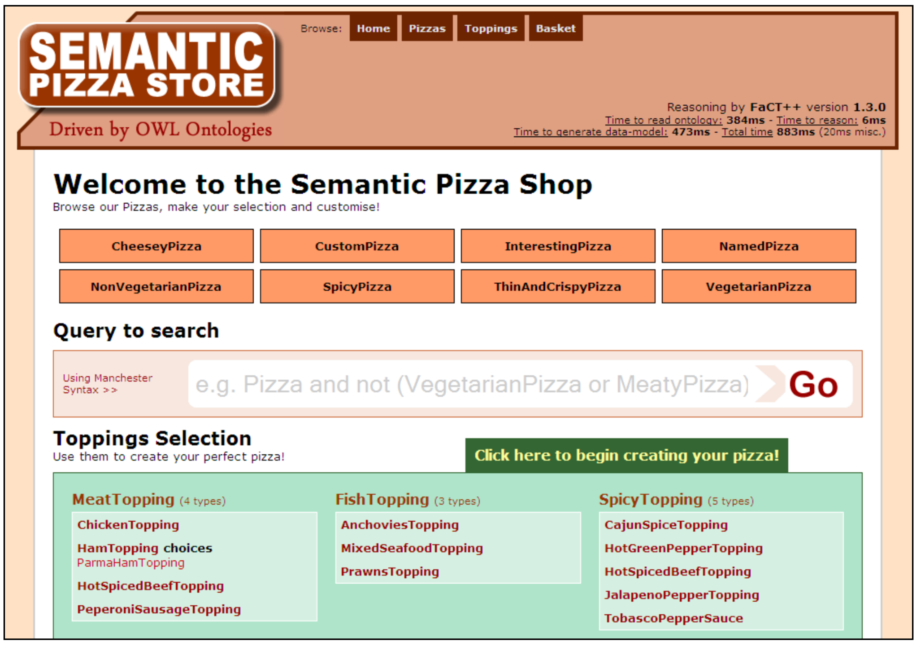
```
<#set whichClass = NonVegetarianPizza>
<#list whichClass?keys as pizzaName>
<table>
<tr>
<td><b>${pizzaName}</b></td>
</tr>
<tr><td>
        <table>
        <tr><td>

        <!-- Toppings //-->
        <table>
         <#list whichClass[pizzaName].PureClasses?keys as topping>
               <tr>
               <td>${topping}</td>
               </tr>
        </#list>
        </table>
        <!-- End Toppings //-->

        </td></tr>
        </table>
</td></tr>
</table>
</#list>
```

Figure 3: Snippet of the Pizza template module (FTL in red)

The first implementation of the Fly-By-OWL framework focuses on using it to create ontology driven business-to-consumer (B2C) e-commerce stores. This context was chosen as it can demonstrate in practice the benefits gained from being “ontology-driven”. Developers can explore the advanced product handling abilities gained by being able to reason and make inferences, and not have to rely on hard coded rules. The stores created using Fly-By-OWL will be the Web’s first ontology driven websites in real world context that require live reasoning to produce every page of output. By placing ontologies in real world Web context, the framework aims to make them relevant to Web developers. Once a product ontology has been prepared, for instance in Protégé, a Web developer must then create appropriate templates that showcase the products and allow the user to undertake common e-commerce store functions (such as browsing products, viewing additional product details). The implementation of the framework includes functionality within it that provides common features expected in an off-the-shelf e-commerce platform, such as a shopping cart, which is accommodated within the specification of the data-model.

5. Semantic Pizza Store

The example store that has been created to demonstrate the framework and its capabilities is the “Semantic Pizza Store”, based upon the output of the well known pizza ontology tutorial, written in Manchester by Matthew Horridge. The tutorial teaches ontological concepts to new Protégé users [5]. The ontology output from following the tutorial is used as the knowledge base for the store. The pizzas are offered for sale, with live inferences being made to categorise pizzas as Vegetarian, Non-Vegetarian, Spicy, etc and allowing the user to customise standard pizzas and create their own. The knowledge base can be queried on the fly in “Manchester Syntax”, for example requesting all pizza’s from a specific country or of a certain spiciness. The queries can be either input by the user and POST, or they can be passed in via an encoded URL in a GET request. The “Semantic Pizza Store” demonstrates both how ontologies are appropriate as the backend knowledge base to sell the products in question, pizzas and how they overcome the limitations of a product database. Fig. 4 shows a screen capture of the home page of the “Semantic Pizza Store”.



Reasoning by FaCT++ version 1.3.0
Time to read ontology: 384ms - Time to reason: 6ms
Time to generate data-model: 473ms - Total time 883ms (20ms misc.)

Welcome to the Semantic Pizza Shop

Browse our Pizzas, make your selection and customise!

Query to search

Toppings Selection

Use them to create your perfect pizza!

MeatTopping (4 types)
- ChickenTopping
- HamTopping choices
  ParmaHamTopping
- HotSpicedBeefTopping
- PeperoniSausageTopping

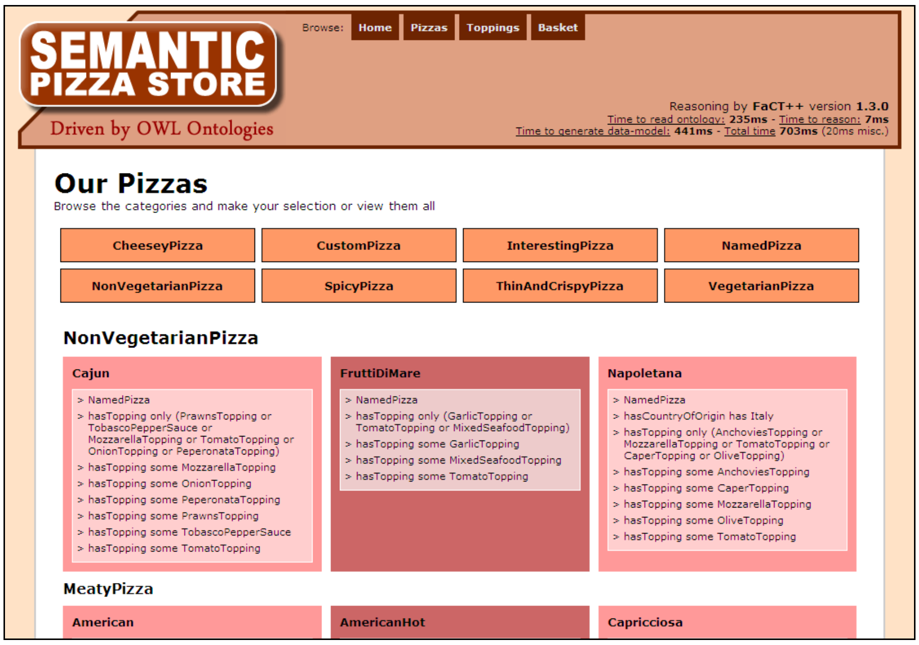
FishTopping (3 types)
- AnchoviesTopping
- MixedSeafoodTopping
- PrawnsTopping

SpicyTopping (5 types)
- CajunSpiceTopping
- HotGreenPepperTopping
- HotSpicedBeefTopping
- JalapenoPepperTopping
- TobascoPepperSauce

Figure 4: Screen capture of the “Semantic Pizza Store” home page

The popularity of the pizza ontology tutorial makes it appropriate to use in demonstrating Fly-By-OWL. Its concepts will be familiar to most researchers in the field. Fig. 4 shows the home page module (“index”) of the “Semantic Pizza Store”. The pizza subclasses are fetched from the knowledge base and displayed as clickable boxes, acting as “dynamic categories”, allowing the user to browse pizza’s of those types. The results of these classes are inferred. A user can query the knowledge base in Manchester Syntax, and toppings are displayed so a user can begin to create their own pizza. Additional functionalities of the “Semantic Pizza Store” include customising the preset pizzas, creating half and half’s and adding pizzas to a shopping cart from where the user can proceed to a checkout, among other functionality expected in a typical online pizza ordering service. The template modules of the example store are all written in HTML and make use of CSS and JavaScript, similar to other current day websites. The data-model is manipulated through FTL scripting, as was shown in the code snippet in Fig. 3.

The source code of the templates that are used by the “Semantic Pizza Store” is viewable online at the Fly-By-OWL website. Fig. 5 shows the pizza module, displaying the subclasses of “NonVegetarianPizza” (results inferred). It is using the code snippet from Fig. 3 to generate the pizza description boxes. The objective of the example store is to present the capabilities of the framework to Web developers. The framework itself can handle any OWL or OWL 2 ontology that is loaded as the knowledge base. The framework does not treat the pizza ontology different to any other knowledge base. To create a store using the Fly-By-OWL framework, a Web developer must upload their ontology and then create appropriate template modules using the data-model specifications and FTL scripting. This allows a Web developer to present the concepts within the knowledge base however they best envision.



Reasoning by FaCT++ version 1.3.0
Time to read ontology: 235ms - Time to reason: 7ms
Time to generate data-model: 441ms - Total time 703ms (20ms misc.)

Our Pizzas
Browse the categories and make your selection or view them all

NonVegetarianPizza

Cajun
> NamedPizza
> hasTopping only (PrawnsTopping or TobascoPepperSauce or MozzarellaTopping or TomatoTopping or OnionTopping or PeperonataTopping)
> hasTopping some MozzarellaTopping
> hasTopping some OnionTopping
> hasTopping some PeperonataTopping
> hasTopping some PrawnsTopping
> hasTopping some TobascoPepperSauce
> hasTopping some TomatoTopping

FruttiDiMare
> NamedPizza
> hasTopping only (GarlicTopping or TomatoTopping or MixedSeafoodTopping)
> hasTopping some GarlicTopping
> hasTopping some MixedSeafoodTopping
> hasTopping some TomatoTopping

Napoletana
> NamedPizza
> hasCountryOfOrigin has Italy
> hasTopping only (AnchoviesTopping or MozzarellaTopping or TomatoTopping or CaperTopping or OliveTopping)
> hasTopping some AnchoviesTopping
> hasTopping some CaperTopping
> hasTopping some MozzarellaTopping
> hasTopping some OliveTopping
> hasTopping some TomatoTopping

MeatyPizza

American
AmericanHot
Capricciosa

Figure 5: “Semantic Pizza Store” viewing the NonVegetarianPizza class (results inferred)

Fly-By-OWL allows the creation of “ontology-driven” e-commerce websites while emphasising its practical relevance to web developers. During development, the pages output to a user were all generated in under one second. The lengthiest operations within a page load are reading the ontology itself and the generation of the data model. While Web application platforms are generally tried-and-tested, the "ontology-driven" with live reasoning concept is new and further research is required to understand how it will cope under various loads, and the hardware/ software setup required to best handle such traffic. Performance monitoring a Fly-By-OWL store under various traffic loads has been identified as a further research topic. The size of the ontology being used as the knowledge base and it's affect on load times will also be studied. With further development of the platform's architecture, load times are intended to be comparable to any database-driven Web application.

6. Research agenda

With the first implementation of Fly-By-OWL focused towards an e-commerce context, research questions arise in regards to how the "ontology-driven" concept will cope against traditional tried-and-tested back-ends (e.g. a DBMS). A number of research questions arise in regards to its real-world usage and further development.

- How are page load times affected by traffic?
- How are page load times affected by size of the knowledge base ontology?
- What types of ontology will be used as the knowledge base? E.g. GoodRelation's vocabulary and how will they be handled by Fly-By-OWL?
- How should the platform be expanded beyond the e-commerce context and become more applicable for general Web applications?
- What kind of Web services can be created by using XML templates with FTL scripting and how should they harness reasoning?

7. Conclusion

This paper has presented Fly-By-OWL, a framework that enables the creation of "ontology-driven" Web applications, with its first implementation aimed specifically towards e-commerce stores. By following calls for "rapid simplification" of the Semantic Web, this paper aims to make ontologies relevant to Web developers. By starting with an analysis of ontology usage within six popular LAMP based Web applications, it became apparent that the developers of those applications found no place for ontologies within any standard or road mapped feature. While the Semantic Web community was active in creating add-on's for those applications that allowed them to make use of ontologies in some form, the scope of their depth was trivial and they did not play any role vital to the operation of the applications. With Fly-By-OWL, e-commerce store's can be created for the present day Web that are driven by an ontology. With live reasoning, present day limitations experienced when using a database can be overcome. The paper demonstrated some features such as using inferences to dynamically categorise products and customising products without hard coded rules but verifying consistency. The paper also showcased the "Semantic Pizza

Store", an example e-commerce store based upon the pizza ontology tutorial created to demonstrate the framework. A research agenda has been formulated that looks to address some of the initial questions posed by the "ontology-driven" concept and how to further Fly-By-OWL's overall relevance and scope.

8. Project on the Web

The Fly-By-OWL project website and the "Semantic Pizza Store" are hosted at Brunel University, U.K. and can be found online at http://www.flybyowl.org. You are welcome to use the framework to create your own "ontology-driven" e-commerce stores and interact with our online community.

9. References

[1] P. Warren, The next steps for the web: Putting meaning into the web. *Computing & Control Engineering 1(2),* pp. 27-35, 2003.

[2] D. Bradbury, Unweaving the tangled web of dumb data. *Computer Weekly 10*pp. 1-7, 2003.

[3] T. Berners-Lee, J. Hendler and O. Lassila, "The Semantic Web," *Sci. Am.*, May. 2001.

[4] B. McBride, "Four steps towards the widespread adoption of a semantic web," in *ISWC '02: Proceedings of the First International Semantic Web Conference on the Semantic Web,* 2002, pp. 419-422.

[5] M. Horridge, H. Knublauch, A. Rector, R. Stevens and C. Wroe, "A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plug-in and CO-ODE Tools Edition 1.0," August. 2004.

[6] Netcraft. (2009, 07/28/2009). July 2009 Web Server Survey. *2009(08/10),* Available: http://news.netcraft.com/archives/2009/07/28/july_2009_web_server_survey.html

[7] T. O'Reilly, "Lessons from open-source software development," *Commun ACM,* vol. 42, pp. 32-37, 1999.

[8] G. Lawton, "LAMP lights enterprise development efforts," *Computer,* vol. 38, pp. 18-20, Sept. 2005.

[9] I. Horrocks, "Semantic web: The story so far," in *W4A '07: Proceedings of the 2007 International Cross-Disciplinary Conference on Web Accessibility (W4A),* 2007, pp. 120-125.

[10] R. McCool, "Rethinking the Semantic Web, Part 2," *IEEE Internet Comput.,* vol. 10, pp. 96-95, 2006.

[11] J. G. Breslin, S. Decker, A. Harth and U. Bojars, "SIOC: an approach to connect web based communities," *Int.J.Web Based Communities,* vol. 2, pp. 133-142, 2006.

[12] M. Hepp, "GoodRelations: An ontology for describing products and services offers on the web," in *EKAW '08: Proceedings of the 16th International Conference on Knowledge Engineering,* 2008, pp. 329-346.

[13] L. Stojanovic, N. Stojanovic and R. Volz, "Migrating data-intensive web sites into the semantic web," in *SAC '02: Proceedings of the 2002 ACM Symposium on Applied Computing,* 2002, pp. 1100-1107.

[14] Y. Lei, E. Motta and J. Domingue, "OntoWeaver: An ontology-based approach to the design of data-intensive web sites," in *Hypertext/Hypermedia Handbook,* 2005, pp. 244-262.

[15] Y. Jin, S. Decker and G. Wiederhold, "OntoWebber: Model-driven ontology-based web site management," in *Semantic Web Working Symposium (SWWS),* 2001.

[16] M. Hepp, D. Bachlechner and K. Siorpaes, "OntoWiki: Community-driven ontology engineering and ontology usage based on wikis," in *WikiSym '06: Proceedings of the 2006 International Symposium on Wikis,* 2006, pp. 143-144.

[17] R. McCool, "Rethinking the Semantic Web, Part 1," *IEEE Internet Comput.,* vol. 9, pp. 88-87, 2005.

[18] M. Horridge, N. Drummond, J. Goodwin, A. L. Rector, R. Stevens and H. Wang, "The Manchester OWL Syntax," in *OWLED; CEUR Workshop Proceedings,* 2006.

[19] FreeMarker Project, "FreeMarker: Java Template Engine Library," 2009(08/10), Available: http://www.freemarker.org

How To Simplify Building Semantic Web Applications

Matthias Quasthoff, Harald Sack, Christoph Meinel

Hasso Plattner Institute, University of Potsdam
{matthias.quasthoff, harald.sack, meinel}@hpi.uni-potsdam.de

Abstract. This paper formalizes several independent approaches on how to develop Semantic Web applications using object-oriented programming languages and Object-Triple Mapping. Using such mapping, Semantic Web applications have been developed up to three times faster compared to traditional Semantic Web software engineering. Results show that at the same time, developer satisfaction has been significantly higher if they used object triple mapping. We present a formal notation of object triple mapping and results of an experimental evaluation clearly showing the benefits of such mapping. The work presented here may one day help to make Semantic Web technologies part of the majority of future applications.

1 Introduction

Using and handling RDF data in software is not trivial to implement, especially with regards to the large number of best practices to consider. Before even that, developers need to learn about RDF concepts and how to deal with them using RDF programming libraries [1]. For many tasks, additional knowledge about RDF schema and OWL is required. This makes getting started with the Semantic Web quite a challenge for Semantic Web beginners and entry-level developers.

In general, two issues with current Semantic Web programming libraries can be identified: First, their complete potential is always visible to developers, instead of by default only revealing those parts that would be sufficient for a majority of implementation problems. Second, lots of RDF- and Linked Data-related implementation tasks, such as discovery, retrieval, and publishing of datasets need to be implemented by hand, resulting in huge implementation efforts even for relatively small implementation problem (cf. Section 4).

A promising approach for simplifying Semantic Web software development is Object Triple Mapping (OTM) [2, 3]. There do exist some OTM implementations, and also some research on its expressivity (cf. Section 2). However, there is no research on the actual building blocks needed to develop Semantic Web applications on top of existing RDF programming libraries, and there is no research on whether, or how OTM actually simplifies application development. With our work, we are confident to lay out guidelines for the development of future, easy-to-use Semantic Web programming libraries.

In Section 3, wie formalize OTM and extend it by a small pseudo-code vocabulary describing linked data functionality. This formalization is a starting point for further research in this important area. We used the formalization for an experimental evaluation of the benefits of OTM. The results of this evaluation are presented in Section 4. Section 5 concludes the paper.

2 Related work

Linked Data has become one of the most popular topics among the emerging Semantic Web [4]. Best practices need to be identified and described, how to efficiently implement linked data applications. Design patterns formalize such best practices; not as programming libraries, but as solutions to frequently recurring problems. They have been introduced to software engineering by [5]. In the world of relational database management systems, widely-used design patterns on object-relational mapping (ORM) have been identified [6]. These patterns allow developers to simply instantiate objects, and they are automatically filled with contents from a relational database. Modifications to these objects will automatically be persisted in the database.

The need of simplifying Semantic Web software engineering in a similar way to ORM has been identified in [2], and an implementation called So(m)mer, based on meta-programming, is available[1]. Up to now, several other tools for OTM have been inspired by object-relational mapping. The D2RQ Platform [7] uses a declarative language to describe mappings between relational database schemata and Semantic Web ontologies and lets clients access RDF views on the underlying non-RDF data. Other approaches such as RDFReactor[2] take mappings between OO programming units and Semantic Web schemas and allow software developers simplified access to a triple store. OntoJava [8] uses a similar approach to use auto-generated inference rules inside application source code. Winter [9] extends So(m)mer to allow for mappings of complex patterns instead of plain RDF classes onto Java classes.

