MoRe Semantic Web Applications

Maksym Korotkiy[α] and Jan L. Top[αβ]

[α]Vrije Universiteit Amsterdam, Department of Computer Science
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
[β]Wageningen Centre for Food Sciences
P.O. Box 557 6700 AN Wageningen, The Netherlands
maksym@few.vu.nl jltop@few.vu.nl

Abstract. We present *MoRe* – a framework that allows one to extend a domain ontology with a remotely invocable reasoning service applicable to concepts defined in that ontology. Our approach bridges the gap between ontology and application developers. We allow any reasoning service to be wrapped by a *MoRe* ontology extension, the services ranging from generic logics-based reasoners to specific black box software components. The application developer directly accesses these reasoning services through documents stated in terms of domain concepts rather than dealing with remote procedure calls. We describe a case that applies *MoRe* to an OWL-ontology of units of measure, and we demonstrate how this extended ontology can be integrated into a unit conversion application.

1 Introduction

Ultimately, the Semantic Web [1] aims to significantly improve the experience of web application end-users. To achieve this, the Semantic Web is to provide an environment that enables an application developer to advance web applications beyond what we observe nowadays. Ontologies are the keystones of the Semantic Web, and ontology developers play a crucial role in developing that enabling environment.

It is believed that the current approaches to the Semantic Web make it rather difficult to develop applications needed so much to materialize the Semantic Web vision [2, 3]. A number of solutions to facilitate design of Semantic Web applications has been proposed, ranging from authoring [4], browsing and annotation frameworks [5, 6] to infrastructures for the Semantic Web Services [7]. These approaches address specific aspects of the Semantic Web application development. In *MoRe* we take a step back to see how ontologies can be extended to make them more attractive for application development in general.

Presently, application developers do not profit much from the increasing availability of domain ontologies. The latter are typically devised for representation of static domain knowledge, whereas applications require problem-specific answers and computations. Generic reasoners and query languages associated with formalisms like RDFS or OWL are often not expressive, efficient and transparent enough to be used in applications. We propose *MoRe* [1] – a simple approach to extend ontologies with application-oriented, but

[1] "MoRe" used to be an acronym but with the development of our approach its original interpretation has become irrelevant.

still generic concepts and reasoning services. This provides for solutions, arbitrarily on the continuous scale between domain- specific ontologies (using generic reasoners) to task-specific applications (using dedicated procedures). The objective of our approach is to simplify application development by means of increased (re)usability of ontologies.

To improve the usability of ontologies *MoRe* extends them by providing an elementary mechanism for attaching a reasoning service to these ontologies. We believe that the availability of a readily accessible reasoning service allows the application developer to faster evaluate an ontology and to incorporate it more readily into an application.

Our approach can be illustrated by providing an example from e-Science, our field of application. e-Science aims at providing automated support to researchers in performing experiments and constructing models and theories. A simple but very important quality requirement for scientific work is correct and consistent use of units of measure. Traditionally, an application developer would construct a specific algorithm for unit conversion, using an internal database of units and their values in terms of reference units. With the availability of a units ontology, the application developer could instead access this ontology to determine for example the conversion factor between two units. In this case, he would profit from the shared knowledge provided by the ontology, but he would still have to write specific code to employ this knowledge in the application.

In *MoRe* we extend the units ontology with an associated reasoning service. For unit conversion, we define an additional but still generic concept `ConversionExpression`. In this case, the application developer only needs to specify a document with the following content (simplified):

```
UnitsOntology
    ConversionExpression
       sourceUnit:      inch
       destinationUnit: yard
```

The appropriate middleware detects the ontology underlying the document, locates it on the Web and applies the associated reasoning service to this input document. After that the middleware sends the following document back to the application:

```
UnitsOntology
    ConversionExpression
       sourceUnit:      inch
       destinationUnit: yard
       factor:          0.02777778
```

The resulting document contains a new fact (value of the `factor` property) allowing the developer to convert inches to yards.

