Knowledge Engineering rediscovered: Towards Reasoning Patterns for the Semantic Web

Frank van Harmelen
Vrije Universiteit Amsterdam
Frank.van.Harmelen@cs.vu.nl

Annette ten Teije
Vrije Universiteit Amsterdam
annette@cs.vu.nl

Holger Wache
University of Applied Sciences Northwestern Switzerland
holger.wache@fhnw.ch

ABSTRACT

The extensive work on Knowledge Engineering in the 1990s has resulted in a systematic analysis of task-types, and the corresponding problem solving methods that can be deployed for different types of tasks. That analysis was the basis for a sound and widely accepted methodology for building knowledge-based systems, and has made it is possible to build libraries of reusable models, methods and code.

In this paper, we make a first attempt at a similar analysis for Semantic Web applications. We will show that it is possible to identify a relatively small number of task-types, and that, somewhat surprisingly, a large set of Semantic Web applications can be described in this typology. Secondly, we show that it is possible to decompose these task-types into a small number of primitive ("atomic") inference steps. We give semi-formal definitions for both the task-types and the primitive inference steps that we identify. We substantiate our claim that our task-types are sufficient to cover the vast majority of Semantic Web applications by showing that all entries of the Semantic Web Challenges of the last 3 years can be classified in these task-types.

1. INTRODUCTION

1.1 Lessons from Knowledge Engineering

Starting with the seminal work by Clancy [4], research in Knowledge Engineering has developed a theory of generic *types of tasks*, which can be implemented by a generic set of *problem solving methods*, decomposable into *primitive inference steps*. Examples of the task types that were identified are `diagnosis`, `design`, `scheduling`, etc. Via the problem solving methods (e.g. a `pruning` method for the classification task) these tasks could be decomposed into elementary inference steps such as `generate-candidate`, `specify-attribute`, `obtain-feature`, etc.

This work has lead to well-founded methodologies for building knowledge-based systems out of reusable components. The CommonKADS methodology [12] is perhaps the best known of such methodologies, although certainly not the only one. Another example is the generic tasks approach by Chandrasekaran [3]. We will use the terminology of the CommonKADS approach, and will identify tasks and inferences in the context of the semantic web.

This work in Knowledge Engineering originated from a shared frustration about the lack of reusable components for building knowledge-based systems. If "Knowledge Engineering" was really "engineering", where were the reusable components, and why did every implementor have to start from scratch? The important insight was to describe the tasks that Knowledge Based Systems perform at a sufficiently abstract level, the "Knowledge Level", introduced by Newell in his 1980 AAAI presidential address [9], later in [10]. Once the discussion moved from the implementation details on the "symbol level" to the more abstract "knowledge level", it became possible to identify generic task-types, reusable problem solving methods and reusable elementary inference steps. Since then, libraries of reusable components have been published both in books (e.g. [2]) and on websites (e.g. `http://www.commonkads.uva.nl`), and are now in routine use.

As is well known from other branches of engineering, reusable components help to both substantially increase the quality of design and construction as well lowering the costs by reusing tried-and-tested design patterns and component implementations. Menzies [8] illustrates the reuse benefits for reasoning patterns.

1.2 Applicable to Semantic Web engineering?

The above raises the question if similar lessons can be applied to Semantic Web engineering. Can we identify reusable patterns and components that can help designers and implementers of Semantic Web applications?

It hardly needs arguing that work on the Semantic Web has put great emphasis on the reusability of *knowledge*, in the form of ontologies. It is fair to say that insights about reusable knowledge elements have been at the birthplace of the Semantic Web enterprise. The idea of reusable ontologies has also been generalised into work on reusable *ontology patterns* (e.g. [1, 7]).