All these solutions and the research done around them focus on specific parts of the approach chosen, such as feasibility and expressivity of such mapping, but do not yet aim at supporting the whole process of discovering, retrieving, processing, and re-publishing linked data, which will involve further steps such as, e.g., policy or data license checking [10]. To overcome this situation we have identified the principles common to these solutions and formulated design patterns for OTM, closely resembling ORM design patterns [3]. As our contribution in this paper, we will formalize the design pattern and evaluate how actual software can be built upon it.

[1] https://sommer.dev.java.net/
[2] http://semanticweb.org/wiki/RDFReactor

3 Formalizing OTM

It is good practice to encapsulate business or domain logic in classes and methods of object-oriented (OO) programming languages [6]. E.g., if a software product deals with people and relations between them, the software's object model likely contains a `Person` class and `friends` field for this class. To use RDF data in most OO programming languages, the mapping from RDF properties to the domain classes' fields has to be implemented by hand. Our hypothesis is that large parts of this OO handling of RDF concepts, including discovery and retrieval on the WWW, should be hidden from software engineers, making the development of Semantic Web software much easier, and hence encouraging software developers to actually start creating such software. In the following sections, we introduce a formal notation of the knowledge representation in OO programming, a mapping between RDF and OO concepts, and a simple pseudo-code vocabulary relevant for building applications using such mapping.

3.1 Basic concepts

The RDF data model has an established formal notation building upon the following concepts [1].

Definition 1 (RDF data model) *Let U be the set of* URI references*, B an infinite set of* blank nodes*, and L the set of* literals*.*

- $V := U \cup B \cup L$ *is the set of RDF* nodes*,*
- $R := (U \cup B) \times U \times V$ *is the set of all* triples *or* statements*, that is, arcs connecting two nodes being labelled with a URI,*
- *any $G \subseteq R$ is an* RDF graph*.*

How RDF graphs are actually constructed, handled, and transferred is subject to standards, conventions, and technical constraints. Linked data principles [4] suggest to provide smaller sub-graphs describing individual resources. In [11], an abstraction on top of these principles is described providing the whole Web of Data as one huge graph. Hence, operating on RDF data involves not only operating on triples and resources, but also retrieving the right sub-graphs of R, which will be described in a later section.

For OO programming, there is no single established formal notation focusing on the information representation part. Hence, we just use some basic formal concepts to capture OO environments form the information representation perspective.

Definition 2 (OO data model) *Let O be a set of* object identifiers*, F a set of* field names*.*

- $S := \mathcal{P}(O)^F$ *is the set of* field assignments $s : F \rightarrow \mathcal{P}(O)$*,*
- $Q := S^O$ *the set of* system states $q : O \rightarrow S$*.*

A system state $q \in Q$ maps each object $o \in O$ onto a field assignment $s := q(o) \in S$, which in turn maps each field name $f \in F$ onto the object's values $s(f) \subseteq O$ for this field. Note that in our notation, a field assignment returns sets of objects as field values. This allows to represent programming concepts such as array or collection objects. Ordered lists and formal cardinality and type restrictions (i.e., scalar-value fields or statically typed object definitions) on OO data models are outside the scope of this paper but can easily represented on top of our formalism if needed.

Example 1 (comparison of RDF and OO data model) *Let* p_1, p_2, $p_3 \in U$ *be URI denoting three people,* $n \in U$ *be the URI* `foaf:name` *and* $k \in U$ *be the URI* `foaf:knows`. *An RDF graph describing* p_1 *and* p_2 *might look the following.*

$$G := \left\{ \langle p_1, n, \text{“John Doe”} \rangle,\ \langle p_1, k, p_2 \rangle,\ \langle p_1, k, p_3 \rangle,\ \langle p_2, n, \text{“Jane Doe”} \rangle \right\}$$

Let us now have a look at this example from the OO perspective. Let o_1, o_2, $o_3 \in O$ *be object identifiers denoting three people,* `name`, `friends` $\in F$ *be field names,* $q \in Q$ *a system state, and* $s_1 := q(o_1)$, $s_2 := q(o_2)$. *The OO representation of* G *will look the following.*

$$\begin{aligned} s_1(\texttt{name}) &= \{\text{“John Doe”}\} \\ s_2(\texttt{name}) &= \{\text{“Jane Doe”}\} \\ s_1(\texttt{friends}) &= \{o_2, o_3\} \end{aligned}$$

3.2 Mapping RDF and OO

In this sections we continue to use the set definitions from the previous section.

Definition 3 (Object triple mapping, OTM) *An* object triple mapping *for an RDF Graph* R, *fields* F *and objects* O *is some* (G, m_t, m_a, q) *such that*

- $G \subseteq R$ *is an RDF graph*
- $m_t : F' \to U$ *for mapped fields* $F' \subseteq F$ *(the vocabulary map),*
- $m_a : O' \to U$ *for mapped objects* $O' \subseteq O$ *(the instance map)*
- $q \in Q$ *a system state such that for all* $u \in U$, $o \in O'$, $f \in F'$ *and* $s := q(o)$
  - $|m_a^{-1}(u) \cap s(f)| \leq 1$
  - $|m_a^{-1}(u) \cap s(f)| = 1 \Leftrightarrow \langle m_a(o), m_t(f), u \rangle \in R$

Note that this definition does not require the instance map m_a to be injective, which would be desirable in many cases, at least from a software engineer's point of view. However, there might be different simultaneous OO representations o_i of a single RDF resource u resulting from, e.g., different data licenses, trust policies, or access control decisions. Hence, the injectivity of the actual instance map presented to the developer should rather be ensured using additional formal representations of policies, contexts and the like, instead of being a general requirement to the instance map. Also, our notion of OTM does not necessarily require a class map for RDF and OO, since many dynamically-typed object-oriented programming languages do not have the notion of classes. For statically-typed programming languages, an implementation of such class map will however be required. Treating other RDF concepts such as lists or reification, and also the semantics of RDFS and OWL are not part of this mapping, but subject to OTM implementations.

Example 2 (OTM) *To map RDF representations of people to the corresponding OO representation (cf. example 1), we need*

- *a vocabulary map* $m_t : \texttt{name} \mapsto n, \texttt{friends} \mapsto k$;
- *mapped objects* $o'_1, o'_2, o'_3 \in O'$ *such that* $m_a : o'_1 \mapsto p_1, o'_2 \mapsto p_2, o'_3 \mapsto p_3$;
- *a system state* $q \in Q$ *and field assignments* $s'_1 := q(o'_1)$ *and* $s'_2 := q(o'_2)$.

The mapped objects o'_1, o'_2 *will exactly look like* o_1, o_2 *from example 1:*

- $s'_1(\texttt{name}) = \{\text{“John Doe”}\}$ *because* $\langle \underbrace{m_a(o'_1)}_{=p_1}, \underbrace{m_t(\texttt{name})}_{=n} \text{ “John Doe”}\rangle \in G$
- $s'_2(\texttt{name}) = \{\text{“Jane Doe”}\}$ *because* $\langle \underbrace{m_a(o'_2)}_{=p_2}, \underbrace{m_t(\texttt{name})}_{=n} \text{ “Jane Doe”}\rangle \in G$
- $s'_1(\texttt{friends}) = \{o'_2, o'_3\}$, *because* $\langle \underbrace{m_a(o'_1)}_{=p_1}, \underbrace{m_t(\texttt{friends})}_{=k}, \underbrace{m_a(o'_i)}_{=p_i}\rangle \in G$.

Although several implementations of such mapping exist (cf. Section 2), for our research we use our own OTM implementation, which is available licensed under the GPL[3]. This implementation strictly follows the OTM design patterns derived from object-relational patterns [3, 6].

3.3 Building Linked Data software

Building a linked data application using OO programming will involve handling RDF resources as OO objects. There are only two ways to obtain objects from resources, and we introduce the following pseudo-code notation for them:

- $\texttt{get}(u)$ for $u \in U$: Request $o \in O'$ such that $m_a(o) = u$;
- `query(pattern)`: Request $O'' \subseteq O'$ matching a `pattern`, using SPARQL.

From Definition 3, it is not clear how the RDF graph G is obtained. G will be constructed during application execution using the follwoing ways.

- Directly query a triple store, e. g., using SPARQL,
- $\texttt{load}(u)$ for $u \in U$: Ensure that the dereferenced graph $G_u \subseteq G$,

Following linked data principles, an OTM implementation can automatically call $\texttt{load}(u)$ on occurences of $\texttt{get}(u)$. Just as this simplifies resource handling, two more pseudo-code operations are required to build linked data applications.

[3] http://projects.quasthoffs.de/otm-j

– `use`(u) for $u \in U$: Set up the OTM implementation to use the dataset u, i. e. evaluate the data license, and set up the SPARQL endpoint to be used. Even further decisions can be made, such as deciding upon some statistics whether to dereference single URIs for this dataset or rather to download a data dump.
– `publish`(O'') for $O'' \subseteq O$: Publish objects as linked data, either by producing serialized RDF files, or by hooking into some Web programming framework. By configuring meta-data for publication such as licenses, several checks, e. g. on license compatiblity, can be performed.

4 Evaluation

Our primary motivation for investigating OTM is to understand why Semantic Web technologies have been picked up so hesistatingly by software developers, and to show software developers how they can simply use and benefit from these technologies. We asked software engineers with little or no experience in Semantic Web software engineering (but yet sufficient programming skills) to solve a problem using RDF data sources and programming libraries.

4.1 Setup

Each participant was assigned two tasks, one of which was to be solved without OTM, and the other one using OTM. The order of the two tasks and the order of using/not using OTM was randomized to ensure the results will not be distorted by learning effects. Participants used the Eclipse programming environment and the jUnit framework to test their results[4]. To simulate the usual work-flow of Web programmers, we provided the participants with example source code of similar solutions [12].

Tasks. The experiment dataset consisted of 12,726 fictious *foaf:Person* resources, 100 *foaf:Document* resources, 15 *foaf:Group* resources, and 169 *bldg:Room* resources[5]. Each document had between 2 and 4 authors, each group between 8 and 15 members, and each person knew a number of other people. All resources could be dereferenced by their URI. The following tasks needed to be solved.

Task 1. Given a set of URI identifying documents, construct the set of all the documents' authors' names.
Task 2. Given a URI identifying a person, construct the set of all person's friends' friends' names.

The participants were expected to find a solution close to the following pseudocode (using the vocabulary from Section 3.3), which was however not presented to the participants.

[4] http://eclipse.org/, http://junit.org/
[5] *foaf:* http://xmlns.com/foaf/0.1/, *bldg:* http://example.org/buildings/

Solution 1. `GET_AUTHOR_NAMES(publication_uris):`

```
load(dataset_uri)
for each uri in publication_uris
    publication = get(uri)
    for each person in publication.authors
        return person.name
```

Solution 2. `GET_SECOND_ORDER_FRIENDS(person_uri):`

```
load(person_uri)
person = get(person_uri)
for each friend in person.friends
    load(friend.uri)
    for each friend2 in friend.friends
        load(friend2.uri)
        return friend2.name
```

4.2 Metrics

Difficulty. After each of the two assignments, participants were asked to *estimate the difficulty* of the assigment, the *maintainability of the resulting source code*, and how difficult a solution would have been using *XML stores* or *RDBMS* instead of RDF, on a scale from 0 (trivial) to 10 (too hard). After the first assignment only (which randomly had to be solved either using OTM or without OTM), participants were asked whether they see potential use of RDF in their near-future projects. Along with these subjective measures, we tracked the *time* required to find a working solution, and the number of *edit-debug cycles*.

Source code. Since OTM encapsulates large parts of RDF data handling it is expected that solutions building on OTM will have less lines of code than solutions than non-OTM solutions. To get a deeper understanding of how lines of code will be reduced, we separately counted lines of code carrying

- *language constructs* such as loops, variable declarations etc.;
- *RDF library initialization code*, i. e. creation of or connection to data stores;
- *data access code*, i. e. imperative statements for URI handling such as `get`, `load`, `use`, or `query` operations, (cf. Section 3.3), and RDF concept manipulation using RDF libraries; and
- *business* or *domain logic*, i. e. lines of code manipulating domain objects such as people, publications or names, e. g., by handling or accessing fields of domain objects.

Feedback. Besides these rather quantitative metrics, we gathered feedback during and after the experiment. In a questionnaire, we asked the participants to identify the biggest problems during finding the solutions, and to name other types of support they wished they had.

4.3 Participants

Undergraduate computer science students at Hasso Plattner Institute, University of Potsdam were invited to to participate in the experiment. Ten participants aged 19 to 27 (mean 22.6) actually took part in the experiment. All participants had between 4 and 9 years of programming experience (mean 6.6). According to their estimation on a scale from 0 (none) to 10 (expert) prior to solving the assignments, only three of them said having some basic knowledge about RDF, the others none (mean 0.9). All participants were experienced with the Java programming language (mean 5.4). Also, most participants had some experience using traditional information stores such as XML documents (mean 3.4) and RDBMS (mean 3.9).

4.4 Results

Difficulty. The results for developer satisfaction were surprisingly clear (Fig. 1). Implementing the assignments was found to be significantly easier using OTM (mean 2.4, $\sigma = 1.8$) than using the Jena RDF library only (mean 6.4, $\sigma = 2.1$). Also, the participants judged their solution significantly easier to maintain (both for themselves: 1.13, $\sigma = 1.6$, and if they let somebody else do it: 2.13, $\sigma = 1.5$), if OTM had been used compared to non-OTM (means 4.3, $\sigma = 1.8$ and 5.9, $\sigma = 1.1$). However, whether the first assignment was to be solved using OTM or without OTM had no influence on the participants' estimation of the difficulty of integrating Semantic Web technologies in their own future projects (means 4 and 5). Regarding the estimated difficulties, we can eliminate variance by comparing the differences of the difficulty of the implementation and the estimated difficulties of alternative approaches using XML or RDBMS (Fig. 2). By means, the non-OTM solution has been rated more difficult compared to traditional data formats, whereas the OTM solution has been rated easier to implement than traditional data formats. However, the differences for these relative difficulties are not significant. For Task 1 (finding the names of publication authors), both the number of *edit cycles* (non-OTM mean 29, $\sigma = 6.6$) and the time needed to find the solution (mean 1.2 hours, $\sigma = 0.5$) was significantly lower using OTM (8.8 edit-debug cycles, $\sigma = 6.2$ and 0.4 hours, $\sigma = 0.2$, Fig. 3). For Task 2 (finding the names of friends' friends), the mean number of edit-debug cycles was increased from 8 to 19, while the mean time needed to find the solution was decreased from 0.6 hours ($\sigma = 0.3$) to 0.4 hours ($\sigma = 0.4$) using OTM. However, the differences for Task 2 are not significant. The combined figures for Task 1 and Task 2 show that the number of edit-debug cycles remains stable, but the development time has been decreased significantly using OTM. It is unclear why the number of edit-debug cycles is larger for Task 2 using OTM. But since the development time was not increased, we do not consider this a general weakness of OTM. It will however be interesting to direct further research in this direction.

Source code. The figures for the source code metrics are clear again (Fig. 4). The overall lines of code are reduced from 20.9 ($\sigma = 5.4$) not using OTM

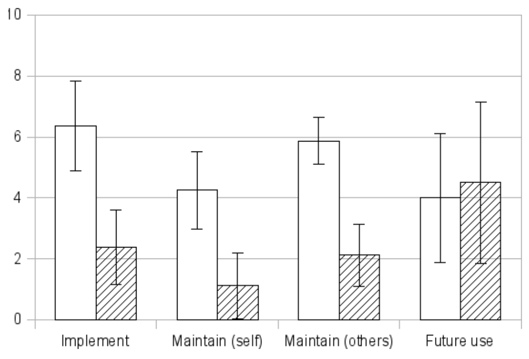


Fig. 1. Difficulty (0 – easy, 10 – too hard) of *implementing the solution*, and estimated difficulty of *maintaining the result's source code* and *using Semantic Web technologies in future projects* for non-OTM (white) and OTM (streaked). Error bars show 95% CI.

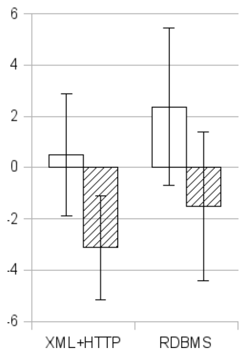
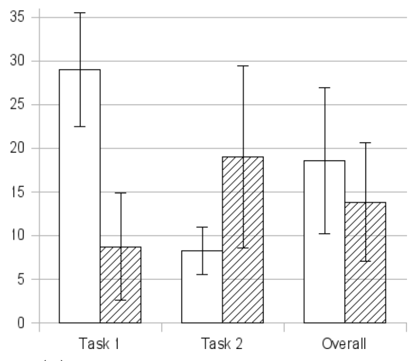
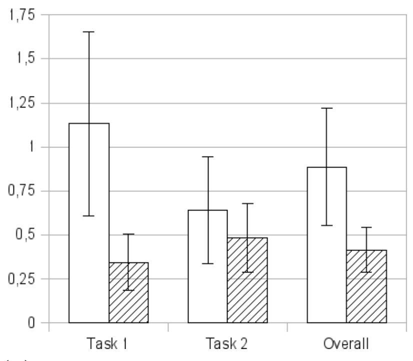


Fig. 2. The mean difference of the difficulty of *implementing the solution* and the and estimated difficulties of implementing alternative solutions using *XML and HTTP* or *RDBMS* is higher for non-OTM (white) solutions than for the OTM solutions (streaked). Error bars show 95% CI.

to 11.3 ($\sigma = 7.7$) using OTM. The number of lines of code carrying language constructs (mean 5.1 for non-OTM and 4.4 for OTM) and lines carrying business and domain logic (means 5.1 and 3.5) remain about the same, no matter if OTM is used or not. But lines of code for library initialization (non-OTM mean 4.8, $\sigma = 1.8$) and data access (12, $\sigma = 5.4$) are reduced significantly to 1.5 ($\sigma = 1.4$) and 2.8 ($\sigma = 3.9$) if OTM is used. This is plausible as using OTM no objects simply representing vocabulary (such as Jena's `Property`) or data access interfaces (such as Jena's `StmtIterator`) need to be instantiated. Additionally, using our OTM implementation the `load` operation can be omitted, as on calls to `get` and on field access `load` is called automatically. However, the main benefit of these implicit calls is not reduced lines of code, but improved separation of concerns, i. e. data access is separated from domain logic.



(a) number of edit-debug cycles



(b) time in hours needed to find solution

Fig. 3. Depending on the implementation task, the number of *edit-debug cycles* can be reduced significantly using OTM (streaked), compared to OTM (white), 3(a). OTM can also significantly reduce the implementation time of Task 1. In spite of increased number of edit-debug cycles for Task 2, the time needed to find the solution is not increased, 3(b). Error bars show 95% CI.

