Three Dimensions of Knowledge Representation in WonderWeb

Daniel Oberle, Steffen Staab, Raphael Volz

WonderWeb has been an EU IST project funded by the initiative on Future and Emerging Technologies (FET). WonderWeb has been conceived for developing foundations of and infrastructure for the Semantic Web. In this article we discuss WonderWeb results to which the University of Karlsruhe has contributed. Our work considered three different dimensions of knowledge representation in the Semantic Web: first, we have contributed to the W3C recommendation process for OWL, a description logics-based language, and we have investigated its relationship to logic programming; second, the definition of reusable ontologies for common purposes like Web service descriptions in OWL; and, third, the use of such ontologies and reasoning mechanisms to support core Semantic Web infrastructure tasks. Eating our own dog food in the latter case, we have applied Semantic Web techniques inside a general and extensible middleware framework.

1 Representation Languages for the Semantic Web

Representation Languages for the Semantic Web and Semantic Web ontologies encounter the typical trade-off between *expressivity* and *efficiency*. As the Semantic Web targets a broad audience of developers, its languages must in addition fulfill criteria of *usability*.

In the process of standardizing the Web Ontology Language, OWL, the search for expressivity has been an elaborate process of finding consensus on what knowledge representation capabilities might be required most urgently by potential users. Requirements for more expressivity were traded off against results of *efficiency* or *decidability* as known from research in description logics [10]. The resulting language OWL, thus, comes in three increasingly powerful flavors OWL lite, OWL-DL and OWL full [2].

Investigating requirements for large-scale Semantic Web applications, we have in addition felt the urgent need to deal with more efficient, though less expressive subsets of OWL and their efficient implementations — especially with regard to data complexity, which is already NP complete for OWL lite.

Figure 1 illustrates the expressivity of languages (or even language families). Languages from the family of description logics (DL) are strict subsets of First-Order Logic. If chosen carefully like OWL lite and OWL-DL, they are decidable and have an overall complexity of Exptime and Nexptime, respectively.

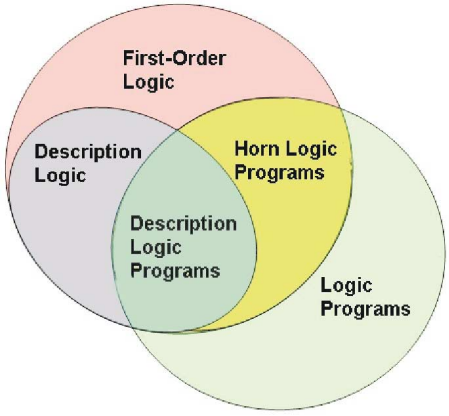

Figure 1: Expressive overlap of DL with LP.

The second major paradigm (that has unfortunately been completely ignored in the OWL standardization process), comes from the tradition of Logic Programming (LP). Though logic programming often uses a syntax comparable to First-Order Logics, it assumes a different interpretation of axioms. Unlike a Tarski-style model theory, logic programming selects only a subset of models to judge semantic entailment of sentences. There are different ways to select subsets of models resulting in different semantics — all of them geared to deal more efficiently with larger sets of data than common approaches based on First-Order Logic. One of the most prominent differences resulting from this different style of logical models is that expressive logic programming axiomatizations become non-monotonic.

An interesting well-known case is horn logic without negation and without function symbols (HL; sometimes referred to by datalog, too) as it is completely contained in First-Order Logic and in the family of Logic Programming approaches and as it has polynomial data efficiency. In order to achieve good data efficiency *and* wide-spread agreement on language primitives, we have investigated the intersection of OWL lite and horn logic/Logic Programming, i.e. Description Logic Programs (DLP). In [4] we show how to perform the bidirectional translation of premises and inferences (including typical kinds of queries) from the DLP fragment of OWL lite to horn logics, and vice versa from the DLP fragment of horn logics to OWL lite. Description Logic Programs inherit the good data efficiency of horn logics. In addition, it is interesting, because it allows for adding new capabili-

ties to improve the *usability* of the DLP language. We have investigated in particular how to

- use rules for querying the ontology and knowledge base,
- define views (comparable to database views) using logical rules [15],
- maintain materializations of queries to improve efficiency [14],
- map between existing relational databases and Semantic Web data [13].

WonderWeb
Ontology Infrastructure for the Semantic Web
`http://wonderweb.semanticweb.org`

WonderWeb has run from January 2002 to December 2004. Four academic partners (the University of Manchester, the Vrije Universiteit Amsterdam, the Laboratory for Applied Ontology, National Research Council, Italy and the University of Karlsruhe) have participated and were advised by a board of about 20 industrial partners. The project was funded by the European Union IST programme (Information Society Technologies) under contract number 2001-33052.