This example provides a simple illustration of the application of our approach. The main motivation behind *MoRe* is to make ontologies more (re-)usable to application developers. Moreover, *MoRe* can help to overcome the lack of expressiveness, efficiency and transparency of present ontology languages (such as OWL) and associated generic reasoners. We emphasize that we do not claim to replace or improve existing formalisms, but rather to provide a pragmatic framework for applying more or less specific algorithms were needed.

In this paper we first introduce the “Unit Converter” scenario in Section 2 in which we outline the main steps a developer takes to employ an ontology in the application at hand. Then in Section 3 we outline the main ideas behind *MoRe* and in Section 4 we apply our approach to the “Unit Converter” scenario. After that, in Section 5 we discuss relationships between *MoRe* and present approaches to ontologies and Semantic Web Services. Also we elaborate on how both the application and the ontology developers are effected by *MoRe* and how they benefit from it. Finally, we conclude with Section 6.

2 The “Unit Converter” Scenario

To demonstrate the effect of *MoRe* on application and ontology developers we employ the “Unit Converter” scenario. In the scenario we consider a task of developing an ontology-based unit conversion application – Unit Converter – that assists a user with conversion between different units of measure.

To develop the Unit Converter, an application developer begins with the application domain analysis. As a part of the analysis, the developer searches for existing ontologies covering the target domain. Let us assume that the developer has found an ontology of units of measure describing the application domain.

The ontology of units of measure can for example be utilized to organize the unit space in a way familiar to the end-user, to select subsets of units that can be converted to each other and, finally, to determine a conversion expression between two given units of measure. In this scenario we elaborate on the last application of the ontology.

Let us assume that the ontology describes a number of units of measure (yard, inch etc) and a conversion factor between a unit and a corresponding reference unit. For example, the ontology states that yard unit has the `SI unit factor` property with value 0.9144 and for inch unit the value is 0.0254. In this example, the `SI unit-` part of the property refers to meter unit (meter is the standard SI-unit for length), so the previous sentence means that 1 yard = 0.9144 meter and 1 inch = 0.0254 meter. An application developer can use the two property values to compute a conversion factor between yard and inch: 1 yard = 0.9144 / 0.0254 inch.

At present we see two main styles of employing ontologies into applications. The first approach is to extract relevant information from an ontology in application-specific form (most often a database) and then employ traditional techniques to access the data and to apply application logic to them. The pseudocode in Fig. 1 demonstrates distinctive features of such an approach. It includes the use of a data query language and computation of the conversion factor in the application.

The main advantage of this approach is that as soon as relevant data is extracted from the ontology, the application developer is able to apply conventional (and well-known) techniques to access the data. The major drawback is that such an approach degrades an ontology to the level of data and makes it difficult to use domain knowledge captured by it.

```
computeConversionFactor (srcUnit, dstUnit, factor)

  srcFactor=db.query(''
          SELECT SI_unit_factor
          FROM ...
          WHERE unit=srcUnit''
      ).get(''SI_unit_factor'')

  dstFactor=db.query(''...WHERE unit=dstUnit'').get(...)

  factor= srcFactor/dstFactor
```

Fig. 1. Pseudocode of a traditional DB-based approach. Using general purpose ontology middelware leads to a similar solution.

```
computeConversionFactor (srcUnit, dstUnit, factor)

  reqDoc=
      ''MyConversionExpression
          type          ConversionExpression
          hasSourceUnit srcUnit
          hasDestUnit   srcUnit''

  resDoc=MoRe.process(reqDoc)

  factor = resDoc.getProperty(''hasConversionFactor'')
```

Fig. 2. Pseudocode of a *MoRe*-based approach.

The second way to exploit an ontology in an application is to employ general purpose ontology middleware, such as Jena [8] or Sesame [9], that provides storage, reasoning and query facilities for ontologies.

However, in our scenario the reasoning capabilities of the associated ontology languages do not allow us to compute the conversion factors in a feasible way (we elaborate on this in Section 4). As a consequence, the second approach would be very similar to the previous one, only the queries would be expressed in a different language and applied not to a database but to a stored ontology.