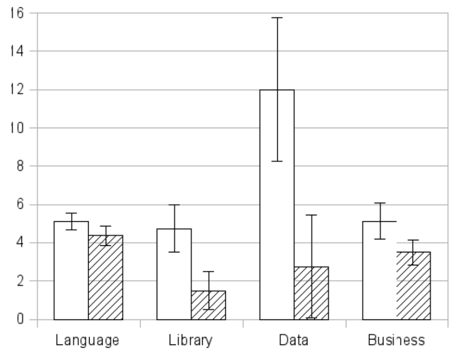


Fig. 4. The significant decrease in lines of code using OTM is achieved by reducing *library initialization* and *data access* code. Error bars show 95% CI.

Qualitative Feedback. Asked for the biggest problems they faced solving the non-OTM tasks, participants found it generally hard to understand the Jena API. Participants found it difficult to understand the central Jena classes `Model`, `Resource`, and `Property`, which is due to their lacking knowledge of RDF concepts. Also, participants had problems understanding `Model.read`, which loads RDF data from the URI specified, and `Model.getResource`, which only creates a `Resource` object to be further processed in Java. Some were unsure when a `String` in Jena was a literal value, and when it represented a URI. One participant commented "The source code contains weird objects, which do not have to do anything with the problem domain. This causes programming mistakes, because these objects are untyped." He mentioned the example of deciding whether `RDFNode` is a `Literal` or a `Resource`. Also, Jena's `StmtIterator`, which needs to be used in order to read triple information, was criticized for being confusing to use. All in all, participants highly valued the source code examples we provided, and said finding a solution would have taken much longer without the example. However, some would have preferred more comprehensive examples, featuring nested loops, or complete howto documents.

OTM received less comments, which is probably due to the fact that each participant was to solve two assignments–one using OTM, one without–and the OTM task was easier to solve than the non-OTM task. Still, we received valuable feedback. Participants found it hard to find the mapped Java class representing resources of a specific RDF type. Some participants have been observed analyzing all mapped Java classes available, others just read the example source code provided and concluded the right classes to use. However, one participant just guessed arbitrary (wrong) classes to be used in his source code and got stuck for a while. Although our OTM implementation simplifies URI dereferencing by implicitly loading RDF graphs when instantiating mapped objects, it took one participant a while to find out how to explicitly load a whole dataset as needed for Task 1. Some participants had trouble in dealing with generic Java types used for collections of objects, and hence could not fully benefit from our OTM framework.

Both the OTM and non-OTM solutions shared some comments. Participants said a graphical representation of the RDF schema, or the mapped class model, and a graphical browser for the dataset would have helped them to understand the data structures and would have improved their implementation performance. Also, participants complained about not having understood how or where the data had actually been stored. Only very few participants had the idea of viewing the URI provided by the test framework in a Web browser window.

5 Conclusion

In this paper, we presented a formalism for Object Triple Mapping (OTM), a promising approach to structuring the development of Semantic Web software. Our OTM formalism harmonizes several implementions seeking to simplify Semantic Web application development and adds process elements to describe complete programs operating on linked data. Our second contribution is an experimental evaluation of OTM. We presented the results of this experiment, clearly showing that

- OTM speeds up the development of Semantic Web software.
- Lines of code needed to solve several tasks are reduced by half using OTM, and the share of "purely technical" lines of code is diminished so that using OTM, business logic and program structure stands out in the code.
- Improved programming experience can be measured, as developers without Semantic Web programming experience find it simpler to develop software using OTM, and are more satisfied with the quality of their results.

The experiment material, assignments, datasets etc. can be downloaded from the experiment web site[6]. We encourage readers to join the evaluation, and share their results with us. To obtain a broader view on what are the Semantic Web software engineers' pains, how we can help them, and which technology they actually prefer, we will extend our evaluation to more programming languages and RDF programming libraries. Besides this planned continuous evaluation, we will publish the direct and indirect feedback we receive from participants, and will incorporate that feedback into our own OTM implementation for further evaluation, and are willing to contribute to other existing OTM implementations.

As the results of our experiment are very promising, we are confident to contribute in further spreading the word about positive experience using Semantic Web standards and technologies. Once software engineers and managers are convinced that Semantic Web technologies can be introduced in software projects without adding costs, or even reducing costs, software will start to contain more and more Semantic Web technologies, fostering interoperability and data mash-ups. By then, software engineers will be willing to learn more about these technologies and more complex software projects going beyond the features of off-the-shelf OTM implementations can finally be done.[7]

References

1. Manola, F., Miller, E.: Rdf primer. w3c recommendation 10 february 2004. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/ (2004)
2. Story, H.: Java annotations and the semantic web. http://blogs.sun.com/bblfish/entry/java_annotations_the_semantic_web (2005)
3. Quasthoff, M., Meinel, C.: Design patterns for the web of data. In: Proc. of IEEE SCC 2009. (2009)
4. Berners-Lee, T.: Linked data. http://www.w3.org/DesignIssues/LinkedData.html (2006)
5. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley (1994)

[6] http://hpi-web.de/meinel/quasthoff/otm-experiment-2009-06

[7] For the cited work still to appear ([13] and [3]), please find the electronic version at http://hpi-web.de/meinel/quasthoff.

6. Fowler, M., Rice, D.: Patterns of Enterprise Application Architecture. Addison-Wesley (2003)
7. Bizer, C., Seaborne, A.: D2rq – treating non-rdf databases as virtual rdf graphs. In: Proc. of the 3rd International Semantic Web Conference, Springer (2004)
8. Eberhart, A.: Automatic generation of java/sql based inference engines from rdf schema and ruleml. In: Proc. of the 2nd International Semantic Web Conference, Springer (2002)
9. Saathoff, C., Scheglmann, S., Schenk, S.: Winter: Mapping rdf to pojos revisited. In: Proccedings of the ESWC 2009 Demo and Poster Session. (2009)
10. Miller, P., Styles, R., Heath, T.: Open data commons, a licence for open data. In: Proceedings of the WWW2008 Workshop on Linked Data on the Web, Springer (2008)
11. Hartig, O., Bizer, C., Freytag, J.C.: Executing sparql queries over the web of linked data. In: Proc. of the 8th International Semantic Web Conference, Springer (2009)
12. Brandt, J., Guo, P.J., Lewenstein, J., Klemmer, S.R.: Opportunistic programming: How rapid ideation and prototyping occur in practice. In: Proc. of the Fourth Workshop on End-User Software Engineering, ACM (2008)
13. Quasthoff, M., Sack, H., Meinel, C.: Can software developers use linked data vocabulary? In: Proc. of I-Semantics '09. (2009)

Ontology-Driven Software: What We Learned From Using Ontologies As Infrastructure For Software *Or How Does It Taste to Eat Our Own Dogfood*

Csongor Nyulas, Natalya F. Noy, Michael Dorf, Nicholas Griffith, Mark A. Musen

Stanford University, Stanford, CA 94305, US
{nyulas,noy,mdorf,ngriff,musen}@stanford.edu

Abstract. In recent years, researchers have argued that using ontologies to represent and drive knowledge infrastructure of software projects provides separation of the declarative and procedural knowledge and enables easier evolution of the declarative knowledge. We have validated these conjectures in the context of BioPortal, a repository of biomedical ontologies, which was developed in our group. We are using the BioPortal Metadata Ontology to represent details about all the ontologies in the repository, including internal system information and the information that we collect from the community such as mappings between classes in different ontologies, ontology reviews, and so on. To the best of our knowledge, BioPortal is the first large-scale application that uses ontologies to represent essentially all of its internal infrastructure.
The BioPortal Metadata Ontology extends several other ontologies for representing metadata, such as the Ontology Metadata Vocabulary and the Protégé Changes and Annotations Ontology. In this paper, we show that it is feasible to describe the structure of the data that drives an application using ontologies rather than database schemas, which are used traditionally to store the infrastructure data. We also show that such approach provides critical advantages in terms of flexibility and adaptability of the tool itself. We demonstrate the extensibility of the approach by enabling representation of views on ontologies and their corresponding metadata in the same framework.

1 Representing Knowledge Infrastructure: From Database Schemas to Ontologies

The topic of using Semantic Web technology to facilitate software development, integration, and evolution has been an active area of Semantic Web research, with annual workshops on Semantic-Web Enabled Software Engineering (SWESE). Researchers have pursued several different directions in this line of work: generating software code from ontologies [7] or using ontologies to describe inputs, outputs, or tasks of software components to enable integration of software and services [12]; facilitating gathering of requirements from domain experts [1]; using ontology-based reasoning to validate integrity and consistency of software models [2]; or facilitating critical software-engineering tasks, such as configuration management [16] and product management [8]. These novel applications, as well as the traditional ones, use ontologies to describe only some of the artifacts, whereas the structure of the rest of the data is reflected in a database schema. In this paper, we describe an approach to application development that pushes this envelope to use ontologies and ontology instances to represent essentially *all* data that the application requires in a single flexible framework, from declarative high-level descriptions of the data as in the examples above to internal system data.

For large-scale distributed architectures today, the development stack includes several technologies wrapped around a SQL database schema, such as persistence managers (e.g., Hibernate), a web server, and so on. When the database schema changes, these changes often need to be propagated through the development stack, thus making such changes expensive in distributed web-based applications. We have encountered this problem in developing BioPortal[1]—a community-based repository of biomedical ontologies, containing 170 ontologies with more than one million classes among them at the time of this writing. Users can submit their ontologies to BioPortal; search across all ontologies; browse the ontologies, their different versions, and the associated descriptions and provenance information; describe their ontology-related projects and link the descriptions to the ontologies; leave comments on classes and on ontologies; create mappings between concepts in one ontology and concepts in another ontology [11].

The BioPortal application is heavily *knowledge-driven*: most of what the users see when browsing BioPortal (in addition to the ontologies themselves), is some rendering of information that would traditionally be in a database. This internal information that drives the application includes the metadata about ontologies in the repository, such as ontology domain, authors, and other provenance information, as well as information on which property to use for preferred name and synonyms in each ontology, information on where the ontology itself resides in the system (e.g., the specific database table), when it was uploaded, the name of the administrator of the ontology in BioPortal, and so on. Some of this information (such as provenance) is intrinsic to the ontology artifact and is relevant outside of BioPortal; some information is internal system information.

Because the BioPortal application is novel in many of its aspects, our internal infrastructure continues to evolve constantly, as we understand better user requirements, learn what works and what does not, get new collaborators that would like to extend BioPortal in a certain way. With the knowledge infrastructure constantly in flux, we found that describing and representing the structure of the knowledge as a relational database schema did not provide the flexibility and quick adaptability that our users required. Making changes was cumbersome and put a bottleneck in the development of the software code. It also made it much harder for anyone to adapt the BioPortal code for their own purposes as the developers had to be familiar with the entire development stack (including Protégé, Java, Spring, Hibernate, and Ruby-on-Rails).

Thus, we decided to “eat our own dog food:” we developed an ontology to describe this infrastructure and represented the application data itself as ontology instances. Thus the whole BioPortal application is driven by ontologies and ontology instances. Note that while BioPortal is a repository of ontologies, the infrastructure that we describe is not specific to the artifacts represented in a repository. It will work for a repository of any other artifacts, not necessarily ontologies. To the best of our knowledge, BioPortal is the first large-scale application of this approach.

This paper makes the following contributions:

[1] `http://bioportal.bioontology.org`

- We developed an ontology to represent the infrastructure and run-time data of a large community-based ontology repository.
- We implemented the infrastructure of BioPortal using an ontology to represent most of the data required to drive the application.[2]
- We validated the extensibility of the approach by adding functionality to support flexibly representation of ontology views.

2 Types of Metadata in the Repository

The BioPortal ontology repository is an active ontology repository with a large user community that contributes its content and uses its web services in their applications. In addition to more than 170 ontologies, it currently contains multiple versions of these ontologies, submitted by their authors and almost one million mappings between concepts in the ontologies. There are descriptions of ontology-based projects, and notes and discussions on classes and ontologies. The BioPortal Resource index provides ontology-based access to several biomedical data sets available online (e.g. entries in GEO, ClinicalTrials.gov). All BioPortal functionality is supported by a rich metadata infrastructure, which includes the following types of metadata:

- **ontology metadata** describing the ontologies and their provenance and includes ontology name, domain, description and keywords, authors, license information, versions, references, and metrics such as the number of classes and properties;
- **mappings** between concepts, and metadata associated with mappings, such as how the mapping was created, whether it was created manually or computed automatically by a particular algorithm (and which one) and context for the mapping [10];
- **ontology reviews** are contributed by users as part of their evaluation of ontologies in BioPortal;
- **notes on classes** are user-contributed notes that can contain questions, comments, and suggestions, usually addressed to the authors of specific ontology classes;
- **projects** that use ontologies, described by BioPortal users;
- **user information** such as user profiles, information on who administers each ontology and each project description, who contributed notes, mappings, and reviews to BioPortal, and so on.

3 Functional and Architectural Requirements for Metadata Support

The BioPortal application dictates the following functional and architectural requirements for the metadata support:

Efficient and scalable support of BioPortal main functions: Any metadata infrastructure must support fast access to metadata, flexible querying of specific metadata items and their combination, and be scalable. We envision that the number of users, notes, projects, and mappings will grow significantly in the coming months.

[2] At the time of this writing, some data, such as mappings and user information, is still in database tables from our initial implementation of BioPortal.

Support for ontology versioning: Users can upload successive versions of their ontologies and explore any ontology version. There must be services that always resolve to the latest version of an ontology, with each ontology having a "virtual" location that always redirects to the latest version. Metadata referring to an ontology or its components (e.g., reviews, notes, mappings) must be attached to a specific version of an ontology.

Flexible evolution of the metadata schema: One of the key requirements for metadata support is its ability to adapt easily to new requirements and types of metadata. The types of metadata that an ontology repository requires is still an active area of research. Thus, the structure of the metadata and the specific properties change frequently. These changes to the schema describing the metadata must be easy to implement and roll out.

Customizability of the metadata schema: Some groups install their own versions of BioPortal software to support either a broader scope than just biomedicine (e.g., the Open Ontology Repository sandbox[3]) or to maintain a repository open only to a specific set of users (e.g., the Marine Metadata Initiative[4]). Developers that maintain these BioPortal installations usually customize the code to satisfy the local requirements. For example, the fields that describe an ontology and its provenance are different for different communities. Definitions of mappings and the associated metadata differ as well. The representation of metadata schema must make it easy for these developers to custom tailor what gets represented and what gets presented in the user interface.

Reuse of existing technologies and ontologies: Wherever possible, we would like to use existing technologies and standards for representing metadata. For example, the Ontology Metadata Vocabulary (OMV) [14] provides a vocabulary for describing ontologies. There are several ontologies and APIs for describing mappings (e.g., the alignment API [4] or the Protégé mapping ontology [10]). Reusing these ontologies enables us not only to use technologies that have already been tested but also to share the data represented using these ontologies. For example, by using OMV to represent ontology metadata, we can share these descriptions with other repositories that use OMV, such as Oyster [13] and Cupboard [3]. Similarly, the comments on ontologies that BioPortal users provide are useful for ontology authors when they evolve their ontology. In order for ontology authors to see these comments alongside the ontology classes in their favorite ontology-editing environment (such as Protégé). Thus, the comments must be represented in the format that an ontology editor, such as Protégé, can understand (e.g., the Protégé Changes and Annotations Ontology, CHAO [9]).

In our initial implementation of BioPortal we used a database schema to describe our metadata, with column names corresponding to metadata fields. This approach is fairly traditional for many large-scale implementations and supports the first two requirements in our list—efficient and scalable handling of metadata and support for ontology versioning. However, in our experience, this approach did not fare so well on other requirements.

Any time we needed to add a new metadata field, we had to change not only the database schema, but also the rest of the application stack (e.g., Hibernate) to reflect the

[3] `http://oor-01.cim3.net/home/release`
[4] `http://mmisw.org/or/`

change. These changes were time-consuming and cumbersome. Yet, the more we were developing BioPortal, the more features we were adding, the more often we needed to adjust the metadata representation. For instance, in addition to describing the ontologies themselves, we needed to add ontology views, to support ontology reviews along several different dimensions, and to represent a large set of ontology metrics—all of these requirements crystalized after the start of the development.

With the metadata schema encoded as a database schema, any customization of new BioPortal installation requires changes to the schema as well. And, as we mentioned earlier, this process is cumbersome and error-prone.

Reusing and sharing the metadata that we collect also was not straightforward: it is hard to find two applications that use the same database schema. Thus, in order to transform the metadata from our internal representation to the representation that another repository (e.g., Oyster) uses or to represent comments in a format that a tool such as Protégé would understand, we must write a script to export the data.

While none of these challenges are insurmountable, we decided to apply a completely different approach to representing metadata infrastructure in BioPortal to address these requirements. As we show in the remaining sections, this approach satisfied our requirements and proved to be extensible enough to support new requirements.

4 Architecture

Figure 1 shows the architecture of BioPortal. It is a traditional service-oriented layered architecture, with the front end (Ruby-on-Rails) accessing the backend information through RESTful services.[5] There are services to access ontology information (e.g., get information about a specific ontology, upload a new version, get a diff between two versions), concept-level services (e.g., get class definition), hierarchy services (e.g., get all subclasses of a class), search services (e.g., search for a term across all ontologies), and other services. The business logic tier implements these services in Java, using the Spring framework. This layer is the one that contains the logic to translate the internal metadata representation into responses to service request (e.g., a service may request a list of all versions for a specific ontology). Before we transitioned to the ontology-based approach, the database schema of the underlying relational database (mySQL) was reflected directly in the implementation of this layer and its metadata functions. In our current approach, the metadata structure is accessed through the Protégé ontology API. This API, in turn, does use a database to store the ontology and the instances. However, the schema of the database that the Protégé uses does not depend on the ontology itself. It is a single table that stores both the ontology and the instance information.[6]

The types of metadata and their properties are describe in the **BioPortal Metadata Ontology**.[7] The metadata values are Protégé instances and property values (see Figure 2 for an example). We use the Protégé API to access the ontology and instances.

[5] `http://bioontology.org/wiki/index.php/BioPortal_REST_services`
[6] `http://protege.cim3.net/cgi-bin/wiki.pl?JdbcDatabaseBackend`
[7] `http://bioportal.bioontology.org/virtual/1148`

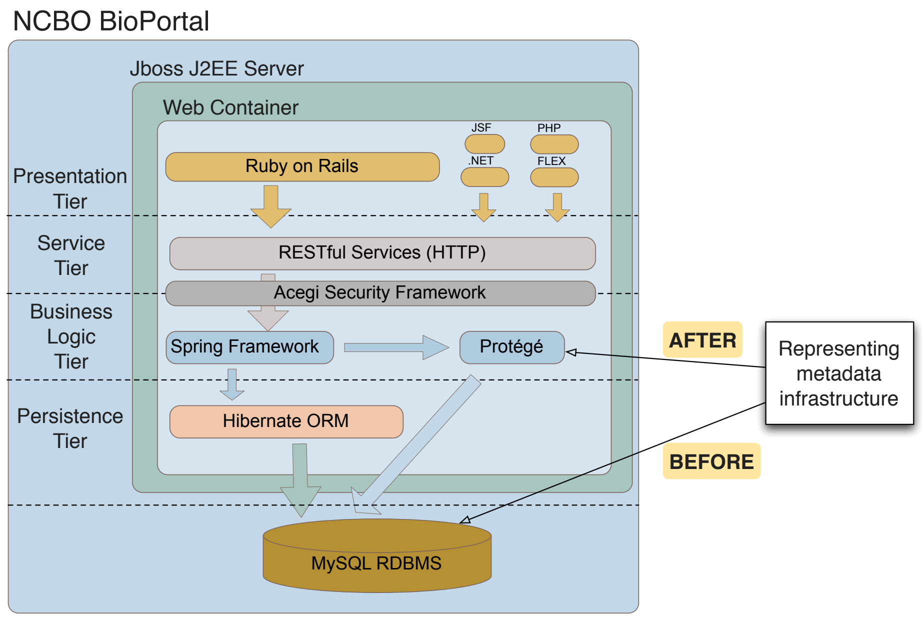


Fig. 1. Layered architecture of BioPortal: The Presentation Tier contains the user interface and external applications that call BioPortal REST services. The Business Logic Tier implements the logic of translating the data and metadata stored in the database into responses to the service calls.