The aim of the project was to develop the infrastructure required for the large-scale deployment of ontologies as the foundation for the Semantic Web. For this purpose, WonderWeb was split in four work packages (WP) with each providing research on a different dimension:

WP1 (language dimension) The development of OWL [2] — the standard ontology web language — as well as proposals for rule extensions maintaining backwards compatibility.

WP2 (runtime infrastructure dimension) the development of a comprehensive technical infrastructure and tool support that will be required by real world applications in the Semantic Web [5, 6, 8].

WP3 (ontology content dimension) The development of a set of foundational ontologies covering a range of application domains. Each provides a carefully crafted taxonomic backbone with a sound high level structure that can be used as the basis for the development of more detailed domain ontologies [7].

WP4 (build and modification infrastructure dimension) The development of a framework of techniques and methodologies that provide an engineering approach to the building and use of ontologies. This included research on ontology change management as well as on the logical foundations of distributed ontologies [9].

2 Reusable Ontologies

Semantic Web applications depend on ontologies and corresponding semantic metadata conforming to these ontologies. While some applications already benefit from low quality of ontologies and corresponding metadata (cf., e.g., [11]), many applications will only generate interest when the problem of high quality ontologies and metadata has been solved to sufficient extent. For the foreseeable future, such high quality may only be provided manually and not by techniques like ontology learning [12]. This is however a rather cumbersome and error-prone work. Hence, it has been our first intent to enable the reuse of existing ontologies in such a way that high quality can be rather easily achieved by just re-using a proven *foundational ontology*. Foundational ontologies are conceptualizations that contain specifications of domain independent concepts and relations. Typically they feature an extensive axiomatization based on principles known from philosophy, linguistics and mathematics. Secondly, a high-quality foundational ontology should be general enough to allow for mediation between many interesting (meta)data sources. Thirdly, a foundational ontology that attracts wide-spread agreement from its community may also lead to a wide-spread understanding of its definitions — and correspondingly to more concise semantic (meta)data.

To explain how we tackle these objectives in a more concrete way, we discuss two use cases that we have drawn from the third dimension of knowledge representation described here (Section 3). In WonderWeb we have considered a particularly relevant and interesting use case that requires high quality ontologies and metadata, i.e. the case of managing distributed systems composed of a middleware system and many subsystems. The concrete middleware we consider are an Application Server and Web services. While we describe the idea underlying these use cases in particular in the following section, in this section we report on the principle of re-usable ontology libraries and their development.

In order to construct ontologies for the two use cases, we have pursued a modularized *and* layered approach that adds ontological commitment in a piece-wise manner for maximum possibility of ontology re-use at all layers. Figure 2 depicts the ontology modules working inside our two use cases. The responsibilities are distributed as follows:

1. DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) [7] is part of the WonderWeb Foundational Ontology Library and has been chosen as the library's basis. Being a foundational ontology, it features a rich axiomatization of domain independent concepts, explicit construction principles, careful reference to interdisciplinary literature and it strives to model human common sense. DOLCE is axiomatized in a modal logic (S5), but it is maintained also in other languages, used according to the particular trade-off between expressivity and computational complexity that is required by a certain application. An OWL-DL version is currently maintained for Semantic Web applications.
   DOLCE is based on the fundamental distinction between enduring and perduring entities. The main relation between *Endurants* (i.e. objects or substances) and *Perdurants* (i.e. events or processes) is that of participation: an endurant "lives" in time by participating in a perdurant. For example, a person, which is an endurant, may participate in a discussion, which

is a perdurant. A person's life is also a perdurant, in which a person participates throughout its duration. DOLCE introduces *Qualities* as another category that can be seen as the basic entities we can perceive or measure: shapes, colors, sizes, sounds, smells, as well as weights, lengths or electrical charges. Spatial and temporal qualities encode the spatio-temporal attributes of objects or events. Finally, *Abstracts* do not have spatial or temporal qualities, and they are not qualities themselves, e.g. (quality) regions or sets. In particular, regions are used to encode the measurement of qualities as conventionalized in some metric or conceptual space.