In both cases, the application developer has to incorporate part of the domain knowledge into the application. Nevertheless, it is natural to expect that the way a conversion factor is computed is part of the units of measure domain. *MoRe* allows an ontology developer to incorporate such domain knowledge as a domain-specific reasoning procedure connected to concepts from the units of measure domain. If an application developer would have such a *MoRe*-ontology to his disposal, the pseudocode depicted in Fig. 2 could be used.

The major difference with the previous cases is that the application developer now reuses domain knowledge about the conversion factor via the reasoning service provided by the *MoRe*-ontology. The second distinction is that the application developer does not need an additional query language to utilize the domain knowledge captured in the ontology. He only needs to refer to an instance of `ConversionExpression` from the extended units ontology. We will elaborate on this in Section 4 and now we introduce the main ideas behind *MoRe* in the following section.

3 *MoRe* in a Nutshell: Documents, Ontologies and Handlers

This section provides a compressed description of the *MoRe* framework. Section 4 fills in missing details and describes how the proposed approach is applied to the introduced scenario.

In *MoRe* we use the notion of `Document` to provide a unified view of Semantic Web resources. All documents share the same structure (a collection of object-property-value triples) and syntax (XML-RDF). A document describes a particular situation in a domain and explicitly refers to exactly one *MoRe*-ontology. This ontology extension defines a reasoning service applicable within that domain.

A *MoRe*-ontology (Fig. 3) extends conventional ontologies by serving as a reasoning service provider. To achieve this, a *MoRe*-ontology exposes exactly one handler providing an entry point to a reasoning service. In *MoRe* we assume that the handler can be invoked via HTTP POST-request with one document as an attachment. The handler processes the attachment and delivers another document as its output. Essentially, a handler is a black box and a *MoRe*-ontology provides the information sufficient for its invocation: the handler's URL. A *MoRe*-ontology can reuse reasoning services provided by other *MoRe*-ontologies. This aspect will not be discussed further in this paper.

Conceptually, in *MoRe* we make use of a subset of RDFS (`Class`, `Property`, `subClassOf`, `subPropertyOf`, `type` and `label`) extended with concepts representing main *MoRe* concepts (`MoReOntology`, `Document`, `Handler`, `URL`) and relationships between them.

Having obtained a document, *MoRe*-middelware is able to identify the extended ontology underlying it and to apply the reasoning service described in this ontology. The outcome of the reasoning service depends on domain-specific knowledge captured in the ontology and the input document that represents a particular situation in the domain. The ontology also provides terminology for input and output documents.

4 Applying *MoRe*

In this section we elaborate on the use scenario presented earlier and describe in detail how *MoRe*-ontologies can be developed to support development of Semantic Web applications. We also show how an application should be designed to utilize extended ontologies.

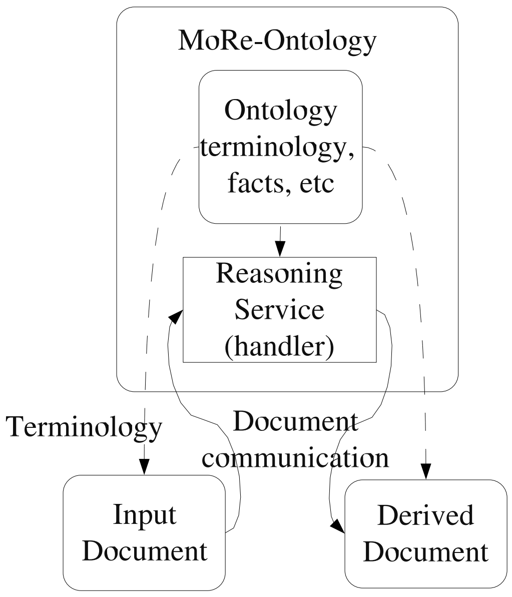


Fig. 3. Relationship between *MoRe*-ontology, document and handler.