5 The BioPortal Metadata Ontology

The BioPortal Metadata Ontology is an OWL ontology that imports a number of other ontologies (Figure 3) and includes classes to describe an ontology itself, its versions, information about the ontology, creators of an ontology, user-contributed content, such as notes, reviews, and mappings. It also contains the system information that is relevant for maintaining and representing the ontology in BioPortal, such as which users administer the ontology in BioPortal, where the ontology itself is located in the BioPortal system, internal ontology id and version ids, and so on. The instances of classes in this ontology represent the actual metadata for the BioPortal content. The BioPortal Metadata Ontologyis an OWL-Lite ontology, specifically, RDF Schema constructs, plus `owl:import`, thus supports in a scalable manner any reasoning that BioPortal requires (for example for transitivity in getting superclasses or subclasses of a class).

The BioPortal Metadata Ontology imports several ontologies that deal with the types of metadata that BioPortal supports:

– **The Ontology Metadata Vocabulary (OMV)** describes most of the metadata for ontologies themselves (e.g., domain, author, version, ontology language, etc.)
– **The Protégé Changes and Annotations Ontology** (CHAO) provides definitions for generic annotations and ontology components that they annotate.

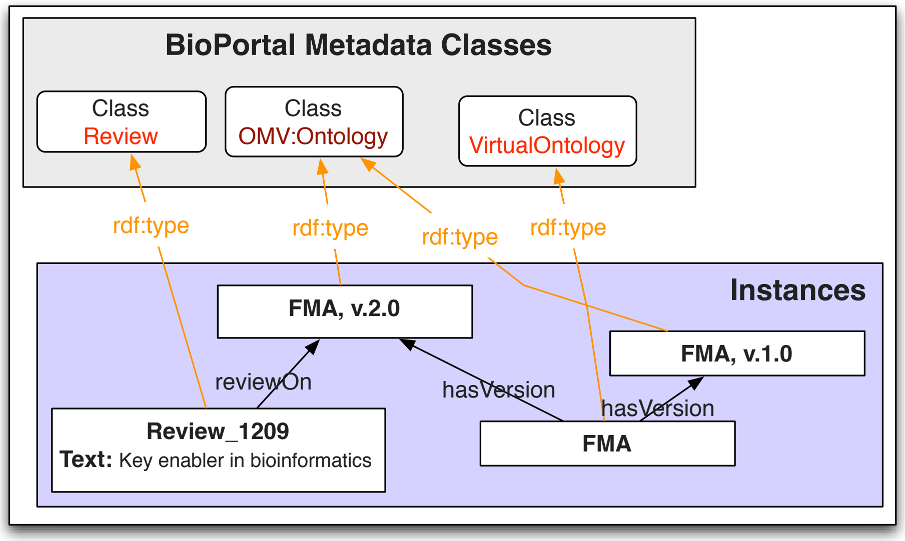


Fig. 2. Representing metadata as ontology and instances in BioPortal: The diagram shows some examples of classes and instances that represent metadata in BioPortal. There is an instance of the class `VirtualOntology` that corresponds to the Foundational Model of Anatomy (FMA) and not any specific version of it. This instance points to instances of `OMV:Ontology` describing specific versions of FMA in BioPortal. A review provided by a user (an instance of the class `Review`) points to a specific version of the FMA for which the review was created.

– **The Protégé Mapping Ontology** provides vocabulary for describing one-to-one mappings between concepts and corresponding metadata.

The OMV provides the vocabulary for describing a specific ontology version. An instance of the class `OMV:Ontology` describes a single version of an ontology. This class contains properties describing pertinent information about the ontology. The BioPortal Metadata Ontology extends this class to add properties that are specific to BioPortal as well as some missing properties that should have been in OMV.[8] These properties include system information such as the internal id, the user who submitted the ontology, the internal status (e.g., scheduled for parsing, loaded, error), and associated reviews.

We use the instances of the Protégé CHAO ontology to represent comments that BioPortal users contribute to the ontologies. Each comment is represented as an annotation attached to a specific class (in a specific ontology version) or to another annotation (if it is a response to a comment). Users can use the comments, for example, to carry out discussions about modeling decisions, make suggestions for changes, ask questions. The same mechanism exists in Collaborative Protégé, a version of the Protégé ontology editor that supports collaborative ontology editing. Because BioPortal and Protégé share the same structure for representing user comments and discussions, one can potentially

[8] We collaborate with OMV developers to include these properties in future versions of OMV

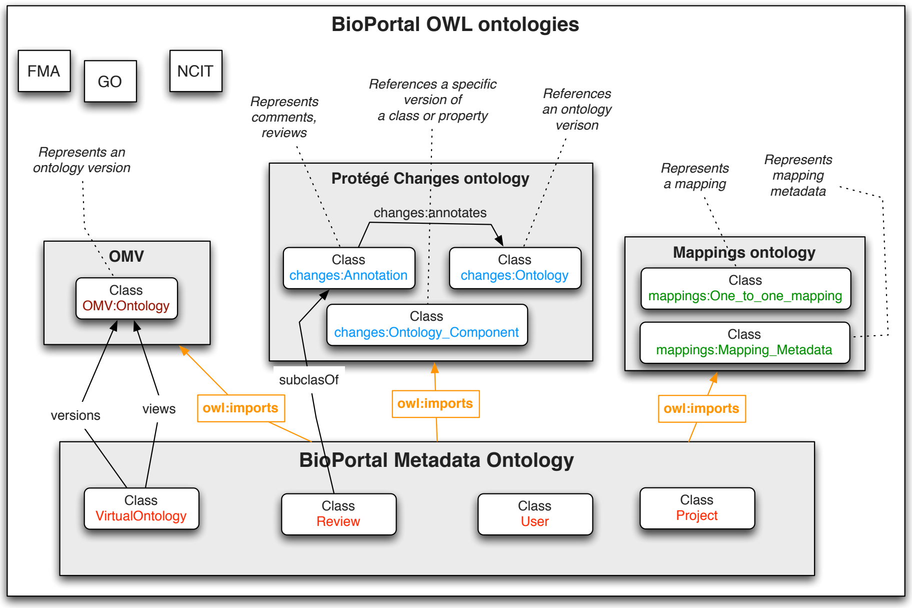
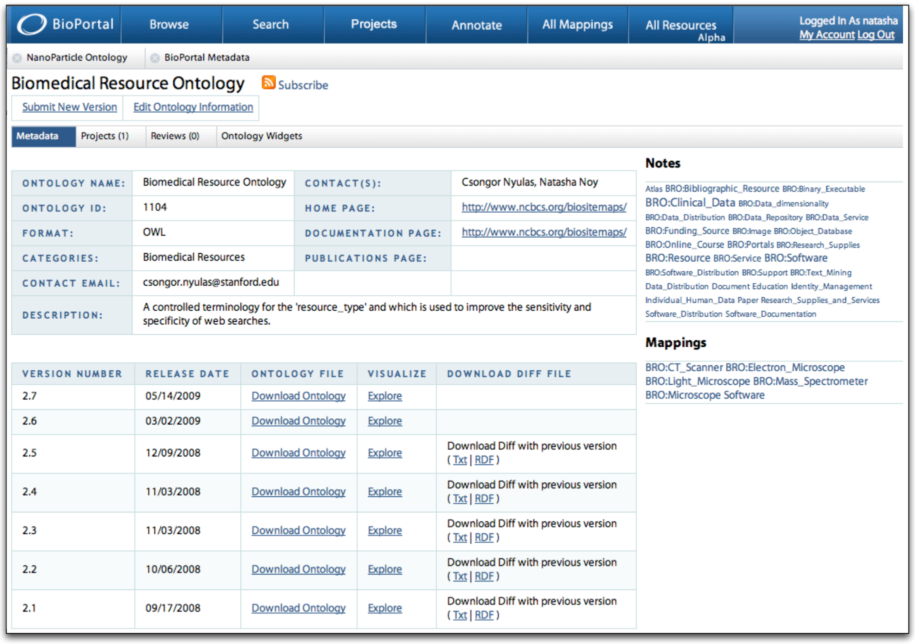

BioPortal OWL ontologies
FMA
GO
NCIT
Represents an ontology version
OMV
Class OMV:Ontology
Represents comments, reviews
References a specific version of a class or property
References an ontology verison
Protégé Changes ontology
changes:annotates
Class changes:Annotation
Class changes:Ontology
Class changes:Ontology_Component
Represents a mapping
Represents mapping metadata
Mappings ontology
Class mappings:One_to_one_mapping
Class mappings:Mapping_Metadata
versions
views
owl:imports
subclasOf
owl:imports
owl:imports
BioPortal Metadata Ontology
Class VirtualOntology
Class Review
Class User
Class Project

Fig. 3. The BioPortal Metadata Ontology: Some classes and ontologies that the BioPortal Metadata Ontology imports. The BioPortal Metadata Ontology is itself in the BioPortal repository, along with domain-specific ontologies such as the Gene Ontology (GO), the FMA, and others.

open a BioPortal ontology in Protégé and see the comments contributed by BioPortal users. We are currently working on the tighter integration of the two tools.

The BioPortal Metadata Ontology adds classes that are specific to BioPortal functionalities and that are not described in the imported ontologies. These classes represent ontology projects and reviews on ontologies and dimensions that users can use to evaluate ontologies.

The BioPortal Metadata Ontology introduces some convenience classes that abstract the information already present in other classes. This approach poses minor maintenance challenges, which we mitigate by changing instances only programmatically. However, this approach greatly facilitates access to the information. Consider the following example: In OMV, the description of an ontology version (instance of `OMV:Ontology`) points to a previous version of the same ontology. Thus, in order to present a table with the information about all versions of an ontology (as in Figure 4), we must first find the latest version of the ontology, and then traverse the instances to collect all versions. Thus, we introduce the notion of `VirtualOntology` which is an object representing the collection of all versions of the ontology. This object has the minimal information that is shared among all versions, such as the ontology name (e.g., Gene Ontology); the “virtual ontology id”—the global id that, when used to access an ontology, always resolves to the latest version; the information on who administers the ontology and the list of versions (instances of `OMV:Ontology`).



Biomedical Resource Ontology

ONTOLOGY NAME:	Biomedical Resource Ontology	CONTACT(S):	Csongor Nyulas, Natasha Noy
ONTOLOGY ID:	1104	HOME PAGE:	http://www.ncbcs.org/biositemaps/
FORMAT:	OWL	DOCUMENTATION PAGE:	http://www.ncbcs.org/biositemaps/
CATEGORIES:	Biomedical Resources	PUBLICATIONS PAGE:	
CONTACT EMAIL:	csongor.nyulas@stanford.edu		
DESCRIPTION:	A controlled terminology for the 'resource_type' and which is used to improve the sensitivity and specificity of web searches.		

VERSION NUMBER	RELEASE DATE	ONTOLOGY FILE	VISUALIZE	DOWNLOAD DIFF FILE
2.7	05/14/2009	Download Ontology	Explore	
2.6	03/02/2009	Download Ontology	Explore	
2.5	12/09/2008	Download Ontology	Explore	Download Diff with previous version (Txt | RDF)
2.4	11/03/2008	Download Ontology	Explore	Download Diff with previous version (Txt | RDF)
2.3	11/03/2008	Download Ontology	Explore	Download Diff with previous version (Txt | RDF)
2.2	10/06/2008	Download Ontology	Explore	Download Diff with previous version (Txt | RDF)
2.1	09/17/2008	Download Ontology	Explore	Download Diff with previous version (Txt | RDF)

Notes

Atlas BRO:Bibliographic_Resource BRO:Binary_Executable BRO:Clinical_Data BRO:Data_dimensionality BRO:Data_Distribution BRO:Data_Repository BRO:Data_Service BRO:Funding_Source BRO:Image BRO:Object_Database BRO:Online_Course BRO:Portals BRO:Research_Supplies BRO:Resource BRO:Service BRO:Software BRO:Software_Distribution BRO:Support BRO:Text_Mining Data_Distribution Document Education Identity_Management Individual_Human_Data Paper Research_Supplies_and_Services Software_Distribution Software_Documentation

Mappings

BRO:CT_Scanner BRO:Electron_Microscope BRO:Light_Microscope BRO:Mass_Spectrometer BRO:Microscope Software

Fig. 4. Ontology metadata in BioPortal user interface: This page presents the metadata for one of the BioPortal ontologies. It shows the provenance and other information about the ontology itself, the list of different versions of this ontology in BioPortal, and links to notes that users contributed to this ontology and mappings between concepts in this ontology and concepts in other ontologies. The notes and mappings are presented as a tag cloud: classes that appear in a larger font have more notes and mappings than others.

6 Validating Feasibility: Implementing BioPortal metadata

We have validated the feasibility of our approach by implementing it as an infrastructure for BioPortal.[9] The previous version of BioPortal used a database schema to reflect the metadata schema and posed exactly the flexibility and customizability challenges that we described in Section 3.

First, we replaced the databases in the storage layer with a Protégé ontology, implemented in its own one-table ontology-independent schema (Figure 1). Second, we replaced the metadata implementation in the business-logic layer with the appropriate Protégé API calls. Third, we transferred the metadata from the old database to instances in the metadata ontology. Our goal was to maintain the same API at the service layer so

[9] Note to reviewers: At the time of the paper submission, the main BioPortal server at `http://bioportal.bioontology.org` runs using the database-based metadata representation. We use the new infrastructure in our development server that is not yet accessible to the public. We expect to transition the new implementation to our production server before the end of the Summer 2009.

that the user interface does not need to be modified and other applications that already use our REST service API can continue to use it.

Since we access the metadata ontology through the Protégé API —which was successfully used in other projects to access in a scalable manner ontologies having more than 8000 classes and 5 million instances [10]— we predict with confidence that the new representation of the metadata about ontologies will scale well. The version of BioPortal currently running on our development server uses this infrastructure successfully, thus validating the approach. We plan to release this implementation to production at end of August 2009.

At the time of this writing, the transition of metadata to ontology-based approach is not complete. Currently, ontology details, ontology versions, and ontology views (see Section 7) are represented as ontology instances. We are in the process of transitioning the rest of the metadata.

7 Validating Extensibility: representing ontology views

As we discussed earlier, one of our main motivations to moving to the ontology-based approach was greater flexibility and adaptability of the metadata. We validated these properties by implementing support for ontology views in BioPortal, which did not previously exist. In our implementation we store only materialized views computed and submitted by the users, together with the metadata describing how the view was generated (the language, engine, etc.)

In this context, a view is any subset of an ontology that is itself an ontology. A view can be created manually or automatically by a view-generation tool. For instance, our collaborators have several views of the Foundational Model of Anatomy (FMA) [15]. One of the views represents the subset of the FMA that would be of interest to a radiologist; another view deals exclusively with Liver; yet another view focuses on the representation of neuroanatomy. The first of these views was generated manually, by starting with the FMA in Protégé and removing the unnecessary branches. The other two views are the results of queries in an extension of SPARQL that our collaborators have developed [17]. When ontology users generate and materialize the views, they often want to share them with other researchers in the field. Thus, when users view a page for FMA, they can see not only its different versions, but also the views available for it.

7.1 Requirements for Representing Views

Discussions with our collaborators led to the following set of requirements on view representation:

- Each view is itself an ontology and can have metadata, be explorable, have reviews, statistics, and so on.
- A view is defined on a specific version of an ontology.

[10] `http://protegewiki.stanford.edu/index.php/Scalability_and_Tuning`

– There is a notion of a "virtual view" (cf. "virtual ontology") such as a view of Liver-related concepts in FMA created for a particular purpose.
  - Each virtual view will have at least one version, but can have several.
  - Each version of the view also has its own metadata (inherited from ontology metadata, but with additional fields).
– We must be able to represent views on views (with the same requirements).
– We must be able to represent views that use more than one ontology.

Note that these requirements suggest a fairly complex structure for view representation, with many cross-references with different components of ontology metadata (ontologies and ontology versions).

7.2 Representing Views in BioPortal

When representing views in BioPortal, we treated them essentially in the same way as regular ontologies, thus getting the browsing, annotation, and other features "for free." We extended the BioPortal Metadata Ontology to represent features that are specific to describing views (Figure 5). A class `VirtualView` is a subclass of `VirtualOntology` and points to the list of versions of the view. Each version of a view is an instance of the class `OntologyView`, which is a subclass of `OMV:Ontology`. Thus, it inherits all the properties that describe an ontology version (e.g., description, domain, author) and adds its own. The view-specific properties include the following:

– The property `viewDefinition` is the textual representation of the view definition. This definition can be a query that was used to create the view, a set of traversal directive (as in Prompt), or any other way that specifies how the view was extracted.
– The property `viewDefinitionLanguage` defines the language that was used to define the view (e.g., the query language, such as SPARQL, for a view that was generated by a query). The range of the property is the class `ViewDefinitionLanguage`, whose instances will have all the pertinent attributes of the language (name, creator, url) as well as the specific version that was used for the view.
– The property `viewGenerationEngine` contains the engine that was used to compute the view. The range of the property is the class `ViewGenerationEngine`, whose instances will have all the pertinent attribute of the engine (name, creator, url) as well as the specific version that was used to generate the view.

7.3 Providing Support for Views in BioPortal

In this ontology-based infrastructure for representing metadata, in order to handle views to BioPortal and satisfying all the requirements that we have outlined earlier, we had to do the following. First, we extended the BioPortal Metadata Ontology as we described in Section 7.2 and Figure 5. After this step, we already had all the structure to represent the views. Second, we implemented the new view-specific REST services (e.g., returning all views for an ontology). This implementation uses the Protégé API to access the view metadata. As we expected, in this implementation we indeed needed to

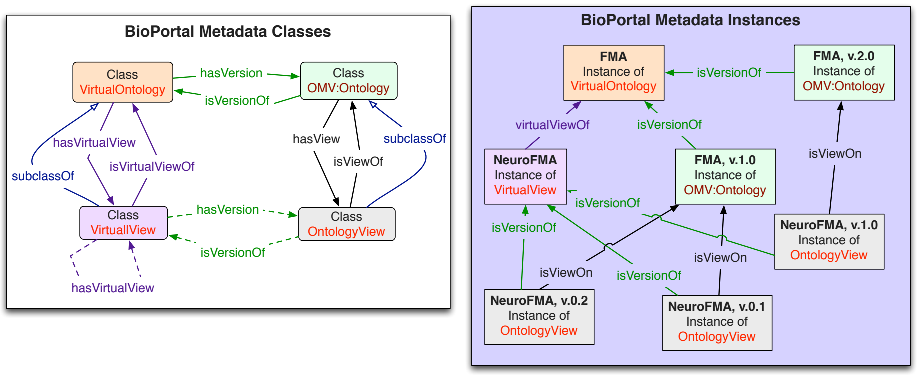

BioPortal Metadata Classes
Class VirtualOntology
hasVersion
isVersionOf
Class OMV:Ontology
hasVirtualView
subclassOf
isVirtualViewOf
hasView
isViewOf
subclassOf
Class VirtualView
hasVersion
isVersionOf
Class OntologyView
hasVirtualView
BioPortal Metadata Instances
FMA Instance of VirtualOntology
isVersionOf
FMA, v.2.0 Instance of OMV:Ontology
virtualViewOf
isVersionOf
isViewOn
NeuroFMA Instance of VirtualView
FMA, v.1.0 Instance of OMV:Ontology
isVersionOf
isVersionOf
NeuroFMA, v.1.0 Instance of OntologyView
isViewOn
isViewOn
isVersionOf
NeuroFMA, v.0.2 Instance of OntologyView
NeuroFMA, v.0.1 Instance of OntologyView

Fig. 5. Classes and instances for representing views: The class `VirtualView` corresponds to the logical description of a specific view (e.g., NeuroFMA, which is an extraction of FMA concepts relevant to Neuroimaging). It is a subclass of `VirtualOntology` because each view is an ontology itself. Similarly, the class `OntologyView` extends the class `OMV:Ontology` and describes specific versions of the view. The right-hand side of the figure shows some example instances of these classes and properties that link them.