However, all of this work deals with reusable *knowledge* elements. There is little if any work on reusable *reasoning* patterns. *This paper is a first attempt at finding reusable reasoning patterns for SemWeb applications.*

1.3 Structure of the paper

After discussing related work (section 2) and some formal preliminaries (section 3), the paper is structured in the following steps:

1. **Identify typical task types** and give semi-formal definitions to characterise them (section 4.1);
2. **Validate the task types** by showing that a large number of representative and realistic Semantic Web applications can be classified into a limited number of such task types (section 4.2);
3. **Define primitive inference steps** through semi-formal definitions (section 5.1);
4. **Validate the primitive inference steps:** show that the identified task types can be decomposed into the given inference steps. (section 5.2).

If the above steps would succeed, this would be of great value to Semantic Web application builders, leading to the possibility of libraries of reusable design patterns and component implementations. It would also constitute an advance in our understanding of the landscape of Semantic Web applications, which has until now mostly grown bottom up, driven by available technical and commercial opportunities, with little or no theory-formation on different types of applications and their relationships.

2. RELATED WORK

As described above, most if not all, work on reusability for the Semantic Web has focussed on reusable *knowledge*, to the exclusion of reusable *reasoning patterns*. More or less the only exception that we are aware of is the work by Oren [11, 5]. Although similar in approach (they also survey the past years of Semantic Web Challenge entries to detect recurring patterns in these applications), their focus is rather different. In Knowledge Engineering terms, they are focusing more on "symbol level" issues such as architectural components, programming language used and (de)centralisation of the architecture, whereas we are interested in a "knowledge level" analysis that is independent of implementation details. The one element in Oren's analysis that comes closest to our goals is his "application type", which has a large overlap with our notion of "task types". However, Oren then links these application types to required architectural components (storage, user-interface, etc), but does not link them into primitive *reasoning* steps, which is the goal that we are pursuing.

[6] analyses 33 semantic search applications with a similar aim to ours (discovering re-usable components) but more limited than ours (considering only search applications), and not attempting any (semi-)formal definitions.

From this brief analysis, we conclude that ours is the first attempt at a systematic analysis of Semantic Web *reasoning* patterns.

3. FORMAL PRELIMINARIES

We will use the terms terminology, ontology, class, concept and instance as follows: a terminology is set of class-definitions (a.k.a. concepts) organised in a subsumption hierarchy; instances are members of such concepts. Ontologies consist of terminologies and sets of instances.

More formally we will consider an *ontology* O as a set of triples $\langle s, p, o\rangle$, where $\in$ and $\subseteq$ are special cases of p. In other words, we consider two specific predicates: $\subseteq$ for the subsumption relation, and $\in$ for the membership relation. We use $\langle c_1, \subseteq, c_2\rangle$ to denote that a class c_1 is subsumed by a class c_2. We use $\langle i, \in, c\rangle$ to denote that an individual i is a member of a class c. A *terminology* T is a set of triples whose predicate is the subsumption $\subseteq$ only, and an *instance* set I is a set of triples which predicate is the membership relation $\in$ only. T resp. I can be extracted from ontology O with the function $T(O)$ resp. $I(O)$. An ontology is the union of its terminology, its instance set, and possibly triples $\langle s, p, o\rangle$ using other relations p: $O \supseteq T \cup I$. We will overload the $\in$ notation and also use it to denote that a triple is a member of a set (as in: $\langle s, p, o\rangle \in O$). We do *not* assume that triple-sets are deductively closed. We will use $\vdash$ to denote that a triple can be *derived* from a set (as in: $O \vdash \langle s, p, o\rangle$), using some appropriate semantics (e.g. RDF Schema or OWL DL derivations). O^* contains all triples $\langle s, p, o\rangle$ which can be derived from O. Please note that O^* contains O and may be infinite. We will use lower case letters c, i for a single concept or instance, and uppercase letters C, I for concepts sets containing $\langle c_i, \subseteq, c_j\rangle$ or instance sets containing $\langle i, \in, c_k\rangle$. We will often use the terms "ontologies" and "knowledge" as interchangeable.