2. Several additional theories exist for DOLCE that come in the form of ontology modules. Descriptions & Situations (D & S) is such a module and axiomatizes a theory of ontological contexts. It is capable of describing various notions of context or frame of reference (non physical situations, topics, plans, beliefs, etc.) as entities.
   D & S introduces a distinction between *descriptive* and *ground entities*. *Parameters*, *Functional Roles* and *Courses of Events* belong to the first kind and describe the ground entities of DOLCE in the following way: *Parameters* are *valued-by Regions*, *Functional Roles* are *played-by Endurants* and *Courses of Events sequence Perdurants*. A DOLCE physical endurant, e.g. a hammer, could play the functional role of a murder weapon or a tool depending on the context described.
   D & S shows its practical value when applied as an *ontology design pattern* for (re)structuring application ontologies that require contextualization.
3. The descriptions of services show a clear contextual nature, one may only have to consider the number of different views that may exist on a service: the view of a service provider, that of the service requestor or the legal view of a contract etc. The concepts used to formulate any given view are clearly separate from the actual objects they act upon and often independent from the concepts appearing in other views.
   Hence, we have applied both DOLCE and the additional modelling capabilities of D & S to formalize core ontologies for components and for services. While the first axiomatizes typical concepts in an Application Server (most prominently software components and their interrelationships) the second deals with similar aspects of services. Currently, we consider five frequently occurring descriptions of a service, where each represents a separate viewpoint: (Service) Offering, Request, Agreement, Assessment and Norms (more views may be added in the future when needs arise)
4. Domain and application ontologies can finally reuse the core ontologies. The first enrich the core ontologies, which are generic within a domain, by additional domain dependent knowledge. E.g., in the tourism domain, we might specialize a *Service Offering Description* to a *Flight Offering Description* and corresponding axioms, e.g. that a functional role *Operator* is played by an airline.
   Application ontologies are the ones that are finally implemented in a running system. While the ontologies discussed so far are typically heavy-weight, i.e. they feature a rich axiomatization, one has to sacrifice some of the axioms and use a less expressive ontology language that is executable, too[1]. E.g., the domain ontology might be in modal logic S5 inheriting all the axioms of DOLCE. As no decidable reasoning algorithm is known, the developer has to compromise, use the less expressive OWL-DL and restrain the axiomatization accordingly.

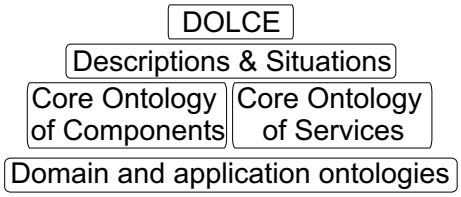

Figure 2: Ontology Library.

The usage of a common foundational ontology also allows to harmonize both ontologies for a simpler translation acknowledging the fact that most services will be exposures of components residing in an Application Server. [3]

3 Semantic Middleware

Besides research on the representation language itself, WonderWeb also addressed its usage in applications. The two use cases we present apply knowledge representation in Application Servers and Service Oriented Architectures, respectively. They both aim at facilitation and improvement of the management and administration of complex distributed applications and systems.

3.1 Application Servers

Application Servers provide many functionalities urgently needed for the development of a complex distributed application. Therefore, Application Servers are nowadays widely used in industrial applications.

Up to now, the functionalities of an Application Server have mostly been developed and managed with the help of administration tools and corresponding configuration files, recently in XML. Though this allows a flexible way of developing and administrating an Application Server with its components, the disadvantage is that the conceptual model underlying the different configurations is *only implicit*. Hence, its bits and pieces are difficult to retrieve, survey, check for validity and maintain.

To remedy such problems, we have taken an ontology-based approach to support the development and administration of software components in an Application Server. The

[1] The typical trade-off between *expressivity* and *efficiency*.

ontology captures properties of, relationships between and behaviors of the components that are required for development and administration purposes. As the ontology is an *explicit* conceptual model with formal logic-based semantics, its descriptions of components may be queried, may foresight required actions, e.g. preloading of indirectly required components, or may be checked to avoid inconsistent system configurations — during development as well as during run time. Thus, the ontology-based approach retains the original flexibility in configuring and running an Application Server, but it adds new capabilities for the developer and user of the system (cf. [6] for details). By using the ontology infrastructure we are able to reason with e.g.

Component Dependencies Libraries often depend on other libraries and a certain archive can contain several libraries at once. Given this information, the ontology infrastructure can assist the developer in locating all the required libraries.

Capability Descriptions Database interfaces typically offer some method to execute an SQL command. However, the behavior of specific database implementations can vary dramatically. Earlier versions of MySQL do not support transactions or subqueries. In this case, component capabilities adhering to standard interfaces can be made explicit to the developer.