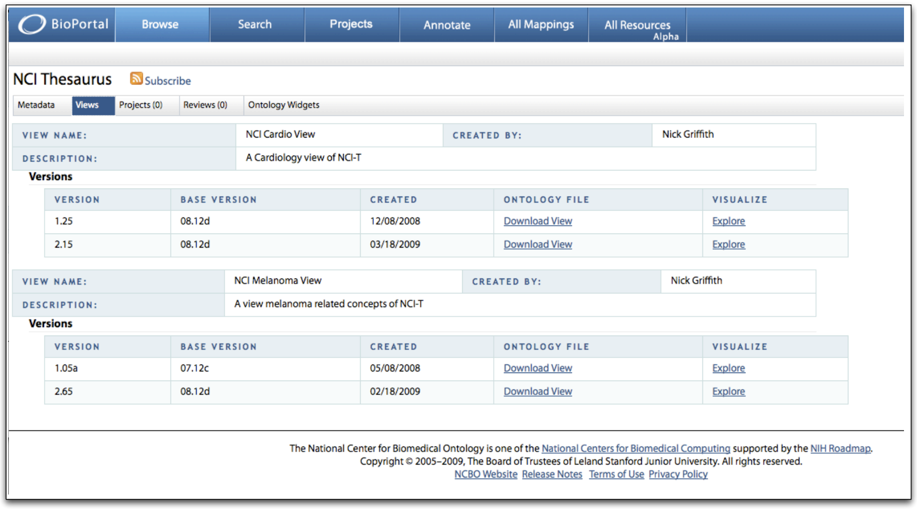
focus exclusively on the view-specific logic and structures and did not need to change anything else (e.g., the application infrastructure or the architecture). Some of the functional requirements for handling the views crystallized while we were already working on the implementation, and the only changes that were necessary were the changes to the ontology itself. We did not need to change any database schemas or to add any additional structures (e.g., Hibernate stubs). The work that was required was limited to implementing the logic for the services directly.

In comparison, adding views to the old (Hibernate-based) infrastructure, would have involved some additional changes like: (a) creating new database tables (for views and virtual views), (b) creating a number of foreign keys in order to represent superclass-subclass relationships and relationships expressed by object properties, (c) creating integrity constraints to enforce data integrity along those relations, and (d) generate Hibernate classes (so called “entity beans”) from the database tables.

Figure 6 shows the extended ontology-metadata page that includes the information about the views.

8 Discussion and Lessons Learned

The new implementation of BioPortal infrastructure validates the use of ontologies and ontology instances to represent all the metadata as well as system data for a repository of this kind. To the best of our knowledge, BioPortal is the first large-scale web-based application that uses ontologies to represent its internal data. Our experience has shown that using an ontology to describe the knowledge infrastructure of an application does indeed provide the flexibility and adaptability that our application required. Our obser-



NCI Thesaurus

VIEW NAME:	NCI Cardio View	CREATED BY:	Nick Griffith
DESCRIPTION:	A Cardiology view of NCI-T		

Versions

VERSION	BASE VERSION	CREATED	ONTOLOGY FILE	VISUALIZE
1.25	08.12d	12/08/2008	Download View	Explore
2.15	08.12d	03/18/2009	Download View	Explore

VIEW NAME:	NCI Melanoma View	CREATED BY:	Nick Griffith
DESCRIPTION:	A view melanoma related concepts of NCI-T		

Versions

VERSION	BASE VERSION	CREATED	ONTOLOGY FILE	VISUALIZE
1.05a	07.12c	05/08/2008	Download View	Explore
2.65	08.12d	02/18/2009	Download View	Explore

The National Center for Biomedical Ontology is one of the National Centers for Biomedical Computing supported by the NIH Roadmap.
Copyright © 2005–2009, The Board of Trustees of Leland Stanford Junior University. All rights reserved.
NCBO Website Release Notes Terms of Use Privacy Policy

Fig. 6. The user interface for representing view metadata in BioPortal. The screenshot shows two views of the NCI Thesaurus. Each view has two different versions.

vations and discussions with developers show that it was easier and far more efficient to implement view support that satisfied the requirements that we outlined in Section 7.1 using this approach than using a database schema to describe the views and to link them to ontologies. It also enabled us to reuse much of the infrastructure that we have already developed for ontology metadata, as we could treat views as special cases of ontologies. As our requirements for view representation continue to evolve, we know that our changes will be easy to implement as we will need only to evolve the ontology representation and the corresponding service implementations.

It is important to note that some of the design choices in the BioPortal Metadata Ontology are driven purely by application and implementation considerations. Thus, there are parts of this ontology that are specific to the context of BioPortal implementation. We also made some of the ontology-design choices not because they were "ontologically correct" in an abstract sense but because they simplified access to information (cf. `VirtualOntology` class to collect information about all versions of an ontology). Thus, we used the ontology not only as a conceptualization of our domain (metadata representation) but also to represent the physical properties and location of the data.

There are a number of systems that use ontologies to represent some of their metadata. These include Oyster [13], the alignment server [5], and Cupboard [3]. However, in these systems the items in the repository (such as ontologies or alignments) are separate from the part that represents the metadata. Furthermore, the ontology-based metadata does not expand to representing internal system information. The ontology-based representation of metadata focuses on describing intrinsic properties of ontologies and other objects that are shared across applications.

Another class of applications that uses ontologies actively are semantic desktops (e.g., [6]). However, most semantic desktop applications do not represent system data itself using the ontologies. Furthermore, semantic desktop applications focus on creating a "semantic web" of desktop resources rather than representing the internal system data itself.

In our work, we have extended other metadata ontologies. In the future, we plan to integrate other standards. Specifically, we plan to replace the Protégé Mapping ontology that we used here for expediency with an ontology that extends SKOS with mapping-specific metadata.

Our current effort leaves some questions and concerns, however. The main concern is scalability. While Protégé is quite scalable and has been used with ontologies that have hundreds of thousands of classes and instances, we do not know how it will behave with millions of instances describing metadata. As we noted in Section 6, we have not yet moved mappings to this infrastructure. At the same time, we have just uploaded one million new mappings to BioPortal. We are yet to test whether our current Protégé-based infrastructure will be sufficiently scalable for this number of mappings. If we learn that it is not, it will be the limitation of the Protégé implementation itself as modern triplestore implementations easily handle this amount of data. If scalability turns out to be an issue, we will transition to a triplestore to store the instances.

We would like to emphasize that our solution is not limited to the biomedical domain. It so happens that our repository is a repository of biomedical ontologies. However, there is nothing in the BioPortal Metadata Ontology itself or in our use of it that is specific to biomedicine (except perhaps, the list of possible ontology categories). We will install the new infrastructure in other local BioPortal installations, such as the OOR sandbox that we mentioned earlier (and that accepts ontologies in any domain).

Finally, the BioPortal software is open-source. The software is domain-independent and can be used for an ontology repository in any domain or for a domain-independent one. The BioPortal Metadata Ontology is available in BioPortal and can be accessed through the BioPortal user interface or its web services.[11] We have created a snapshot of the instances of the BioPortal Metadata Ontology for our development version of BioPortal. This ontology and instances can be accessed directly through the WebProtégé server at `http://bmir-protege-dev1.stanford.edu/webprotege/`. [12]

Acknowledgments

This work was supported by the National Center for Biomedical Ontology, under roadmap-initiative grant U54 HG004028 from the National Institutes of Health and by NIH grant HL087706. We are grateful to Jim Brinkeley, Todd Detwiller, Onard Mejino, and other members of the Structural Informatics Group at Univesity of Washington for helping us identify requirements for views in BioPortal.

[11] `http://bioportal.bioontology.org`
[12] Please use the Firefox browser to access the server. Select "BioPortal Metadata Ontology" and then go to the "Individuals" tab to view the instances.

References

1. M. V. Bossche, P. Ross, I. MacLarty, B. V. Nu?elen, and N. Pelov. Ontology driven software engineering for real life applications. In *3rd International Workshop on Semantic Web Enabled Software Engineering (SWESE 2007)*, Innsubruk, Austria, 2007.
2. M. Bräuer and H. Lochmann. An ontology for software models and its practical implications for semantic web reasoning. In *The 5th European Semantic Web Conference (ESWC 2008)*, pages 34–48, Tenerife, Spain, 2008. Springer.
3. M. dAquin and H. Lewen. Cupboard a place to expose your ontologies to applications and the community. In *6th European Semantic Web Conference (ESWC 2009)*, Heraklion, Greece, 2009.
4. J. Euzenat. An api for ontology alignment. In *Third International Semantic Web Conference (ISWC 2004)*, pages 698–712, Hiroshima, Japan, 2004. Springer.
5. J. Euzenat. Alignment infrastructure for ontology mediation and other applications. In *Workshop on Mediation in Semantic Web Services*, 2005.
6. T. Groza, S. Handschuh, K. Moeller, G. Grimnes, L. Sauermann, E. Minack, C. Mesnage, M. Jazayeri, G. Reif, and R. Gudjnsdttir. The nepomuk project - on the way to the social semantic desktop. In T. Pellegrini and S. Schaffert, editors, *Proceedings of I-Semantics' 07*, pages 201–211. JUCS, Sept. 2007.
7. L. Hart, P. Emery, R. Colomb, K. Raymond, D. Chang, Y. Ye, E. Kendall, and M. Dutra. Usage scenarios and goals for ontology definition metamodel. In *5th International Conference on Web Information Systems Engineering (WISE 2004)*, pages 596–607, Brisbane, Australia, 2004. Springer.
8. T. Lehmann. A framework for ontology based integration of structured it-systems. In *3rd International Workshop on Semantic Web Enabled Software Engineering (SWESE 2007)*, Innsubruk, Austria, 2007.
9. N. F. Noy, A. Chugh, W. Liu, and M. A. Musen. A framework for ontology evolution in collaborative environments. In *Fifth International Semantic Web Conference, ISWC*, volume LNCS 4273, Athens, GA, 2006. Springer.
10. N. F. Noy, N. Griffith, and M. A. Musen. Collecting community-based mappings in an ontology repository. In *7th International Semantic Web Conference (ISWC 2008)*, Karlsruhe, Germany, 2008.
11. N. F. Noy, N. H. Shah, P. L. Whetzel, B. Dai, M. Dorf, N. Griffith, C. Jonquet, D. L. Rubin, M.-A. Storey, C. G. Chute, and M. A. Musen. Bioportal: ontologies and integrated data resources at the click of a mouse. *Nucleic Acids Research*, 2009.
12. C. I. Nyulas, M. J. O'Connor, S. W. Tu, A. Okhmatovskaia, D. Buckeridge, and M. A. Musen. An ontology-driven framework for deploying jade agent systems. In *IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, Sydney, Australia, 2008.
13. R. Palma, P. Haase, and A. Gomez-Perez. Oyster: sharing and re-using ontologies in a peer-to-peer community. In *15th international conference on World Wide Web (WWW 2006)*, pages 1009–1010, Edinburgh, Scotland, 2006. ACM.
14. R. Palma, J. Hartmann, and P. Haase. OMV: Ontology Metadata Vocabulary for the Semantic Web. Technical report, http://ontoware.org/projects/omv/, 2008.
15. C. Rosse and J. L. V. Mejino. A reference ontology for bioinformatics: The Foundational Model of Anatomy. *Journal of Biomedical Informatics.*, 2004.
16. H. H. Shahri, J. A. Hendler, and A. A. Porter. Software configuration management using ontologies. In *3rd International Workshop on Semantic Web Enabled Software Engineering (SWESE 2007)*, Innsubruk, Austria, 2007.
17. M. Shaw, L. T. Detwiler, J. F. Brinkley, and D. Suciu. Generating application ontologies from reference ontologies. In *AMIA Annual Symposium*, Washington, DC, 2008.

Integrating Linked Data Driven Software Development Interaction into an IDE

Aftab Iqbal, Oana Ureche, and Michael Hausenblas

DERI, National University of Ireland, Galway, Ireland
{firstname.lastname}@deri.org

Abstract. With "Linked Data Driven Software Development" (LD2SD) we have introduced a light-weight, linked data-based framework that allows to integrate software artefacts, such as version control systems and issue trackers, as well as discussion forums. The so created interlinked data-space enables uniform query and browsing facilities. In this paper we elaborate on the interaction part of LD2SD, and demonstrate how the LD2SD-interaction can be integrated into an Integrated Development Environment (IDE). We have performed an end-user evaluation and report on our findings and outline future steps.

Key words: linked data, interaction, Software Engineering, IDE

1 Introduction

In the software development process, both humans and so called *software artefacts* are involved (cf. Fig. 1 from [IUHT09]). Some of the software artefacts are directly under the control of the developers, while others are shared among users and developers, such as bug tracking system, documentation, discussion forums etc.

Developers use different mediums of communication to interact with each other and to solve problems. For example, Java source code and bugs are often discussed in forums or project mailing lists. However, the interconnections among software artefacts in the various data sources are typically not explicit. Developers nowadays have to perform keyword-based searches on the Web to find examples for source code or need to manually trace discussions about a bug on a blog. The attraction of using semantic technologies in order to address this issue is based on the idea to transform software artefacts into an conceptually organised and interlinked data-space, incorporating data from different software artefacts [DB08].

With "Linked Data Driven Software Development" (LD2SD) [IUHT09] we have introduced a linked data [BHBL09] based, light-weight framework that allows to integrate software artefacts. The so created interlinked data-space enables uniform query and browsing facilities. In this paper we elaborate on the interaction [Hea08] part of LD2SD, and demonstrate how the LD2SD-interaction can be integrated into an Integrated Development Environment (IDE).

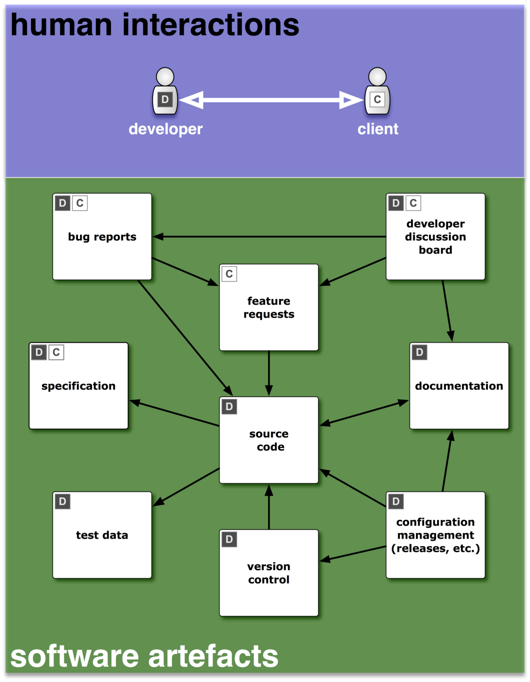


Fig. 1: The software development process, including its participants.

With the work at hand, we aim at enabling Java developers to explore related information about software artefacts. We present an interface that developers can use as a plug-in in their development environment (IDE), such as Eclipse[1] to find related information about Java source code, which might be found in a bug tracking systems, Subversion logs or in a blog discussion.

The paper is structured as follows: in Section 2, we discuss our LD2SD-based approach. We report on a concrete use case and an implementation of the LD2SD interaction in Section 3. The Section 4 presents the results from an end-user evaluation. In Section 5 we review related and existing work and eventually, in Section 6, we conclude our work and give an outlook on future work regarding LD2SD.

2 Linked Data Driven Software Development (LD2SD)

There are manifold ways to integrate different software artefacts. In order to provide a unified access to the different software artefacts, one needs to interlink and integrate these software artefacts, as we have argued in [IUHT09]. The overall concept of *Linked Data Driven Software Development* (LD2SD)—depicted in

[1] `http://www.eclipse.org/`

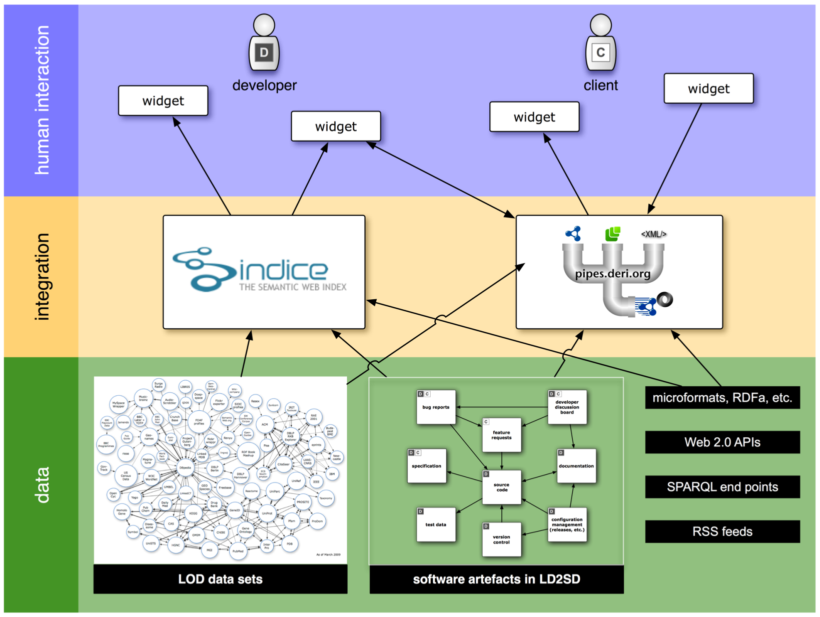


Fig. 2: The layered LD2SD approach for software artefacts integration.

Fig. 2 from [IUHT09]—is a layered, linked data based integration of software artefacts.

In this paper we elaborate on the top-most part of the LD2SD approach, the *interaction part*. We have chosen a particular setup, assuming a Java developer using Eclipse. The goal was to create an Eclipse plug-in that allows the developer to consume LD2SD data.

To enable a rapid development of our demonstrator, we decided to build upon an already available indexing service for the integration layer. The data layer, including RDFication and Interlinking was reused from [IUHT09].