4. TASK TYPES

In this section we will first (section 4.1) identify a limited number of general task types and give semi-formal definitions for each of them. Notice that these tasks are identified for the semantic web application in the same way that the tasks of the CommonKADS framework

(like diagnosis etc.) are meant for knoweldge based systems. The selection of the tasks represents the most prominent ones which can be found in current semantic web application; the selection is not intented to be complete. For each of these task types, we give the most common definition of that task, although we show in places that variations in these definitions are possible. Subsequently (section 4.2) we will show how a representative set of Semantic Web applications can all be understood as instances of this small set of task types.

4.1 Defining Semantic Web task types

We will characterise seven different task types. For each of them, we will give an informal description, the signature (types of their input and output parameters), and a semi-formal definition of the functionality (relation between input and output).

Search:

Perhaps the most prototypical Semantic Web application is search, motivated by the low precision of current search engines. Traditional search engines take as their inputs a query (usually in the form of a set of keywords) plus data-set of instances (usually a very large set of web-pages), and return a subset of those instances. A Semantic Web search engine would take a query in the form of a concept description, this concept description would be matched against an ontology (ie. a terminology used to organise an instance set), and members of the instance-set matching the query-concept would be returned. Hence, search is a mapping which maps a given concept c and ontology O to a set of instances $I' \subseteq I(O)$:

Search: $c \times O \mapsto I'$

input request: a concept c
input knowledge: an ontology O, hence consisting of a terminology $T(O)$ (a set of $\langle c_i, \subseteq, c_j \rangle$), and an instance set $I(O)$ (set of $\langle i, \in, c_k \rangle$).
answer: $search(c, O)$ returns an instance set such that:

$$search(c, O) = \{\langle i, \in, c \rangle | \exists c_j : \begin{array}{l} \langle c_j, \subseteq, c \rangle \in T(O) \wedge \\ \langle i, \in, c_j \rangle \in I(O)) \end{array} \}$$

In other words: $search(c, O)$ returns all instances i that are known to be members of subconcepts of c (and hence are members of c as well).

Notice that this definition only returns instances of concepts c_j that are *known* to be a subconcept of c (since we demanded $\langle c_j, \subseteq, c \rangle \in T(O)$). An alternative definition would be to allow the use of deductive machinery to *derive* the subconcepts of c:

$$search(c, O) = \{\langle i, \in, c \rangle | \exists c_j : \begin{array}{l} O \vdash \langle c_j, \subseteq, c \rangle \wedge \\ O \vdash \langle i, \in, c_j \rangle \end{array} \}$$

Instead of $O \vdash \langle c_j, \subseteq, c \rangle$ (resp. $O \vdash \langle i, \in, c_j \rangle$) we can write $\langle c_j, \subseteq, c \rangle \in T(O^*)$ (resp. $\langle i, \in, c_j \rangle \in I(O^*)$).

Browse:

Browsing is very similar to searching (and often mentioned in the same breadth), but has as crucial difference that its output can either be a set of instances (as in search), or a set of concepts, that can be used for repeating the same action (ie. further browsing).

Thus, browse is a mapping which maps a given concept c and ontology O to a set of instances I' plus a set of concepts C:

Browse: $c \times O \mapsto I' \times C$

input request: a concept c
input knowledge: an ontology O consisting of a terminology $T(O)$ and an instance set $I(O)$
answer: $browse(c, O)$ returns a set of instances and a set of concepts such that:

$$\begin{array}{l} browse(c, O) = search(c, O) \times \\ \quad \{\langle c_j, \subseteq, c \rangle | \langle c_j, \subseteq, c \rangle \in T(O) \wedge \neg \Delta(c_j, c, O)\} \cup \\ \quad \{\langle c, \subseteq, c_j \rangle | \langle c, \subseteq, c_j \rangle \in T(O) \wedge \neg \Delta(c, c_j, O)\} \\ \text{with } \Delta(c_j, c, O) \leftrightarrow \exists c_k : \begin{array}{l} \langle c_k, \subseteq, c \rangle \in T(O) \wedge \\ \langle c_j, \subseteq, c_k \rangle \in T(O) \end{array} \end{array}$$