4.1 Designing a *MoRe*-Ontology

We now describe the detailed design of a *MoRe*-ontology that extends the ontology of units of measure. In our case we employ UnitDim [2] for this purpose. One of the problems we faced during our initial attempts to exploit the knowledge captured in UnitDim was the difficulty to access domain knowledge expressed in terms of OWL restrictions. Another reason to extend UnitDim into a *MoRe*-ontology is the inability to compute conversion expressions between units using state-of-the-art general purpose reasoners. We have addressed the former problem by creating the UnitRS *MoRe*-ontology extension and the latter has been addressed by the UnitCS ontology extension. Since the general design steps for both UnitRS and UnitCS are similar, in this paper we only elaborate on the UnitCS ontology.

The fact that we develop a *MoRe*-ontology for an application may seem to contradict the idea that ontologies should be application independent. However, we will see that the developed *MoRe*-ontology does not lose any generality but only gains utility in our approach. Moreover, we do not create a *MoRe*-ontology from scratch, but rather extend UnitDim. This demonstrates how an existing ontology can be made more attractive for application development and still preserve its generality.

Unit Conversion (UnitCS) Ontology UnitDim describes a quantitative relation between every unit and a single reference unit. The relation is called `SI_unit_factor`

[2] Rijgersberg, H., Top, J.: UnitDim: an ontology of physical units and quantities. http://www.atoapps.nl/foodinformatics. Sec. News (2004)

and represents a conversion factor between the unit and its counterpart from the SI System of Units. Two such relations can be combined to determine the conversion factor between any two units. In principle, we could have used a general OWL-reasoners to do so. Unfortunately, the OWL language cannot express this domain knowledge in a feasible way. Given a subset of N units, such that any two units can be converted to each other, in OWL we would have to use the complete enumeration of conversion factors. This would result in N^2 property values instead of the more feasible N `SI unit factor` values combined with a capability to infer the rest.

An application can employ the UnitCS ontology to describe documents which then can be subjected to the reasoning service defined in UnitCS. The UnitCS ontology employs the conceptualization defined in two ontologies:

- *MoRe*-Ontology – defines the *MoRe* framework (concepts and their interpretation). All *MoRe*-ontologies reuse this ontology. The handler defined in the *MoRe*-Ontology implements the core of what we will be referring to as *MoRe*-middleware.
- UnitDim – is a conventional domain ontology expressed in OWL. The UnitCS reasoning service exploits domain knowledge captured in UnitDim about relationships between units of measure.

From the ontological point of view, the UnitCS ontology extends UnitDim to infer a conversion expression between convertible units. To achieve this we introduce the `ConversionExpression` class to represent a ternary relationship between two units and a corresponding conversion factor. The `ConversionExpression` class has three properties:

- `hasSource` – points to a source unit, the unit to which we apply the conversion factor;
- `hasDestionation` – points to the destination unit to which the `hasSource`-unit is to be converted to;
- `hasFactor` – contains the numerical value of the conversion factor.

We design the handler for the UnitCS ontology in such a way that for every instance of `Conversion Expression` contained in the input document, the handler determines `SI unit factors` for both the source and destination unit and combines them to compute the corresponding `hasFactor` value. More precisely:

$$Factor_{srcUnit,dstUnit} = factor_{SIUnit,srcUnit}/factor_{SIUnit,dstUnit}.$$

The computed factor allows us to use the following conversion expression

$$srcUnit = Factor_{srcUnit,dstUnit} \cdot dstUnit.$$

Note that the UnitCS ontology could have contained an OWL reasoner instead of our custom-built handler if it could have provided the required functionality. In that case we could see the UnitRS ontology as a wrapper around UnitDim and the standard OWL reasoning mechanism.

An application exploiting the UnitCS ontology needs to have access to the *MoRe*-middleware in order to utilize the reasoning capabilities of the UnitCS ontology. The application uses the terminology defined in UnitCS to create a document describing an instance of the `ConversionExpression` class (Fig. 6). In addition the document contains a URL that points to the UnitCS ontology.

Having created the input document, the application submits it to the *MoRe*-middleware. The middleware analyzes the document, locates its underlying ontology and invokes the handler of that ontology. All inferred facts (statements) are added to the document, which is then returned to the *MoRe*-middleware. The middleware, in turn, returns the result document to the application. The application updates its state according to the newly obtained information.