In Fig. 3 our setup is described in detail:

1. RDFizing the software artefacts based on the linked data principles, yielding LD2SD datasets;
2. Using an indexing engine, *Apache Lucene/Solr*[2] to index the LD2SD datasets;
3. Develop a lookup service on top of the indexing service to enable keyword-based search over the LD2SD datasets;
4. Deliver the information to the developer via an Eclipse plug-in.

In order to achieve the linked data functionality for software artefacts, one needs to generate RDF data and interlink them. We have shown elsewhere [IUHT09] how to achieve the RDFizing and interlinking of the software artefacts.

[2] `http://lucene.apache.org/solr/`

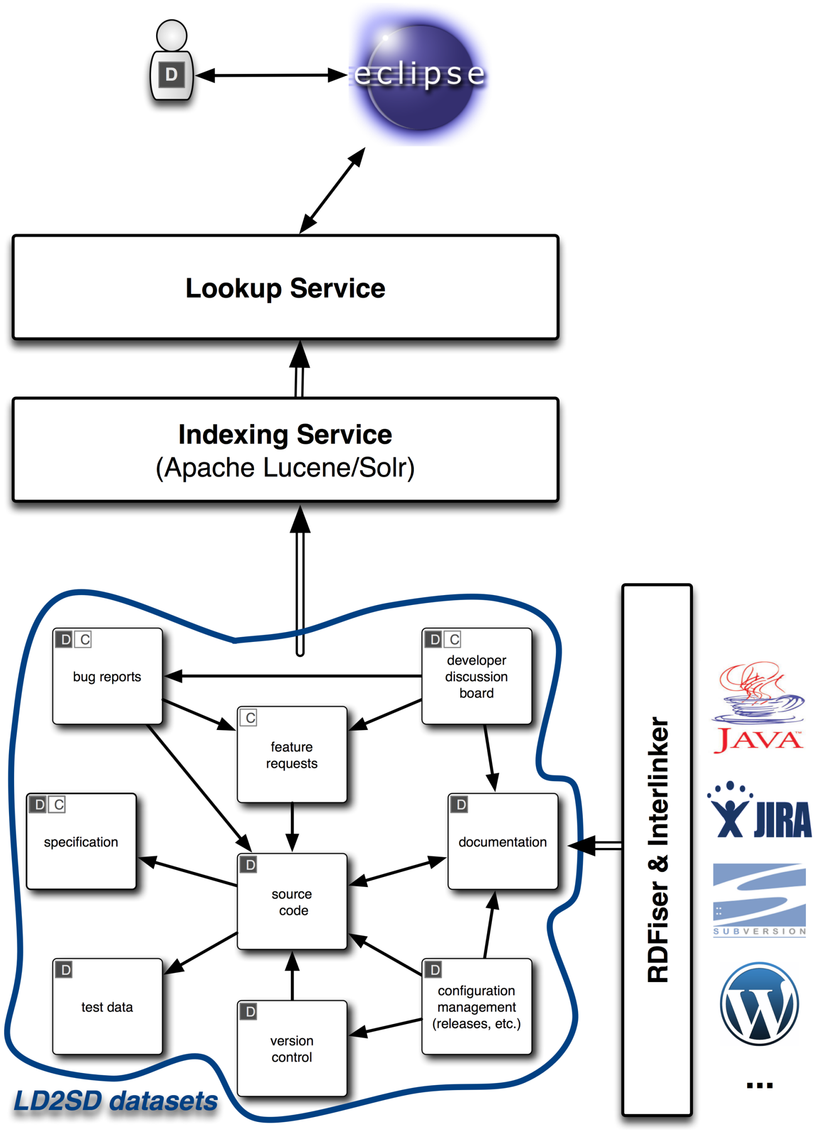


Fig. 3: LD2SD Interaction Setup.

After RDFizing and interlinking, the datasets are indexed by the *Apache Solr* indexing service. Each document is split into multiple documents based on the number of resources described in it and each resource is indexed as a separate document. The advantage of splitting an RDF document into sub-documents is that the lookup service returns the specific resource as a result rather than the whole document. For example, the Java2RDF [IUHT09] parser generates a single RDF dump of a software project, which contains description about the Java packages, the Java source in each package and the Java methods in each class along with JavaDoc.

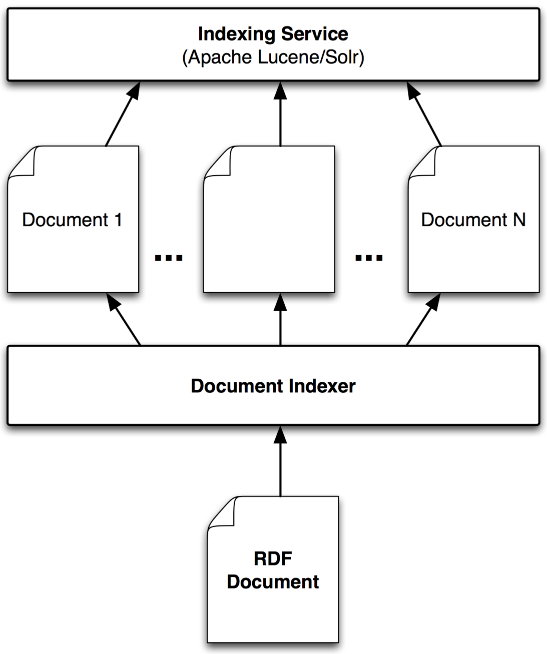


Fig. 4: Document indexing process.

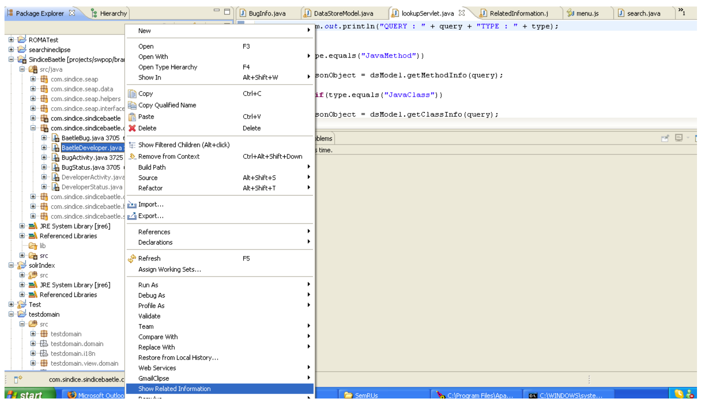
The *Document Indexer* (cf. Fig. 4) indexes each package, Java classes and Java methods as a separate document. This approach allows the user to query for a Java class or a Java method and the *Apache Solr* indexing service will return the specific document instead of the RDF document for the entire software project. In order to query for the documents stored by the indexing service, the lookup service enables keyword-based queries against the indexing service. Although, different software artefacts are indexed by the indexing service, the *Document Indexer* does not only store content, but also the URI of document and the type of document, for example `JavaClass`, `Package`, `Discussion` etc.

3 Implementation

For the concrete implementation of the LD2SD plug-in, we have chosen Eclipse, a popular Java IDE. Software developers spend most of their time in IDEs such as Eclipse, though need to access bug tracking system to log bugs or discuss development issues in blogs or mailing lists, etc. As we wanted to offer a cross-platform, extensible solution, we decided to implement the actual interface of the plug-in as a linked data application [Hau09] based on HTML and utilising the Eclipse-internal Web browser. Alternatively, the LD2SD-interaction is also possible via a standalone Web browser.

3.1 Interaction via Eclipse plug-in

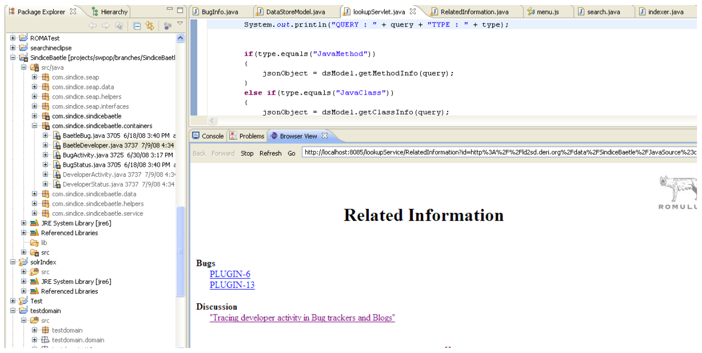
To enable developers to search for documents or to find related information about software artefacts without leaving their development environment, we have implemented an Eclipse plug-in. This enables the developer to retrieve related information about entities (such as classes, methods, etc.) in the Java source code of a software project. Say, the developer is interested in related information about a certain Java class. One way to trigger the LD2SD plug-in is to right click on the Java class and select the “Show Related Information” command (cf. bottom of Fig. 5) from the context menu.



```
m.out.println("QUERY : " + query + "TYPE : " + type);

pe.equals("JavaMethod"))

sonObject = dsModel.getMethodInfo(query);

if(type.equals("JavaClass"))

sonObject = dsModel.getClassInfo(query);
```

Fig. 5: Triggering the plug-in via the context menu.

In response, the lookup-service provides URIs as entry points and issues automatically a SPARQL query, which is executed on the entire LD2SD datasets to retrieve related information about that Java class as shown in Fig. 6.



```
System.out.println("QUERY : " + query + "TYPE : " + type);


if(type.equals("JavaMethod"))
{
    jsonObject = dsModel.getMethodInfo(query);
}
else if(type.equals("JavaClass"))
{
    jsonObject = dsModel.getClassInfo(query);
```

http://localhost:8085/lookupService/RelatedInformation?id=http%3A%2F%2Fld2sd.deri.org%2Fdata%2FSindiceBaetle%2FJavaSource%23c

Related Information

Bugs
PLUGIN-6
PLUGIN-13

Discussion
"Tracing developer activity in Bug trackers and Blogs"

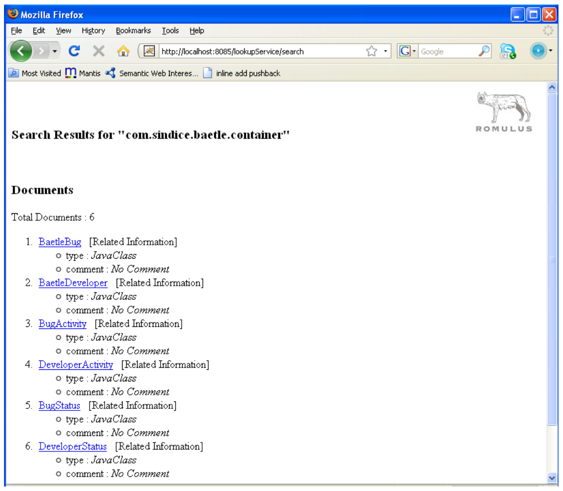
Package Explorer
Hierarchy
ROMATest
searchineclipse
SindiceBaetle [projects/swpop/branches/SindiceBaetl
src/java
com.sindice.seap
com.sindice.seap.data
com.sindice.seap.helpers
com.sindice.seap.interfaces
com.sindice.sindicebaetle
com.sindice.sindicebaetle.containers
BaetleBug.java 3705 6/18/08 3:40 PM
BaetleDeveloper.java 3737 7/9/08 4:34
BugActivity.java 3725 6/30/08 3:17 PM
BugStatus.java 3705 6/18/08 3:40 PM
DeveloperActivity.java 3737 7/9/08 4:34
DeveloperStatus.java 3737 7/9/08 4:34
com.sindice.sindicebaetle.data
com.sindice.sindicebaetle.helpers
com.sindice.sindicebaetle.service
JRE System Library [jre6]
Referenced Libraries
lib
src
solrIndex
src
JRE System Library [jre6]
Referenced Libraries
Test
testdomain
src
testdomain
testdomain.domain
BugInfo.java
DataStoreModel.java
lookupServlet.java
RelatedInformation.java
menu.js
search.java
indexer.java
Console
Problems
Browser View
Back
Forward
Stop
Refresh
Go
ROMULUS

Fig. 6: Related information about a Java class in the Eclipse IDE.

3.2 Interaction via standalone Web browser

Alternatively to the Eclipse plug-in, the interaction is possible via a standalone Web browser. Developers can issue simple keyword-based queries and can navigate the search results concerning related information for that resource.

Let us assume the developer is interested about related information concerning a certain Java package. He can query the LD2SD data by using Web browser. Typically, the developer will enter, say, a package name and the lookup service returns relevant information about all Java classes belonging to that package (Fig. 7). Additionally, the developer can browser for related information about a Java class by clicking the "Related Information" link displayed next to it.



http://localhost:8085/lookupService/search

Search Results for "com.sindice.baetle.container"

Documents

Total Documents : 6

1. BaetleBug [Related Information]
   - type : *JavaClass*
   - comment : *No Comment*
2. BaetleDeveloper [Related Information]
   - type : *JavaClass*
   - comment : *No Comment*
3. BugActivity [Related Information]
   - type : *JavaClass*
   - comment : *No Comment*
4. DeveloperActivity [Related Information]
   - type : *JavaClass*
   - comment : *No Comment*
5. BugStatus [Related Information]
   - type : *JavaClass*
   - comment : *No Comment*
6. DeveloperStatus [Related Information]
   - type : *JavaClass*
   - comment : *No Comment*

Fig. 7: Search results for a Java package in the Web interface.

4 Evaluation

To evaluate our approach, we have prepared data from a software project (bug tracking data, subversion logs, blogs and Java source code). We assessed the usability of our Eclipse plug-in approach via end-user evaluations requiring participants to perform a set of tasks. We measured the time it took them to complete the tasks and asked questions around the usability.

No.	Software Artefact	Familiarity (%)
1	Eclipse IDE	80
2	Bug tracking systems	60
3	Discussion blogs	100
4	Subversion plug-in for bug tracking system	42
5	Mylyn plug-in for Eclipse IDE	45

Table 1: Familiarity of the participants with the tools.

Twelve participants took part in the evaluation. The participants have development experience ranging from one to five years with different backgrounds. Table. 1 shows the familiarity of the participants with the different software artefacts.

The participants were asked to carry out a set of tasks on the test data:

Task 1 Identify all blog posts that mention a specific Java class.
Task 2 Identify all bugs that have been fixed by modifying a specific Java class.
Task 3 Identify all developers that are working on a Java package.
Task 4 Identify all blog posts that mention a specific Java package.
Task 5 Identify all bugs that belong to a specific Java package.

For the *Task 2* and *Task 5* we have installed a Subversion plug-in for, which shows the logs for bugs, if any. We conducted our evaluation in two phases:

Manual Approach In the first phase, we gave participants access to the software project, the bug tracking system, and the blog. The participants searched through blog posts, traversed bug reports and searched source files for authors to carry out each task.

Plug-in Approach In the second phase, we asked participants to carry out the tasks by using our LD2SD plug-in for the Eclipse IDE.

In the first phase, we found that participants used different heuristics to list the results of each task. After the second phase, we asked the participants to compare the results of both approaches. We found that some participants missed certain results using the first approach while carrying out the tasks. Further, participants apparently had difficulties going through each bug report to identify corresponding bug entries in Subversion. During the evaluation, we asked the participants to answer a set of *yes/no* questions as shown in Table. 2.

Question	Yes	No
Is the tool useful to discover related information?	12	0
Does our approach added value compared to the usual exploration of related information?	12	0
Is the design and layout of the tool suited enough for usage?	9	3
Does the integration of software artefacts as an Eclipse plug-in offer an advantage?	10	2

Table 2: User-study regarding tool.

The time for each task has been measured in both phases of the evaluation; the resulting graph (Fig. 8) is plotted based on the average time each participant spent in carrying out the task.

A big majority of the participants found our approach of extracting related information and presenting it in an integrated manner inside Eclipse interesting and useful. For example: *"... single point access within the Eclipse IDE seems a natural tool to use. It provides information much faster than accessing individual sources. The integrated view is very convenient"*.

Participants liked the LD2SD approach to interlink the Java sources with subversion logs and bugs using linked data principles. They were able to answer questions such as: (1) who has fix the certain bug and which source files he has modified in fixing the bug, (2) who should i talk to, and (3) which blog posts are talking about that certain bug; an exemplary comment highlights this: *"... it saves time for a developer and provides a unique interface to have a look at all relevant information in a single view ... all relevant information is available on a single click ... the idea looks promising, the tool would be more useful as it evolves and add features"*.

The evaluation helped us identifying the limitations of the tool as well. Some participants commented on the interface: *"... interface is relatively small, might be an usability issue for large amount of data ... would be interesting to see how*

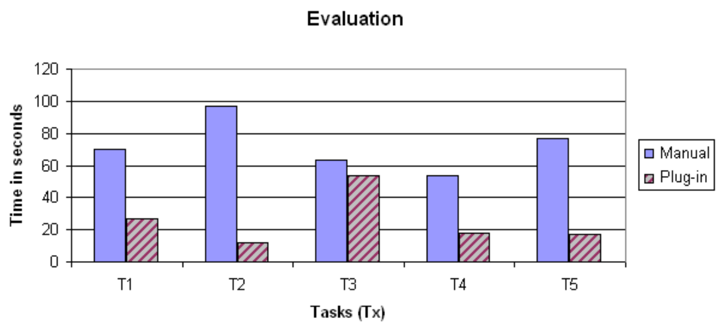


Fig. 8: Time required to perform tasks (manual vs. plug-in approach).

well it works with large amount of data–may be difficulties with presentation". Participants showed considerable interest in using our tool, especially in the case when they had to search information within hundreds of blog posts and bugs for a software project. (e.g., *"... I would definitely use the tool once it is available"*).

5 Related Work and Discussion

There are technologies available in the open source community that allow the integration of software artefacts. An interesting and closely related approach is Tesseract [SMWH09], a socio-technical dependency analyzer to enable exploration of cross-linked relationships between artefacts, developers, bugs and communication.

A related work concerning the combination of software artefacts has been described by Damljanovic et. al. [DB08]. The annotations are based on Key Concept Identification Tool(KCIT) which is capable of producing ontology-aware annotations. To interlink documents based on mentions of key concepts, the authors have used the PROTON KM ontology[3]. To enable semantic-based access through text-based queries, they used QuestIO (Question-based Interface to Ontologies), allowing to translate text-based queries into the relevant SeRQL queries, execute them and presents the results to the user. Their approach differs from our approach in that we have used *Apache Solr*, which provides high-performance engine for full-featured text search.

Another related approach has been described by Ankolekar et. al. [AS$^+$06]. Dhruv is a Semantic Web enabled prototype to support problem-solving processes in web communities. The main focus of their research is how open source communities deals with bugs in their software under development. Their approach helps to connect the communication of developers (via forums and mailing lists) with bug reports and source code of the software. They have provided an ontology to model software, developers and bugs. The ontology is semiautomatically populated with data from different information sources to support bug resolution. Their approach assists the communication between developers for bug resolution. In contrast to their approach, we have provided an Eclipse plug-in which allows software developer or project manager to search for related information about a certain Java class or Java package on a single click without leaving their IDE.

In [AGS07], Antunes et. al. have presented SRS, a Semantic Reuse System designed to store and reuse knowledge for software development. The knowledge about software artefacts is represented using *Representation Ontology*, mapped with the concepts of *Domain Ontology* and stored in the *SDKE* (Software Development Knowledge Element) repository, which is managed using *Apache Lucene*[4]. Concepts are extracted from software artefacts using linguistics tools from *Natural Language Processing* (NLP) [JM02] prior to indexed by the *Apache Lucene*.