Besides instances that a user might be interested in (based on the given input concept c), this returns the immediate neighbourhood of c (immediate sub- and superconcepts of c known in T), to be used for repeated browsing by the user. As with search, alternative definitions are possible by returning a wider neighbourhood for c, consisting also of indirect sub- and super-concepts, or by deducing a neighbourhood of c instead of being limited to the explicitly known neighbourhood (using $\vdash$ instead of $\in$).

Data integration:

The goal of data-integration is to take multiple instance sets, each organised in their own terminology, and to construct a single, merged instance set, organised in a single, merged terminology. Hence, data integration is a mapping which maps a set of ontologies to a (new) ontology.

Integrate: $\{O_1, \ldots, O_n\} \mapsto O'$

input request: multiple ontologies O_i with their terminologies $T_i = T(O_i)$ and their instance sets $I_i = I(O_i)$.
answer: a single ontology O' with terminology T' and instance set I'

$$\begin{array}{l} integrate(\{O_1, .., O_n\}) = O' \\ \text{such that } I' = \bigcup I_i \text{ and } T' \supseteq \bigcup T_i \end{array}$$

It is difficult to give a more specific I/O-condition to

characterise data-integration. Typically (but not always), all input instances are part of the output ($I' = \cup I_i$), and typically (but not always), the output terminology consists of all the input terminologies ($T' \supseteq \cup T_i$), enriched with relationships between elements of the different T_i, such as $\langle c_i, \subseteq, c_j \rangle$, $\langle c_i, \texttt{sameAs}, c_j \rangle$ or other relationships.

Personalisation and recommending:

Personalisation consists of taking a (typically very large) data set plus a personal profile, and returning a (typically much smaller) data set based on this user profile. The profile which characterises the interests of the user can be in the form of a set of concepts, or a set of instances. For instance typical recommender services at on-line shops use previously bought items, which are instances, while news-casting sites typically use general categories of interest, which are concepts.

$$personalise : I_{data} \times I_{profile} \times O \mapsto I_{selection}$$
or
$$personalise : I_{data} \times C_{profile} \times O \mapsto I_{selection}$$

Personalise: $I_{data} \times C_{\textbf{profile}} \times O \mapsto I_{selection}$

input request: an instance set I_{data} of triples $\langle i, \in, c_j \rangle$ and a profile characterised as either a set of instances $I_{profile}$ or a set of concepts $C_{profile}$.
input knowledge: an ontology O
answer: a reduced instance set $I_{selection}$ with

$$personalise(I_{data}, C_{profile}, O) = \{\langle i, \in, c' \rangle | \exists c : \; i \in I_{data} \wedge c \in C_{profile} \wedge O \vdash \langle c', \sim, c \rangle \wedge \langle i, \in, c' \rangle \in I(O)\}$$

That is, personalisation returns instances that are members of concepts which are in some way related to the target concept(s) through some relevant relation $\sim$. Interestingly, if we take $\sim$ to be $\subseteq$, this becomes essentially equivalent to our above definition of search. In general, one could also return instances that are "almost" members of the profile-concepts (ie. they are instances of concepts that are not subsumed by but closely related to the target concepts).

Web-service selection:

Rather than only searching for static material such as text and images, the aim of semantic web services is to allow searching for active components, using semantic descriptions of web-services. We can then regard a concept c as the description of some web-service functionality, and an instance i as a particular web-service. Membership $i \in c$ is then interpreted as "service i implements specification c", and $c_i \subseteq c_j$ as "specification c_i is a specialisation of specification c_j" (and consequently, every service i that implements specification c_i also implements specification c_j). Just for mnemonic reasons, we will use f for functionality instead of c, and s for service instead of i, and similarly S for a set of services instead of I.