The above scenario demonstrates how the *MoRe*-framework allows the application developer to abstract from calls to remote procedures. The essential point is that the developer can stay at the conceptual level of domain terminology when requesting external domain knowledge.

4.2 Building the Unit Converter Application

We have employed the UnitRS and UnitCS ontologies in the Unit Converter [3] tool. Figure 5 depicts the architecture of the Unit Converter. In the figure we can see three distinct layers:

- The layer of traditional ontologies is at the top. The UnitDim ontology is the only traditional ontology employed in our application.
- The application layer forms the bottom layer. Inside this layer we can distinguish two sub-layers that represent the application logic and the user interface (UI). The UI sub-layer accepts user's commands and displays the relevant the application state. In our case the user is able to perform three actions: select source (step 1 in Fig. 4) and destination units, and ask for a conversion expression (step 2 in Fig. 4. Two of the actions are connected to the application logic layer in which there are two components responsible for determining 1) a set of convertible units and 2) a conversion expression between two units.
- The application and ontology layers are connected by the *MoRe*-layer. The UnitRS ontology provides a unit retrieval service which makes it easier for an application developer to access domain knowledge captured in UnitDim. UnitCS extends UnitDim with a new concept (`ConversionExpression`) and provides a reasoning service capturing domain knowledge about this concept.
  The application layer interacts with the *MoRe* layer in two ways. First, it employs ontological terminology defined in *MoRe*-ontologies for interfacing purposes (documents). Second, it communicates with *MoRe*-middleware. The middleware is responsible for delivering the input document to the corresponding ontology (its reasoning service) and communicating the output document back to the application layer.