[3] `http://proton.semanticweb.org/2005/04/protonkm`
[4] `http://lucene.apache.org`

In [KBT07], Kiefer et. al. have presented EvoOnt[5], a software repository data exchange format based on OWL. EvoOnt includes software code, code repository and bug information. The authors have used the iSPARQL[6] engine which is an extension of SPARQL, to query for similar software entities. iSPARQL is based on *virtual triples* which are used to configure *similarity joins* [Coh00]. Their approach differs from our approach in that we have provided a methodology [IUHT09] to integrate the software artefacts by RDFizing and interlinking them.

Mylyn[7] is a sub-system for the Eclipse IDE allowing multitasking for developer and task management. It provides the means to render bug-related data in the Eclipse IDE for developers to work efficiently in their development environment without having to log in to the Web based application to update or create new bugs. Mylyn is limited to issue trackers such as JIRA[8] or Bugzilla[9], and hence not able to cope with the variety of software artefacts as we desire it.

Further, there are plug-ins[10] which integrate Subversion with bug trackers. The plug-in display all Subversion commit messages related to a specific issue.

To the best of our knowledge there is no tool available that is able to deal with software artefacts in a flexible and open way as we have provided it based on the LD2SD approach.

In [IUHT09] we have already demonstrated the methodology of RDFizing and interlinking heterogeneous software artefacts. In LD2SD we address the issue of heterogeneous software artefacts by using a common data model (RDF) along with global unique identifiers (URIs). The current version of LD2SD relies on the open access to the different data sources to extract metadata, interlink and query them. As pointed out in [IUHT09], a number of ontologies (BAETLE, FOAF, SIOC, iCalendar) have been used to deal with the domain semantics.

6 Conclusion and Future Work

We have motivated and described LD2SD, a light-weight, linked data-based framework allowing to integrate software artefacts, such as version control systems and issue trackers, as well as discussion forums. In this paper we have focused on the interaction layer, that is, providing means for developers to consume LD2SD-based data from existing software artefacts.

Based on a concrete use case and implementation of the LD2SD interaction part, an Eclipse plug-in, we have conducted an end-user evaluation. Our evaluation shows that the plug-in is relatively easy to use and valuable for developers.

Our future work will focus on the improvement of the current Eclipse plug-in. Currently, our keyword-based lookup service returns a list of all relevant

[5] `http://www.ifi.uzh.ch/ddis/evo/`
[6] `http://www.ifi.uzh.ch/ddis/isparql.html`
[7] `http://www.eclipse.org/mylyn/`
[8] `http://www.atlassian.com/software/jira/`
[9] `http://www.bugzilla.org/`
[10] `http://subversion.tigris.org/links.html#misc-utils`

documents, without ranking. Based on [GGM97] we will focus on the ranking of the results, taking into account the context of the developer. Motivated by the outcome of the evaluation, we want to provide plug-ins for other IDEs, for example for NetBeans.

We want to integrate even more software artefacts (such as mailing lists, Java test cases, developers profiles, etc.) and aim at increasing the interlinking quality and quantity.

Eventually, we plan to perform follow-up evaluations on a real world open source project with a broader target audience (real world developers, etc.).

Acknowledgements

Our work has partly been supported by the European Commission under Grant No. 217031, FP7/ICT-2007.1.2—"Domain Driven Design and Mashup Oriented Development based on Open Source Java Metaframework for Pragmatic, Reliable and Secure Web Development" (Romulus)[11].

References

[AGS07] B. Antunes, P. Gomes, and N. Seco. SRS: A Software Reuse System based on the Semantic Web. In *3rd International Workshop on Semantic Web Enabled Software Engineering (SWESE) of the 4th European Semantic Web Conference (ESWC 2007)*, Innsbruck, Austria, 2007.

[AS$^+$06] A. Ankolekar, K. Sycara, , J. Herbsleb, R. Kraut, and C. Welty. Supporting online problem-solving communities with the Semantic Web. In *Proceedings of the 15th International Conference on World Wide Web*, Edinburgh, Scotland, 2006.

[BHBL09] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data—The Story So Far. *Special Issue on Linked Data, International Journal on Semantic Web and Information Systems (IJSWIS)*, page to appear, 2009.

[Coh00] W. W. Cohen. Data Integration Using Similarity Joins and a Word-Based Information Representation Language. In *ACM TOIS*, pages 288–321, 2000.

[DB08] D. Damljanovic and K. Bontcheva. Enhanced Semantic Access to Software Artefacts. In *5th International Workshop on Semantic Web Enabled Software Enginering (SWESE 2008)*, Karlsruhe, Germany, 2008.

[GGM97] L. Gravano and H. Garcia-Molina. Merging Ranks from Heterogeneous Internet Sources. In *VLDB97, Proceedings of 23rd International Conference on Very Large Data Bases*, pages 196–205, Athens, Greece, 1997.

[Hau09] M. Hausenblas. Linked Data Applications. First Community Draft, Linked Data Research Centre, 2009. `http://linkeddata.deri.ie/tr/2009-ld2webapp`.

[Hea08] T. Heath. How Will We Interact with the Web of Data? *IEEE Internet Computing*, 12(5):88–91, 2008.

[11] `http://www.ict-romulus.eu/`

[IUHT09] A. Iqbal, O. Ureche, M. Hausenblas, and G. Tummarello. LD2SD: Linked Data Driven Software Development. In *21st International Conference on Software Engineering and Knowledge Engineering (SEKE 09)*, Boston, USA, 2009.

[JM02] P. Jackson and I. Moulinier. *Natural Language Processing for Online Applications: Text Retrieval, Extraction and Categorisation.* John Benjamins Publishing Company, Amsterdam, Netherlands, Wolverhampton, United Kingdom, 2002.

[KBT07] C. Kiefer, A. Bernstein, and J. Tappolet. Mining Software Repositories with iSPARQL and a Software Evolution Ontology. In *Proceedings of the ICSE International Workshop on Mining Software Repositories (MSR)*, Minneapolis, MA, 2007.

[SMWH09] A. Sarma, L. Maccherone, P. Wagstrom, and J. Herbsleb. Tesseract: Interactive Visual Exploration of Socio-Technical Relationships in Software Development. In *International Conference on Software Engineering (ICSE 2009)*, Vancouver, Canada, 2009.

Data Modeling and Harmonization with OWL: Opportunities and Lessons Learned

John L. McCarthy[1], Denise Warzel[2], Elisa Kendall[3], Bruce Bargmeyer[1], Harold Solbrig[4], Kevin Keck[1] and Fred Gey[5]
[1]Lawrence Berkeley National Laboratory, 1 Cyclotron Rd, Berkeley, CA
[2]National Cancer Institute, National Institutes of Health, Bethesda, MD
[3]Sandpiper Software, Los Altos, CA
[1]Mayo Clinic, Rochester, MN
[5]School of Information Management, University of California, Berkeley

Abstract. Experience from recent projects helps illuminate the promises and limitations of OWL to specify, review, refine, harmonize and integrate diverse data and concept models. One of the attractive features of OWL is that it can be used by inference engines to help augment queries through inferred semantic relationships. But OWL, like SQL, is only a computer programming language. Using it to review and refine representations of data, metadata, and concept systems, including terminologies, thesauri, and ontologies, requires a well-defined abstraction layer – which itself can be specified in terms of OWL. In order to optimize, harmonize and integrate such information effectively for large scale projects, OWL definitions and relationships should be specified in terms of a standard metamodel, such as ISO/IEC 11179-3, Edition 3.

Keywords: OWL, metadata, UML, ISO 11179, metadata registration, data modeling

1 Introduction

Information technology experts are beginning to recognize the need to combine multi-disciplinary data, metadata, and concepts from a variety of related fields to address complex and/or large scale problems. Doing so requires integrating content from diverse communities with long established but different terminologies, concepts, and ways of naming and organizing their data. This paper explores how OWL (Web Ontology Language [1]) can be used to help advance decades-long efforts to represent, manage, harmonize, and integrate metadata and semantics for concept systems (including taxonomies, thesauri, and ontologies), databases, data elements, and value domains (*i.e.*, data types, and sets of valid values). For the purpose of this paper, the word “ontology” refers to a domain specific conceptualization, for a specific purpose [2].

As elaborated further below, OWL, like other computer languages, is intended for a specific purpose, it is a declarative representation language for representing knowledge and used for authoring ontologies. OWL does not provide built-in, standard modeling constructs to harmonize between ontologies, let alone describe their interrelationships or external relations to data, metadata about databases or application systems, nor to manage the evolution of such relationships over time. Several recent projects illustrate how OWL provides some very useful constructs that complement capabilities of other software engineering technologies and paradigms and can be used in conjunction with them to support new ways to model concept and data semantics. One such paradigm for concept and data management is the metadata registry, (MDR) particularly those based on the ISO/IEC 11179 Metadata registries (MDR) – Part 3: Registry metamodel and basic attributes Edition 3 (E3), [3] and related ISO/IEC 19763 Metamodel Framework for Interoperability (MFI) [4] specifications which are being extended to represent relationships between and across ontologies, as well as relationships between ontologies, terminologies, data models and web services that implement or reuse them.

Our discussion is based in part on results from three recent data and semantic modeling projects that all employed OWL, each for different purposes. The first of these projects used the Ontology Definition Metamodel (ODM)[5] to represent the BRIDG[1] model [6, 7], and transform it to OWL to help analyze and identify potential shortcomings in BRIDG [8]. The second project attempted to use an automatic ODM-based conversion tool to transform the LexGrid [9] terminology model from XML Schema into OWL. The eXtended Metadata Registry (XMDR) project [10] has used OWL in conjunction with other tools to develop a prototype system that implements extensions and enhancements included in the current CD of ISO/IEC 11179 E3, Standard. Lessons learned from these efforts will be highlighted in the sections that follow.

2 Background and Motivation

Descriptions of data, how it was collected, and what it means are an essential component of modern information systems. These descriptions, called metadata (*i.e.*, data about data), help ensure that data is interpretable by both humans and computers over time. Large organizations like the U.S. Environmental Protection Agency (EPA), National Cancer Institute (NCI), and Department of Defense (DOD) perform research that utilizes large amounts of data drawn from a variety of disparate systems. They have long recognized the need for *standardized metadata registry systems* to help manage and harmonize data elements from different databases and application systems [10]. At the same time, such organizations, along with others in Europe and Asia, were among the first proponents of national and international standards for terminologies, thesauri, concept systems and ontologies[2].

In parallel, NCI and small communities of data modelers and software engineers have been using ontologies to extend the capabilities of their metadata registry and software systems to enhance the semantics, identify potentially duplicate metadata, and increase the potential for reuse [11, 12]. Ontologies and ontology tools can facilitate automation supporting categorization and reasoning about increasingly massive amounts of data and metadata that many large organizations have to cope with. Despite apparent potential benefits from cross pollination between data management, data governance, and related disciplines that use semantic technologies, little progress has been made in marrying the two communities outside of a few isolated United States and European government agency activities. The overlap between the enterprise data and semantic web communities is very small at present, as evidenced by discussions at recent workshops at the En-

[1] The Biomedical Research Integrated Domain Group (BRIDG) Model is a domain analysis model that describes biomedical/clinical research data.

terprise Data World[3] and Semantic Technology Conferences[4] earlier this year.

The National Cancer Institute (NCI)'s data management infrastructure, comprised of the Cancer Data Standards Registry (caDSR) and Enterprise Vocabulary System (EVS), is a notable exception to the low level of synergy between data management and ontology communities. caDSR and EVS have enabled NCI to collect, harmonize and integrate detailed metadata and concepts describing some 5,500 data elements and case report forms in hundreds of clinical studies from over 90 different projects. The semantics of these elements are tied directly to over 10,000 concepts drawn from the NCI's Enterprise Vocabulary System (EVS) [13]. The U.S. Environmental Protection Agency's Environmental Data Registry (EDR), which has information covering water, air, and soil in databases and application systems, with many data elements, value domains, and terms from different terminology systems (though not directly linked) also illustrates the synergy between utilization of terminology systems and data management (*e.g.*, GEMET, Chemical Substances Taxonomy, etc.) [14].

Use of formal languages such as OWL to represent data semantics linked to terminologies can provide a tremendous opportunity for novel research and discovery, in particular if the expression of the data semantics is based on a well-defined, shared metadata model and the terminologies are well formed. World Wide Web Consortium founder Tim Berners-Lee, speaking about the "semantic web," stated that "The concept of machine-understandable documents… indicates a machine's ability to solve a *well-defined* problem by performing *well-defined* operations on existing *well-defined* data. Instead of asking machines to understand natural language, it involves asking people to make the extra effort" [15]. (Italics are ours). The implication is that people will have to take additional steps to create machine-understandable documents. Utilizing ISO/IEC11179-3 Ed 3 integrated metamodel for data and concept systems provides well-defined data descriptions for use with the OWL representation , making data comparable within and across communities because the common structure of the metadata allows programmers to develop well-defined operations for machine interpretation. Recent projects have demonstrated the power of such integrated infrastructures, but achievement of this level of integration does not come automatically simply by adopting a new language such as OWL [16].

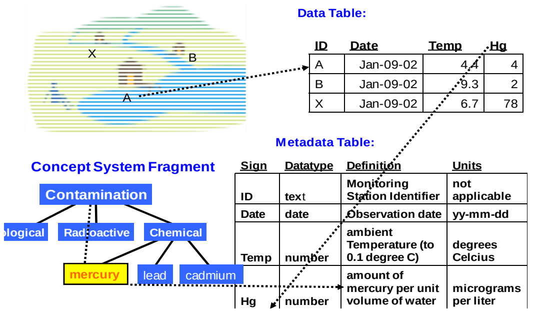
3 Levels of Abstraction for Data, Metadata and Ontologies

To elucidate these ideas more concretely, consider a hypothetical example of something in the real world about which we want to capture data, representation of the data in a database, information about this data (metadata), and a concept system fragment (which could be part of an ontology). Figure 1 shows this example with three "levels" of information – concepts, data and metadata, and some of the relationships between them. The picture at the top left represents two streams and a lake, each with its own monitoring station (A, B, and X). A formalized concept system could describe the relationships between these bodies of water and monitoring stations. The table at the top right contains three rows of data, one collected from each of

[3] See http://edw2009.wilshireconferences.com/.
[4] See http://www.semtech2009.com/.

those stations on a particular date. Each column in the table contains data for a particular observation of the variables (ID, Date, Temperature, and Mercury contamination), while each row represents values observed at a particular monitoring station location (A represents the row of values for the Lake monitoring station, B the values for the second monitoring station, and X the values for the monitoring station further downstream

Figure 1: Data, Metadata, and Concept Systems



ID	Date	Temp	Hg
A	Jan-09-02	4.4	4
B	Jan-09-02	9.3	2
X	Jan-09-02	6.7	78

Sign	Datatype	Definition	Units
ID	text	Monitoring Station Identifier	not applicable
Date	date	Observation date	yy-mm-dd
Temp	number	ambient Temperature (to 0.1 degree C)	degrees Celcius
Hg	number	amount of mercury per unit volume of water	micrograms per liter

The table on the lower right contains "metadata" – a description of the meaning and purpose of each column in the Data table -- (ID, Date, Temp, and Hg). Each column in the metadata table (*i.e*. Sign, Datatype, Description, Units) contains a piece of information about a column of the Data table, while each row represents a particular column in the Data table. Depicted at the lower left is an excerpt from the General European Multilingual Environmental Thesaurus (GEMET) concept system that shows Contamination of a body of water in three forms (Biological, Radioactive, and Chemical), along with three kinds of Chemical Contaminants, namely mercury, lead, and cadmium. Dotted lines in Figure 1 indicate some of the important relationships between different components of the three types of information. For example, the ID with the value "A" in the first cell of the first row of the Data table is the identifier for the Lake monitoring station in the picture. All the values in that row refer to the monitoring station with the ID "A". Likewise the first cell in the bottom row of the Metadata table refers to the label of the fourth column in the Data table and all the values in that metadata row pertain to the values in that column e.g. the cell of the second column in the bottom row refers to the Datatype of the fourth column in the Data table. Another dotted line shows a relationship between the Definition cell for the bottom row of the Metadata table (which relates to the 4th column of the Data table) to a particular item in the hierarchical diagram of terms from GEMET.

Ideally, we would like to be able to answer queries that span all three levels of information, Data, Metadata and Concept Systems, such as *find water bodies downstream from Fletcher Lake where the level of chemical contamination for any of a specified set of substances was greater than the allowable tolerance between December 2001 and March 2003.* Constructing and answering these kinds of queries, which was difficult at best using traditional database technology, is now possible through the open world reasoning facilities supported by OWL in conjunction with the metadata registry capabilities specified in ISO/IEC 11179-3 Ed 3.

3.1 Information Models, Concept Systems, and Ontologies

From a data engineering perspective, a conceptual or information model typically defines a set of properties and relationships that describe real world entities. Frequently, in order to define information models that will ultimately result in business applications or services, multiple information models are needed, each of which may define various aspects of the same set of entities focusing on different perspectives, context and/or processes. Each such *conceptual* information model may, in turn, correspond to one or more *logical* data models that refine various aspects of the conceptual model, which may then be realized in a number of *physical* models, or schemas that correspond to platform-specific implementations (*e.g.*, XML schema, relational databases, etc.). Depending on the level of formality imposed by the organization responsible for designing the business services, the conceptual modeling part of this process may be short circuited, or even skipped. In some cases where conceptual models *are* developed, they may not be well documented, especially if the models are only shared among a small group of developers where assumptions are implicitly understood. Consider the data values in Figure 1. if the organization had only the headings for the data table, but no other metadata, concept systems, or asserted relationships between them and the data to aid with human or machine interpretation.

From the Semantic Web perspective, an ontology provides the semantic grounding for an information model. They can be one and the same; or additional vocabularies or ontologies can be used to provide terminological support for the core business information model. Consider the often-used example of students, classes and teachers that occurs in literally hundreds of Database textbooks. These examples work because there is a relatively consistent and shared understanding of schools, teachers, students, classes, subjects, etc. within 20th century western school systems. Schools hire teachers, teachers teach classes, students attend classes, classes have schedules, etc. There are literally thousands of different information models scattered throughout these textbooks that reference this set of topics. While there is a component of each of these specific conceptualizations of these topics that is invariant, it is highly unlikely that any two of the independently developed information models or ontologies, are identical.

Typically, the purpose of developing an ontology is to define a particular conceptualization for use in a particular application or context (*e.g.*, if we are describing teachers and students, the fact that students are composed of cells, require a certain amount of nutrition each day to survive, may vote in elections and may have a preferred medical doctor, etc. are probably not relevant and are probably not included in a description of an ontology focused on schools.). Information models, particularly at the logical or physical level, may add features that are not required at the conceptual level, for example, details regarding primary or foreign keys, unique identifiers that are application specific GUIDs, and so forth. Obviously an information model attempts to maintain some sort of correlation between these identifiers and the things being identified, but these are still artifacts of the information model, not of the conceptual level.

3.2 Harmonizing Data across Multiple Systems and Ontologies

Metadata registries provide an abstraction "layer" to systematically describe, manage, and query metadata for databases, applications, and concept systems, and are particularly useful for large-scale, distributed environments. The ISO/IEC 11179 Metadata Registry family of standards also provides guidance for managing the evolution of such information over time. Figure 2 shows conceptually NCI's caDSR metadata model and Enterprise Vocabulary System's supported mappings – using concept systems to defining semantics from high-level, conceptual definitions of ISO/IEC 11179 Object Class and Property, to increasing refinement of meaning at the value set level.