At the level of the signature, the characterisation of this task is the same as that of general search:

Service selection: $f \times O \mapsto S'$

input request: required functionality f
input knowledge: an ontology O containing a set of candidate services $S(O)$ and a hierarchy of service specifications $T(O)$.
answer: members of the candidate set whose specification satisfies the required functionality:

$$service\_selection(f, O) = \{\langle s, \in, f \rangle | \exists f_j : \; \langle f_j, \subseteq, f \rangle \in T(O) \wedge \langle s, \in, f_j \rangle \in S(O) \}$$

The difference with search is of course that the query describes functionality (rather than content), and the candidate set consists of services. In general, this will make the membership relation $\in$ and the containment relation $\subseteq$ much harder to compute than in the case of search (where we deal with static data instead). Of course, the different variations that we gave for the definition of search (e.g with or without deduction) can be applied here as well.

Web-service composition:

An even more ambitious goal than web-service selection is to compose a given number of candidate services into a single composite service with a specific functionality. The input of web-service composition is the same as for the selection of a single web-service above, but the output can now be an arbitrary control flow over a set of web-services. We will informally denote such a flow with $\mathcal{FLOW}$ without further specifying this. This then results in a similar specification as search:

Service composition: $f \times O \mapsto \mathcal{FLOW}$

input request: required functionality f
input knowledge: an ontology O with set of candidate services $S(O)$ and a hierarchy of service specifications $T(O)$.
answer: members of the candidate set whose compound specification $\mathcal{FLOW}$ satisfies the required functionality:

$compose(f, O) = \mathcal{FLOW}$
such that $T(O) \vdash \langle \mathcal{FLOW}, \in, f \rangle$, and $\langle s, \in, \_ \rangle \in S(O)$ for each service s occurring in $\mathcal{FLOW}$

ie. the hierarchy of specifications in $T(O)$ allows us to infer that the computed $\mathcal{FLOW}$ satisfies the required functionality, and $\mathcal{FLOW}$ must be composed of services taken from $S(O)$.

Semantic Enrichment

This task type is concerned with annotating objects, such as images or documents, with meta-data. Such added meta-data can be used by task types like search or browse to increase the quality of their answers. It maps a single instance i to a set of triples about that instance:

Semantic enrichment: $i \mapsto I$
input request: an instance i to be enriched **answer:** a set of triples $I = \{\langle s, p, o\rangle \mid s = i\}$ that all have i as their subject.

Notice that we have allowed here triples with other relation-symbols besides $\in$ or $\subseteq$, allowing for other, domain specific, properties.

4.2 Validating the task types

Of course the key question at this point is: how reusable is the above set of task types? Can most Semantic Web applications be described in terms of these task types? Is this small set of seven task-types sufficient, or will we end up inventing new task types for every new application (hence defeating the goal of reusability)?[1]

In order to measure the completeness and reusability of our list of task types, we have analysed all entries to the Semantic Web Challenge events of the years 2005, 2006 and 2007 to see if they could be properly described with our task types. As is well known, the "Semantic Web Challenge"[2] is an annual event that stimulates R&D by showing the state-of-the-art in Semantic Web applications every year. It gives researchers an opportunity to showcase their work and to compare it to others. Since the competition is very unconstrained, and allows submissions of a large variety of Semantic Web applications, we claim that the collected entries over a number of years together provide a representative sample of state of the art Semantic Web applications, and are hence a suitable data-set for verifying the completeness and reusability of our list of task types. It is noteworthy that in his independent analysis, Oren [11, 5] also turned to the entries in the Semantic Web Challenge as a valid dataset.

Figure 3 shows the results of our analysis. It covers all entries to the 2005, 2006 and 2007 competitions with the exception of a small number of applications about which we could not obtain any information, and a single application for which we were not able to understand the functionality. The analysis in figure 3 leads us to the following main observations:

• **A**ll but one of the applications could be classified in terms of our task-types. The single missing application (SMART, from 2007) can best be described as performing "question answering". This would indeed be a valid (and reusable) expansion of our list of task-types, but we were unable to come up with a reasonably formal definition of this task-type.