[3] The Unit Converter is accessible via http://www.cs.vu.nl/∼maksym/MoRe/

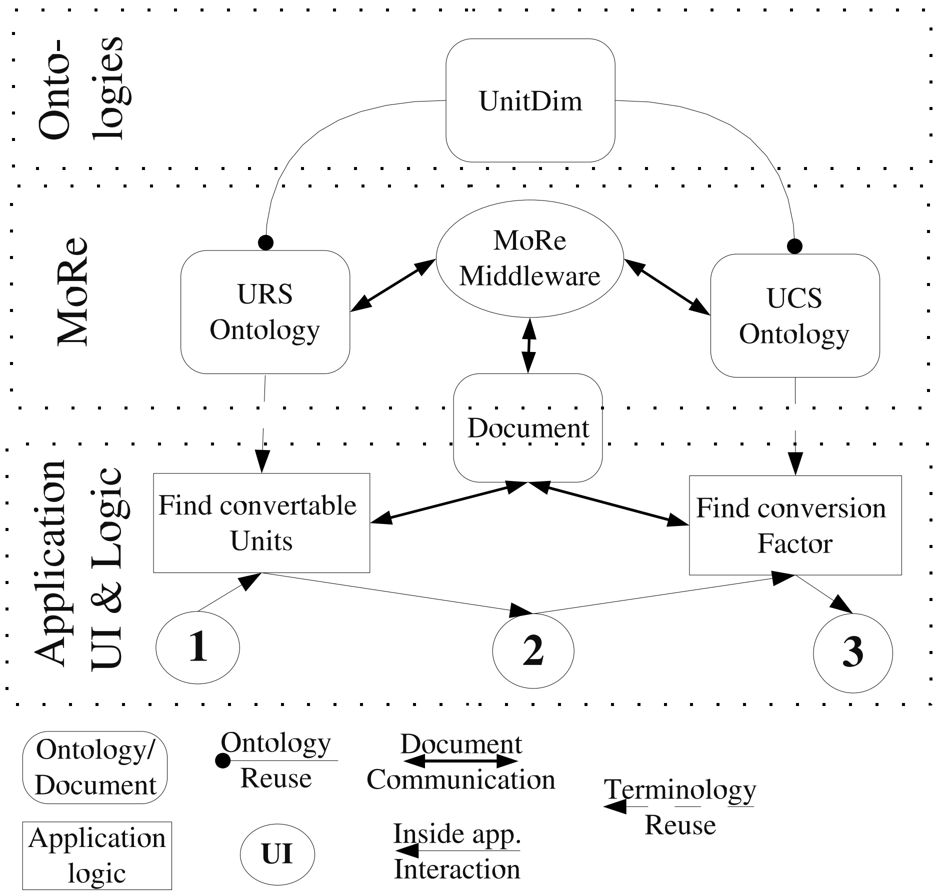
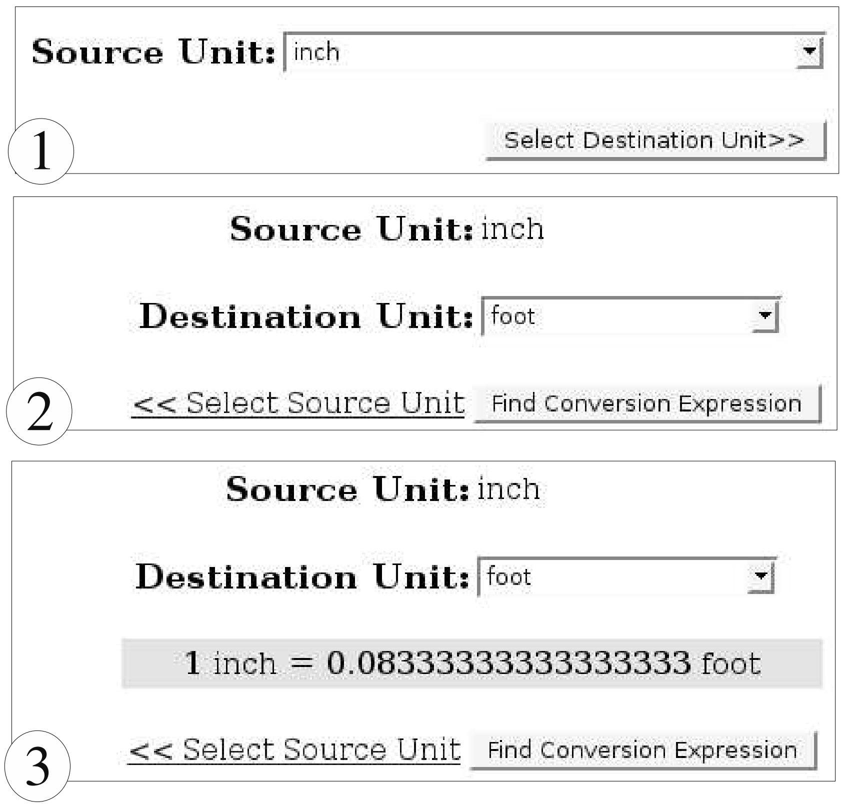


Fig. 4. Main steps of the Unit Converter applicaton



1 inch = 0.08333333333333333 foot

Fig. 5. The architecture and the user interface of the "Unit Converter" application. Numbers on the UI screenshots correspond to the UI components in the application layer.

```
  rdf:Description rdf:about=''ceInst0''
*  hasFactor 0.0277776
   hasSource rdf:resource=''inch''
   rdf:type rdf:resource=''ConversionExpression''
   hasDestination rdf:resource=''yard''
  rdf:Description
```

Fig. 6. A simplified example of a document communicated between the application and *MoRe* layers. Initially the document does not contain a line with "*" which is added by a reasoning service.

Fig. 6 contains a fragment of input and output documents communicated between the "Find Conversion Factor" component of the application layer and the *MoRe* layer. The input document does not contain a line marked with "*". The value of the `hasFactor` property is computed by the UnitCS handler and added to the input document. The initial situation reflects the state of the application after the user has selected the source and destination units. The output document contains a new fact (`hasFactor` property value in our case) which is used to update the application state (step 3 in Fig. 5).

5 Discussion

Despite its brevity, Section 3 describes the main ideas underlying the proposed framework. Nevertheless, we believe that the potential impact of *MoRe* on ontology and application developers can be significant. In this section we discuss the effect of *MoRe* on the current approaches to using ontologies in applications. We also clarify the difference between the proposed approach and the usual view on Semantic Web Services.