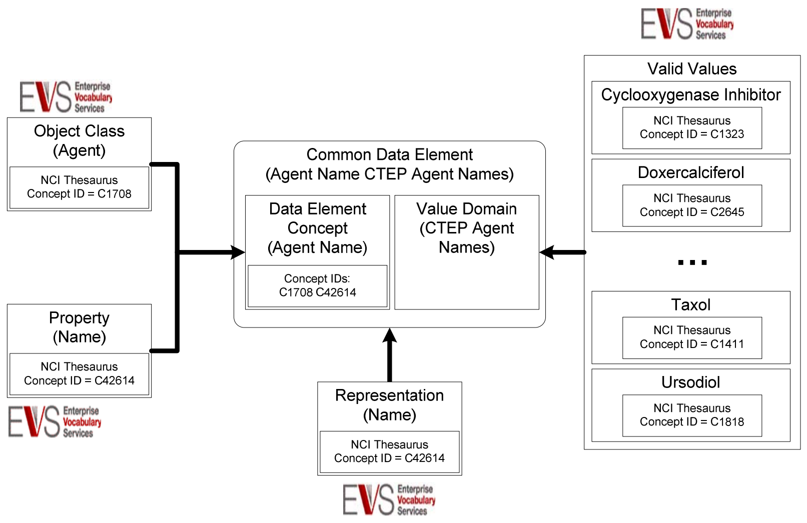

Object Class (Agent)
NCI Thesaurus Concept ID = C1708
Property (Name)
NCI Thesaurus Concept ID = C42614
Common Data Element (Agent Name CTEP Agent Names)
Data Element Concept (Agent Name)
Concept IDs: C1708 C42614
Value Domain (CTEP Agent Names)
Representation (Name)
NCI Thesaurus Concept ID = C42614
Valid Values
Cyclooxygenase Inhibitor
NCI Thesaurus Concept ID = C1323
Doxercalciferol
NCI Thesaurus Concept ID = C2645
...
Taxol
NCI Thesaurus Concept ID = C1411
Ursodiol
NCI Thesaurus Concept ID = C1818
EVS Enterprise Vocabulary Services

Figure 2. Refinement from Conceptual to Logical Definitions Using ISO/IEC 11179 MDR (left to right)

Just as SQL can be used to describe tables, columns, and relationships between columns (such as foreign keys), OWL is well suited for describing terminologies, concept systems or ontologies and various different types of relationships between their components (*e.g.*, subsets, inverses, aggregations, etc.). SQL in and of itself does not provide built-in constructs to describe, manage and query metadata registries. It can be used to construct standard tables and columns containing metadata registry information so it can be stored, managed and queried using SQL, however. As noted earlier, OWL does not provide built-in constructs to create or harmonize ontologies. Absent standard constructs or techniques for anchoring the semantics of the elements represented in OWL to an external reference that could render them comparable across models, even if those elements are grounded in the same higher-level ontologies, reasoning over multiple OWL representations may be limited to evaluation of potentially related content solely on the basis of text labels for the elements and relations. But like SQL, OWL is a powerful language that can be used to create standard constructs for registries of multiple ontologies, metadata, and their inter-relationships, either through an additional external ontology or via annotations, and through a combination of open and closed world reasoning, can enable new capabilities that make such registries invaluable to their users, particularly for question answering over large distributed repositories.

One recent effort employed OWL to evaluate the BRIDG Domain Analysis Model (DAM) – a multi-agency effort to develop a shared view of the data, relationships, and processes which collectively define "protocol-driven research and its associated regulatory artifacts." The first formal release of BRIDG, using the Unified Modeling Language (UML) [14] was published in June 2007 [5]. In late 2008, NCI commissioned a study to use the Ontology Definition Metamodel (ODM) [8] to translate the BRIDG DAM from UML to OWL in order to help identify potential problems with BRIDG as a data model. The absence of built-in constructs for comparative purposes resulted in a lot of manual effort to ensure that comparisons between versions of the model were apples-to-apples. The assignment of immutable identifiers and versions to the elements of the owl model emulate an ISO/IEC 11179 MDR approach.

The resulting analysis concluded that there were several flaws and ambiguities in the BRIDG model, including problems with relationship names and types, and what was termed an "*explosion of attributes in the model...due to the creation of a new Common Data Element (CDE) when the concept is the same but the context of use is different."* The report cites postalAddress as an example: *"This attribute occurs 8 times in the model, all with the same AD datatype expression, and all referencing a physical postal address of an entity but all with a slightly different definition."* In this case, however, showing different variants could be considered a feature or strength of the BRIDG model rather than a "bug," depending on what purposes the model is intended to serve. A more general solution might be to have a single generic postalAddress class, with subclasses for different variants that are used by different groups and agencies. The new draft ISO/IEC 11179 (E3) metadata registry standard and XMDR prototype support just this kind of capability to document and manage the evolution of application and database-specific variants while at the same time showing their commonalities and translation requirements. If the semantics of each component of a particular variant are identical then they should be modeled as one object, but if they differ even slightly, it often is helpful to be able to distinguish and identify those subtle differences, as well as to note how one may be transformed to the other, with or without loss of information in one or both directions.

Whether we use OWL, UML, or other representation paradigms such as Entity-Relationship modeling, each of which can serve multiple purposes in the context we've described, we still need an additional level of abstraction for management of both data model and ontology information, along with the relationships between them in order to document, manage, and harmonize data and semantics from diverse systems – particularly as they evolve over time.

4 ISO/IEC 11179 Metadata Registry Standard

The first edition of ISO/IEC 11179, which became a formal standard in 1994, established a discipline for standardization of data elements such as those found in databases and data interchange. Organizations leading the development of this standard, including DOD, EPA, and NCI, recognized the need for better ways to represent relationships, and to express relational semantics sufficient to enable machine-processing and elementary logical operations. Subsequent editions add tighter linkage between the semantics found in data, metadata and concept systems. Edition 3 of ISO/IEC 11179 Part 3 provides explicit specifications for registering ontologies, thesauri, taxonomies and other semantic artifacts useful in managing the semantics of data. Work on additions and extensions to ISO/IEC 11179 Edition 2 began in 2004. At the same time, staff at the Lawrence Berkeley National Laboratory began to develop a prototype system to demonstrate the feasibility of implementing software that could be used to load, manage, and query an eXtended Metadata Registry, as specified by the evolving third edition working drafts.

ISO/IEC 11179-3 (E3) provides a means by which multiple definitions of particular terms (be they defined via ontologies, taxonomies, other kinds of models, or other metadata), can be articulated and reused across systems and modeling paradigms. It includes a rich model for provenance about the models managed in a compliant registry, providing a means by which one can say "what sources were used to develop this model/concept", "where is this model/concept used", "who depends on it", "what domain was it intended for", etc. -- which the modeling paradigms themselves do not do (nor do they claim to do). It also supports data stewardship, change management/change control, management of technical status and management of the status of organizational acceptance. Figure 3 shows a summary of the top level classes and a few of their major sub-classes from the current draft of ISO/IEC 11179-3 (E3).

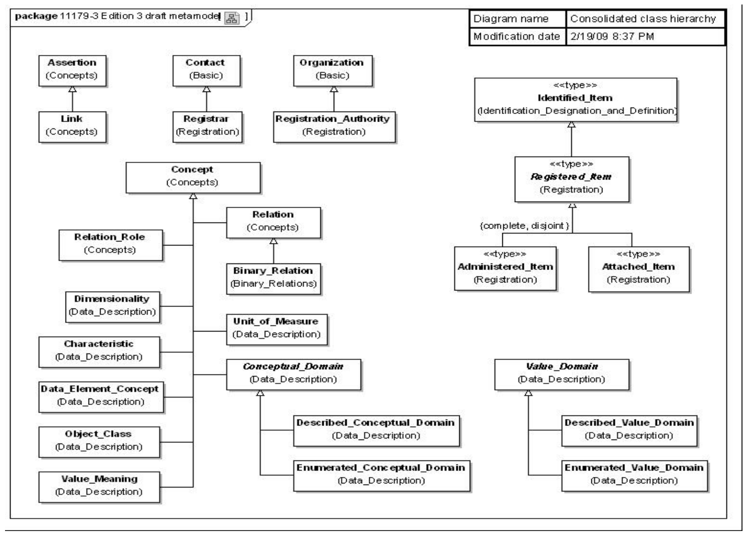

package 11179-3 Edition 3 draft metamodel
Diagram name
Consolidated class hierarchy
Modification date
2/19/09 8:37 PM
Assertion (Concepts)
Link (Concepts)
Contact (Basic)
Registrar (Registration)
Organization (Basic)
Registration_Authority (Registration)
<<type>> Identified_Item (Identification_Designation_and_Definition)
<<type>> Registered_Item (Registration)
{complete, disjoint}
<<type>> Administered_Item (Registration)
<<type>> Attached_Item (Registration)
Concept (Concepts)
Relation (Concepts)
Relation_Role (Concepts)
Binary_Relation (Binary_Relations)
Dimensionality (Data_Description)
Unit_of_Measure (Data_Description)
Characteristic (Data_Description)
Conceptual_Domain (Data_Description)
Value_Domain (Data_Description)
Data_Element_Concept (Data_Description)
Described_Conceptual_Domain (Data_Description)
Described_Value_Domain (Data_Description)
Object_Class (Data_Description)
Enumerated_Conceptual_Domain (Data_Description)
Enumerated_Value_Domain (Data_Description)
Value_Meaning (Data_Description)

Figure 3. Current Draft ISO/IEC 11179-3 (E3) Consolidated Class Hierarchy

Leading the progression from ISO/IEC 11179 (E2) to (E3), NCI added linkage from the semantic metadata content of the metadata registry to controlled terminology, thus providing a more granular way to compare Object Classes and Properties, and providing a way to link other metadata to controlled terminology so researchers can explore/discover what might be similar or related, to the data item of interest. An overview of the architecture, current implementation, including the use of OWL and UML models, and directions this work is taking at NCI as a part of the cancer Biomedical Informatics Grid (caBIG) program is presented in [17].

5 OWL AND THE 11179 XMDR PROTOTYPE

UML models, OWL ontologies, and XML schemas each capture important, but not identical content useful for design and implementation of information systems. The XMDR project demonstrated this at two levels for metadata registries. First, based on formative ISO/IEC 11179 (E3) specifications, the XMDR was designed to capture and interrelate selected content of UML models, XML schemas, and OWL ontologies, as well as concept systems in terms of semantic relationships as well as traditional metadata. Second, UML models, XML schemas and OWL based technologies were used to implement the XMDR prototype.

In response to user needs for extensions to the ISO/IEC 11179 (E2) metamodel such as the extensions implemented in NCI's caDSR, plus the need to test whether such extensions could be practically implemented and deployed using real data and concept systems, the XMDR project developed several types of metadata registry extensions including:

1. standardized representation of logical and other types of relationships;

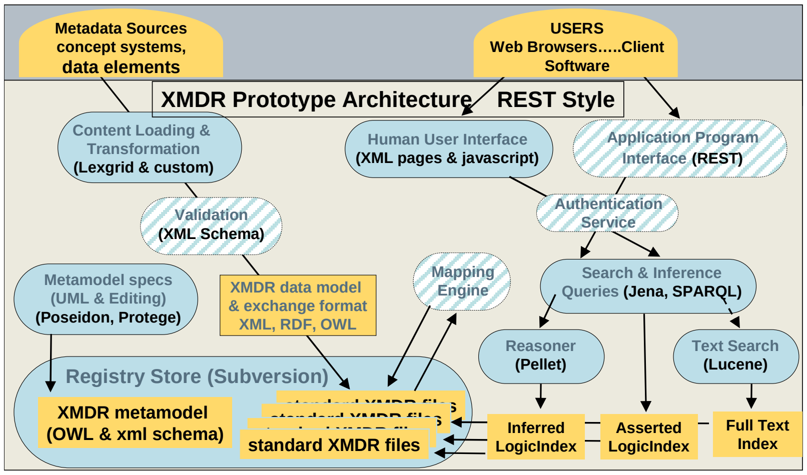
2. a metamodel framework to facilitate controlled and well-documented management and evolution of terminologies, thesauri, concept systems and ontologies in the same way that data elements, value domains, and related types of information are managed in the ISO/IEC 11179 (E2) framework;

3. use of OWL to permit and facilitate reasoning based on logical inference.

After reviewing a number of candidate languages and software packages, LBNL implemented the XMDR prototype as a modular, open architecture system, which makes it relatively easy to substitute software modules for particular components (*e.g.*, database system, reasoner).

Figure 4 shows the over-all modular component architecture of the XMDR prototype, along with particular open source software used for its current components. In this schematic diagram, ovals or rounded boxes depict major components of the XMDR system, while rectangles represent data, metadata, and indexes. Planned extensions are shown in shaded ovals. The XMDR Prototype used Poseidon and Protégé to create and edit UML and an OWL ontology that describes the XMDR metamodel. These specify how metadata is organized within the metadata registry store. The diagram also shows how content is transformed and loaded -- using LexGrid and custom XSLT scripts to create standard "XMDR files." XMDR files are indexed using Jena and Pellet to create RDF files that are stored in a Postgres database, and then further processed using Lucene to create a text index. Human users (using web browsers) can create queries and display results using a combination of JSP code and a standard REST interface. Other software (such as Exhibit[http://www.simile-widgets.org/exhibit/], from MIT's Simile Project [http://simile.mit.edu/]) can make use of the REST Application Programming Interface (API) to provide a "plug and play" Graphical User Interface (GUI).

Figure 4: XMDR Prototype Modular Architecture: with current open source software selections


Metadata Sources concept systems, data elements
USERS Web Browsers.....Client Software
XMDR Prototype Architecture REST Style
Content Loading & Transformation (Lexgrid & custom)
Human User Interface (XML pages & javascript)
Application Program Interface (REST)
Validation (XML Schema)
Authentication Service
Mapping Engine
Search & Inference Queries (Jena, SPARQL)
Metamodel specs (UML & Editing) (Poseidon, Protege)
XMDR data model & exchange format XML, RDF, OWL
Reasoner (Pellet)
Text Search (Lucene)
Registry Store (Subversion)
XMDR metamodel (OWL & xml schema)
standard XMDR files
Inferred LogicIndex
Asserted LogicIndex
Full Text Index

The XMDR prototype demonstrates how technologies based on UML, OWL and XML Schemas can be used to create a metadata registry, which is used to register, manage, and curate selected content from UML models, XML Schemas, and concept systems (including ontologies, thesauri, taxonomies, etc.) along with traditional metadata such as descriptions of common data elements, value domains, etc. Future plans include replacing Poseidon with ODM-based UML capabilities to leverage forward and reverse engineering of OWL from UML, completion of planned modules, and revision to support the evolving ISO/IEC 11179-3 (E3) standard as it approaches formal adoption.

6 Future Research

While the ISO/IEC 11179 standard itself has been in use for over a decade, its application to the management and use of ontologies and other concept systems is relatively new. Some of the organizations involved in the writing of this paper, and other colleagues within the US/ANSI DM32.8 task force and in the broader international metadata standards community have been actively involved in this evolution for at least the last five years. Implementations are nascent, and the standard itself is just now reaching final committee draft stage. We anticipate additional work will be needed to support development of reference implementations and further evolve the standard, informed by those efforts, before finalization is complete. Integration of increasing levels of automation, through ODM-based tools, through the use of reasoning to facilitate validation, search and retrieval, and other analysis activities are also planned. Evolution of the core ISO/IEC 11179 ontology, potentially reflecting the availability of new features in OWL 2, is also on our roadmap.

7 Conclusion

UML, OWL, and ODM have been instrumental in enabling better ways of characterizing and documenting information models and knowledge but they only provide the basic representation language constructs required as a starting point. Our experiences with the BRIDG 2.0 domain analysis model, development of the new draft ISO/IEC 11179 (E3) Metadata Registry Standard, and implementation of the Extended Metadata Registry (XMDR) Prototype System help illustrate the need to combine these technologies with those such as UML/ODM, OWL, and the Semantic Web with more traditional metadata registry tools and procedures to begin to address some of the really difficult information interchange issues faced by large organizations today. Utilizing a combination of ontology and knowledge representation languages, linked with standard metamodels for describing data and metadata provide an unprecedented opportunity to leverage semantic web for knowledge mining and discovery of hidden links and improve their utility in data management.

Disclaimer

This article is based in part upon work supported by the Environmental Protection Agency, Department of Defense, National Cancer Institute, and the National Science Foundation under NSF Grant No. 0637122. Any opinions, findings, and conclusions or recommendations expressed in this article are those of the authors and do not necessarily reflect the views of the sponsors.

References

1. Mike Dean and Guus Schreiber, eds., Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL Web Ontology Language 1.0 Reference, W3C Recommendation, World Wide Web Consortium, Amsterdam, The Netherlands, 10 February 2004. Latest version is available at http://www.w3.org/TR/owl-ref/.
2. Tom Gruber (1993). "A translation approach to portable ontology specifications". In: *Knowledge Acquisition*. 5: 199
3. ISO/IEC 11179 – Metadata Registries - Part 3 (Edition 3), Available at http://metadata-stds.org/11179/index.html#A3.
4. ISO/IEC 19763, Information Technology – Metamodel Framework for Interoperability (MFI), Available at http://metadata-standards.org/19763/index.html.
5. Christopher A. Welty and Elisa F. Kendall, eds. Ontology Definition Metamodel (ODM), Version 1.0 Specification, Object Management Group, Inc., Needham, MA, May 2009. Latest version is available at: http://www.omg.org/spec/ODM/1.0/.
6. http://gforge.nci.nih.gov/projects/bridg-model and
7. http://bridgmodel.org/
8. https://gforge.nci.nih.gov/projects/bridgrev
9. http://informatics.mayo.edu/LexGrid/

10.http://XMDR.org/
11.Peter A. Covitz∗, Frank Hartel, Carl Schaefer, Sherri De Coronado, Gilberto Fragoso, Himanso Sahni, Scott Gustafson and Kenneth H. Buetow, *caCORE: A common infrastructure for cancer informatics*, Bioinformatics, Vol. 19 no. 18 2003, pages 2404–2412
12. Komatsoulis, GA, Warzel, D.B, Hartel, F.W., Shanbhag, K, Chilukuri, R, Fragoso, G., de Coronado, S, Reeves, D.M, Hadfield, J.G., Ludet, C., and Covitz, P. A. (2008) *caCORE version 3; Implementation of a model driven service-oriented architecture for semantic interoperability*. Journal of Bioinformatics, 41:1
13.caDSR http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/cadsr and EVS http://bioportal.nci.nih.gov/ncbo/faces/index.xhtml statistics citation
14.EDR http://iaspub.epa.gov/sor_internet/registry/datareg/home/overview/home.do
15.Berners-Lee, T, http://www.w3.org/DesignIssues/RDFnot.html
16. Unified Modeling Language™ (UML®) Infrastructure and Superstructure Specifications, Version 2.1.2, Object Management Group, Inc., Needham, MA, November 2007. Latest versions of this and related UML specification components is available at: http://www.omg.org/spec/UML/2.1.2/.
17.Alejandra González Beltrán, Anthony Finkelstein, J. Max Wilkinson, and Jeff Kramer *(forthcoming)*. “Domain Concept-Based Queries for Cancer Research Data Sources”.