• **O**ften, a single application belongs to multiple task types. See for instance the prize winning e-culture application "MultimediaN" that performs a combination of searching, browsing, and semantic enrichment. This phenomenon is well known from Knowledge Engineering, where a single system also often performs multiple tasks (e.g. first diagnosis, then planning a treatment). Notice that Search and browse are often occur together in an application.

• **I**t is noticeable that the combined 2005-2007 Challenges do not contain a single submission that can be described as web-service selection. This raises some doubts as to the necessity of this task type. At the same time, there were some (although few) entries that could be properly described as web-service composition.

Taken altogether, we interpret these findings as support for the reasonable completeness and reusability of the task-types that we defined in section 4.1

5. PRIMITIVE INFERENCES

In this section we will define a number of primitive inference steps, and we will show that each of the task types identified before can be decomposed into a limited number of primitive inference steps.

The qualification "primitive" needs perhaps some explanation. Just as in the CommonKADS methodology, we interpret the term "primitive" to mean that from the application builder's point of view, it is not interesting to further decompose this step, ie. application builders would typically regard such a step as atomic. Of course this is not a hard criterion: sometimes it might be useful to further decompose such a step, e.g. for optimisation reasons. Also, what is a primitive, elementary, atomic component for an application builder might well be a highly non-elementary, non-atomic and very complex operation to implement. And indeed, many of the primitive inference steps that we define below have been subject to many years of research and development. Thus, "primitive" should *not* be read as "simple". It only means that this step will typically be regarded as atomic by application builders.

5.1 Defining primitive inference steps

In this section we define a small number of primitive inference steps for Semantic Web applications. We give a semi-formal definition of these primitive inferences, including their signature.

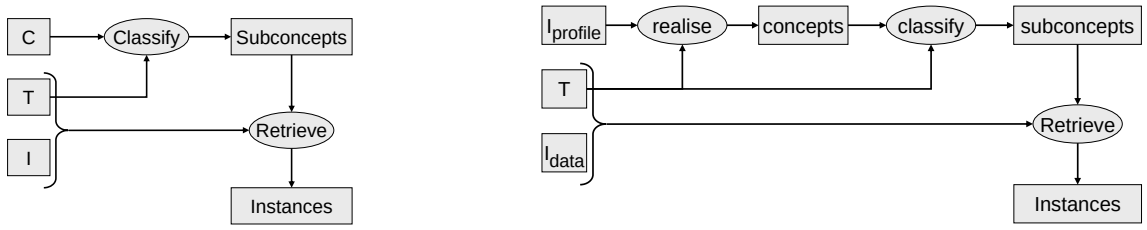
Realisation determines which concepts a given instance

[1]Please note we do not intgend to present a complte list of tasks but a mots prominent one.

[2]`http://challenge.semanticweb.org/`

task types	primitive inference steps			
	realisation	subsumption & classification	mapping	retrieval
search		x		x
browse	x	x		x
data integration	x	x	x	
personalisation	x	x		x
service selection		x		x
service composition		x		
semantic enrichment				

Figure 1: task types in terms of primitive inference steps.



(a) Search (b) Personalisation

Figure 2: Inference structures for the task types search and personalisation

is a member:
• **Signature:** $i \times O \mapsto c$
• **Definition:** Find a c such that $O \vdash i \in c$

Subsumption determines whether one concept is a subset of another:
• **Signature:** $c_1 \times c_2 \times O \mapsto bool$
• **Definition:** Determine whether $O \vdash c_1 \sqsubseteq c_2$

Mapping finds a correspondence relation between two concepts defined in the ontolgy O.[3] We follow the common approach, where the correspondence relation can be either equivalence, subsumption or disjointness:
• **Signature:** $c_1 \times c_2 \times O \mapsto \{=, \sqsubseteq, \sqsupseteq, \perp\}$
• **Definition:** find an $r \in \{=, \sqsubseteq, \sqsupseteq, \perp\}$ such that $c_1\ r\ c_2$