5.1 Explicating reasoning mechanisms

The proposed framework does not compete with existing approaches to ontology languages but rather complements them by explicating the link between an ontology and an applicable reasoning mechanism. We believe that this can improve the flexibility of ontologies and make it easier to develop ontology-based applications.

In Section 4 we have explained that state-of-the-art ontology languages such as RDFS and OWL cannot determine a conversion expression between units of measure. We believe that this problem is caused not only by lack of expressiveness of the language but that it is rather a manifestation of the inflexibility of these languages. In *MoRe*, the ontology developer can attach an arbitrary reasoning service to an ontology, ranging from general purpose reasoners to dedicated, goal-specific algorithms. This allows one to handle the limitations of present ontology languages. These limitations become visible when designing real-world applications, either in terms of limited expressiveness or of limits in performance.

MoRe enables development of domain ontologies which are easy to apply. We submit that the success of knowledge-intensive ontologies will be determined primarily by their usability in applications and only secondarily by their definition as a general standard.

5.2 The Black-Box Approach to Capturing Domain Knowledge

In *MoRe* we apply the black-box model to represent a reasoning mechanism. Such a non-declarative approach makes it impossible to reuse parts of the knowledge captured within the black-box. We argue, however, that any declarative language also requires its own black-box to interpret language statements.

In MoRe *we leave it to the user to decide where to draw the border between a declarative and non-declarative representation of domain knowledge*. Usually, it is much easier to initially capture knowledge in a procedural way because it does not restrict the user to a particular declarative representation. Later on, some parts of the procedural knowledge can be exposed in a declarative way. In this way, *MoRe* allows an evolutionary transition from procedural to declarative knowledge representation. Moreover, sometimes it is just impossible to express domain knowledge in a declarative way. For example, a neural network can be trained to organize domain objects into categories, in an inherently non-declarative way. Current approaches do not allow the user to benefit from advances in such non-symbolic areas as evolutionary computing, machine learning and neural networks. We believe that in *MoRe* we enable the user to combine declarative representation techniques with computational (AI) methods and we are going to investigate this ability in our future work.

5.3 Software Components

One more benefit of the black-box approach is that any software component can be represented in this way. This provides us with a link between software engineering and ontologies. *MoRe* makes it possible for software component developers and ontology engineers to combine their efforts to create reusable domain ontologies. In many cases a software component can be relatively easily modified to become part of the Semantic Web. The interface of the component must be reformulated to express input arguments as RDF-XML documents. The terminology employed in the interface becomes part of the component ontology and the logic implemented in the component defines an applicable reasoning mechanism. We believe that *MoRe* will allow us to bridge software engineering and ontological design, improving reusability of the former and flexibility of the latter.

Additionally, *MoRe* allows application developers to abstract from the level of explicit calls to remotely invocable procedures to the ontological level, in which documents are created according to ontologies and reasoning is applied to those document transparently to developers and end-users.

5.4 *MoRe* and the Semantic Web Services

The proposed approach is based upon ontologies and reasoning mechanisms readily-available on the Web. We rely on well-established Web standards such as URI, HTTML, XML and RDF to make the proposed framework operational. This results in a superficial similarity between *MoRe* and the existing approaches to Semantic Web Services such as OWL-S and WSMO. A detailed overview of present approaches to the Semantic Web Services is given in [10]. Here we highlight the major differences between *MoRe* and the Semantic Web Services in general and OWL-S in particular.

MoRe *is a general extension to ontologies.* In *MoRe* we provide the user with a general mechanism to attach a reasoning mechanism to a domain conceptualization. It should not be compared with, for example, OWL-S, which provides a conceptualization of the domain called "Semantic Web Services".

MoRe does not address problems of automatic discovery and composition addressed by other SWS-techniques. Instead *MoRe* provides a simple framework enabling reuse of ontology-based reasoning services. If desirable, additional reasoning services can be plugged in into *MoRe* to address, for example, the automatic discovery task typical for the Semantic Web Services.