Retrieval is the inverse of realisation: determining which instances belong the given concept:
• **Signature:** $c \times O \mapsto i$
• **Definition:** find i such that $i \in c$

Classification determines where a given class should be placed in a subsumption hierarchy:
• **Signature:** $c \times O \mapsto (c_l, c_h)$
• **Definition:** Find a highest subclass c_l and a lowest superclass s_h such that $O \vdash c_l \sqsubseteq c \sqsubseteq c_h$

Our choice of these five primitive inference steps is not the only choice possible. For instance, both classification and mapping can be reduced to repeated subsumption checks, and are hence not strictly speaking required as separate inferences. Similarly, it is well known that subsumption in turn can be reduced to satisfiability. However, we have chosen the above five as primitive inference steps because they seem to constitute a conceptually coherent (although not formally minimal) set.

[3]The ontology O may be the union — not merger — of several ontologies.

5.2 Decomposing the task types to primitive inferences

The table in figure 1 shows how each of the task-types from section 4.1 can be decomposed into the primitive inferences described in section 5.1. (In the table, classification and subsumption have been merged in a single column since the former is the iterated version of the latter).

We lack the space to discuss all of these decompositions in detail, and will discuss only two examples:

Search: the description of the search task-type in section 4.1 shows that it is a combination of *classification* (to locate the query-concept in the ontology in order to find its direct sub- or super-concepts) followed by *retrieval* (to determine the instances of those concepts, which form the answers to the query).
Personalisation: If the personal profile in the personalisation task-type consists of a set of instances (e.g. previously bought items), then personalisation is a composition of *realisation* (to obtain the concepts that describe these instances), *classification* (to find closely related concepts), and *retrieval* (to obtain instances of such related concepts, since these instances might be of

interest to the user).

In a similar way, all of the prototypical task-types we described in section 4.1 can be implemented in terms of the small set of primitive inference steps described in this section, resulting in the decomposition shown in figure 1.

Notice that the table in figure 1 only displays the minimally required reasoning tasks for each task-type. For example, it is well possible to equip the search task with a mapping component in order to map the vocabulary of a user-query to the vocabulary of the ontology used during search. Similar additions could have been made for many other task-types.

Notice too that semantic enrichment can not be defined based on these reasoning tasks. The reason for this is that usually in reasoning the input facts are assumed to be given, while semantic enrichment deals with *constructing* these input facts. Hence, semantic enrichment cannot be seen in terms of inference steps, but it is the only task type in our list that suffers from not being decomposable into a combination of primitive inference steps.

The CommonKADS method [12] uses the notion of an *inference structure* to graphically depict the decomposition of a task into primitive inference steps, by showing the data-dependencies between the primitive inference steps that together make up a task. In figure 2, we show the inference structures for the Search and Personalisation task-types using the decomposition into primitive inference steps given above. This is what we consider as the reasoning patterns. Also, figure 2 shows the structural similarity between Search and Personalisation: it makes clear that Personalisation is essentially Search, but preceeded by a realisation-step to map instances to the concepts to which they belong.

6. CONCLUDING REMARKS

The main contribution of this paper has been to provide a first attempt at providing a *typology of semantic web applications*. We defined a small number of prototypical task-types, and somewhat surprisingly, almost all entries to three years of Semantic Web Challenge competitions can be classified into these task-types. We also showed how each of these prototypical task-types can be decomposed into a small number of primitive inference steps. This results in the following reusable components: the identified tasks, the inferences and the decomposition of task into inferences (the so-called reasoning patterns). Analogously to established practice in Knowledge Engineering, these results provide a first step towards a methodology for building semantic web applications out of reusable components.

We regard this work indeed as first steps towards this goal. We would expect the typology of task-types to grow beyond the current set of seven to cover a larger corpus of semantic web applications. Also, the details of our semi-formal definitions and our decompositions may well have to be adjusted over time. Nevertheless, such refinements would leave unaltered the general aim of our proposal, namely that a more structured, abstract and implementation independent analysis of the semantic web applications "at the knowledge level" will be necessary if we are to rise beyond the current ad hoc practices.

7. REFERENCES

[1] E. Blomqvist and K. Sandkuhl. Patterns in ontology engineering: Classification of ontology patterns. In C. C. et al., editor, *ICEIS (3)*, pages 413–416, 2005.

[2] J. Breuker and W. van de Velde. *Common KADS Library for Expertise Modelling*. IOS Press, 1994.

[3] T. Bylander and B. Chandrasekaran. Generic tasks for knowledge-based reasoning: The "right" level of abstraction for knowledge acquisition. *Int. J. of Man-Machine Studies*, 26(2):231–243, 1987.

[4] W. J. Clancey. Heuristic classification. *Artif. Intell.*, 27(3):289–350, 1985.

[5] B. Heitmann and E. Oren. A survey of semantic web applications. Technical report, DERI, Galway, 2007.

[6] M. Hildebrand, J. R. van Ossenbruggen, and L. Hardman. An Analysis Of Search-Based User Interaction On The Semantic Web. Technical Report INS-E0706, CWI, Amsterdam, 2007.

[7] L. Lefort, K. Taylor, and D. Ratcliffe. Towards scalable ontology engineering patterns. In *AOW '06: 2nd Australasian workshop on Advances in ontologies*, pages 31–40, 2006.

[8] T. Menzies. Object-oriented patterns: lessons from expert systems. *Softw. Pract. Exper.*, 27(12):1457–1478, 1997.

[9] A. Newell. The knowledge level (presidential address). *AI Magazine*, 2(2):1–20, 33, 1980.

[10] A. Newell. Reflections on the knowledge level. *Artif. Intell.*, 59(1-2):31–38, 1993.

[11] E. Oren. *Algorithms and Components for Application Development on the Semantic Web*. PhD thesis, Nat. Univ. of Ireland, Galway 2007.

[12] G. Schreiber, H. Akkermans, A. Anjewierden, R. de Hoog, N. Shadbolt, W. V. de Velde, and B. Wielinga. *Knowledge Engineering and Management: The CommonKADS Methodology*. ISBN 0262193000. MIT Press, 2000.

Application	Task Types						
	search	browse	data integr.	Personal-ization	service select.	service compos.	semantic enrichment
CONFOTO ('05)		x					x
DynamicView ('05)	x						x
FungalWeb ('05)	x		x				
Oyster ('05)	x						
Personal Reader ('05)			x	x			x
Service Execution ('05)						x	
COHSE ('06)	x	x					x
Collimator ('06)							x
Dartgrid ('06)	x		x				
Dbin ('06)			x	x			
EKOSS ('06)	x						
eMerges ('06)			x				
Falcon-S ('06)	x	x					
Foafing the Music ('06)				x			
Geo Services ('06)						x	
MultimediaN ('06)	x	x					x
Paperpuppy ('06)		x					
Semantic Wiki ('06)							x
ArnetMiner ('07)	x	x	x				
Cantabria ('07)	x	x	x				
CHIP ('07)		x		x			
DORIS ('07)	x						x
EachWiki ('07)							x
GroupMe ('07)							x
iFanzy ('07)			x	x			x
Int.ere.st ('07)		x					
JeromeDL('07)		x					x
MediaWatch ('07)	x		x				x
mle ('07)		x					x
Notitio.us ('07)		x					x
Potluck ('07)		x	x				
Revyu ('07)			x				x
RKB Explorer ('07)			x				
SemClip ('07)			x				
SMART ('07)							
swse ('07)	x		x				
wwwatch ('07)		x	x				

Figure 3: Classification of Semantic Web Challenge in our task-types.