MoRe *promotes different usage patterns for ontologies.* In Semantic Web Services, ontologies are used to annotate a Web Service. To use this resource an agent has to understand the annotation, reason about it and, finally, exploit the resource. In *MoRe* an ontology is rather seen as a language that is used to express documents. We believe that the language and the document have a value of their own. An ontology becomes a resource directly exploitable by an agent. *MoRe* enables us to incorporate any computational activity into the reasoning stage transparently to the agent.

At the operational level, OWL-S primarily uses an RPC-style of interaction with the service, focusing on the procedural approach to service specification. This, along with the extensive exposure of internal service details in the process model contrasts with *MoRe*, where we rely on the document-based interaction style with one predefined entry point to the reasoning mechanism. We believe that such an approach is more flexible and offers more opportunities to manage complexity by hiding details of reasoning.

6 Conclusions

We have proposed *MoRe* – a framework for development of ontology-based applications by enabling an explicit link between domain terminology and the appropriate reasoning mechanisms. The proposed framework is based on documents and ontologies containing explicit references to remotely invocable reasoning services (handlers), providing a simple and flexible foundation for development of ontology-based Web applications.

We start with the observation that at some point any ontology requires a directly invocable reasoning mechanism. *MoRe* provides a framework for linking this mechanism, represented as a black-box, to terminology defined in the ontology. This approach not only allows us to incorporate inherently non-declarative reasoning mechanisms, for example based on neural networks, evolutionary computing or any software component into an ontology, but also empowers the user to decide where to draw a border between declarative and procedural representation of domain knowledge.

We believe that *MoRe* helps us to bridge the gap between ontological domain knowledge and application development. On the one hand, it provides a pragmatic application-driven view of extending ontologies. On the other hand, it facilitates application development by enabling easy integration of reasoning services into end-user software solutions. *MoRe* supports an evolutionary development path from existing (legacy) applications to ontology based services.

We are yet to obtain a definitive answer about the practical implications of *MoRe* and a number of technical design decisions. In particular the question on how to combine several ontologies with their respective reasoning services is to be answered. Our next step will be to further validate and refine the method by applying it in the domain of e-Science, in particular in managing scientific knowledge in models and experimental data.

References

1. W3C: Semantic web. (http://www.w3.org/2001/sw/)
2. McBride, B.: Four steps towards the widespread adoption of a semantic web. In: Proceedings of the First International Semantic Web Conference, Sardinia, Italy (2002)
3. Etzioni, O., Gribble, S., Halevy, A., Levy, H., McDowell, L.: An evolutionary approach to the semantic web. In Poster Presentation at the First International Semantic Web Conference (2002)
4. Quan, D., Huynh, D., Karger, D.R.: Haystack: A platform for authoring end user semantic web applications. International Semantic Web Conference (2003) 738–753
5. Popov, B., Kiryakov, A., Kirilov, A., Manov, D., Ognyanoff, D., Goranov, M.: KIM - Semantic Annotation Platform. International Semantic Web Conference (2003) 834–849
6. Dzbor, M., Motta, E., Domingue, J.: Opening up magpie via semantic services. In McIlraith, S.A., Plexousakis, D., van Harmelen, F., eds.: International Semantic Web Conference. Volume 3298 of Lecture Notes in Computer Science., Springer (2004) 635–649
7. Motta, E., Domingue, J., Cabral, L., Gaspari, M.: IRS-II: A Framework and Infrastructure for Semantic Web Services. International Semantic Web Conference (2003) 306–318
8. HP Labs Semantic Web Activity: Jena Semantic Web Toolkit. (http://www.hpl.hp.com/semweb/)
9. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: An architecture for storing and querying rdf data and schema information. In D. Fensel, J. Hendler, H. Lieberman, and W. Wahlster, editors, Semantics for the WWW. MIT Press. (2001)
10. Cabral, L., Domingue, J., Motta, E., Payne, T.R., Hakimpour, F.: Approaches to semantic web services: an overview and comparisons. In Bussler, C., Davies, J., Fensel, D., Studer, R., eds.: ESWS. Volume 3053 of Lecture Notes in Computer Science., Springer (2004) 225–239