Service Classifications Given APIs with similar functionality, one will find different methods and services with essentially the same functionality. A common service ontology will allow the user to discover implementations for a certain ontology entry and to classify a given service.

Access Rights The access control mechanisms of Application Servers are based on users and roles to whom access can be granted for certain resources and services. In addition, services can be run using the credentials of the caller or those of another user that runs the service on behalf of the caller. We are able to assist the user in suggesting suitable settings and in determining potential flaws in the security design.

The proposed scheme resulted in an infrastructure called *Application Server for the Semantic Web* that was also used as the technical infrastructure of the project [5]. It additionally facilitates plug'n'play engineering of ontology-based modules and, thus, the development and maintenance of comprehensive Semantic Web applications. The infrastructure is implemented in a system called *KAON SERVER* which is part of the KArlsruhe ONtology and Semantic Web Toolsuite (KAON, cf. `http://kaon.semanticweb.org`).

3.2 Web Services

Service Oriented Architectures (SOA) allow for factorizing functionality into loosely-coupled and independent services rather than in components like done in Application Servers. The Web-based middleware for such systems is called "Web services" subsuming a set of protocols and XML-languages for invocation, discovery, and interface description of services.

Building a distributed system using a Service Oriented Architecture lets the developer reap advantages roughly similar to the ones found in Application Servers — however it also implies analogous problems.[2] For instance, Web services are described by a multitude of files in (more or less) standardized formats (e.g., WSDL[3] documents for the interface description, BPEL4WS[4] for composition purposes or security[5]). All these descriptions must be provided by the developers of these services, they must be understood by other developers building an integrated application and the individual services must interact tightly in ways that could not be foreseen by their developers nor fully checked by their integrators.

As a still ongoing effort in the remainder of WonderWeb (and beyond), we investigate how to support the development and administration of Web services based systems. For this purpose, we have drafted an ontology that may partially capture relevant aspects of Web services (interface description, etc.) in a harmonizing explicit conceptual model that can be queried or checked for consistency in order to

- ask for Web services that work under certain configurations (e.g. certain conditions of payment);
- find violation of configuration constraints (e.g. a Web service requiring a particular version of application data formatting);
- ascertain Web service policies (e.g. only users that have a valid public key and whose mail-addresses are not contained in a public spam-list are allowed to use the service);
- find security holes (e.g. in a chain of Web service invocations there might be a service that does not use secure protocols);
- ask for depending services (e.g. when the output format of a service is changed).

4 Conclusion

In this project report we have discussed some of the major results of the EU IST project WonderWeb as seen from the Karlsruhe perspective. Results have been achieved in three different dimensions. First, we described the — still ongoing — effort of finding sweet spots of knowledge representation languages for the Semantic Web located in the strings tied between expressivity, efficiency and usability. Second, we have considered efforts towards a harmonized conceptual model for Web services. Third, this harmonized model has been implemented in a Semantic Middleware to support common needs of industry practice when building distributed systems.

[2] Only the problems in Service Oriented Architectures tend to be even worse than in Application Servers, because Service Oriented Architectures target more widely distributed applications.

[3] Web Service Description Language, cf. `http://www.w3.org/TR/wsdl`

[4] Business Process Execution Language for Web Services, cf. `http://www-106.ibm.com/developerworks/library/ws-bpel/`

[5] WS-Security, cf. `http://www-106.ibm.com/developerworks/webservices/library/ws-secure/`

Though WonderWeb is rapidly approaching its end, work on these topics continues in multiple projects that have just been started, e.g. the BMBF project "SmartWeb" or the EU IST project "Adaptive Services Grid" — to name just two in which the first two authors participate, respectively.

Acknowledgements.

We thank all our colleagues in WonderWeb for their fruitful cooperation, in particular Ian Horrocks and Sean Bechhofer (Univ. Manchester), Aldo Gangemi and Nicola Guarino (LAO), Frank van Harmelen, Heiner Stuckenschmidt, Peter Mika, Marta Sabou and Michel Klein (Univ. of Amsterdam), and Rudi Studer (Univ. of Karlsruhe).

References

[1] A. Maedche, B. Motik, and L. Stojanovic. Managing multiple and distributed ontologies in the Semantic Web. *VLDB Journal*, 12(4):286–302, 2003.

[2] D. L. McGuinness and F. van Harmelen. Web Ontology Language (OWL) Overview. `http://www.w3.org/TR/owl-features/`, Feb 2004. W3C Recommendation.

[3] P. Mika, D. Oberle, A. Gangemi, and M. Sabou. Foundations for Service Ontologies: Aligning OWL-S to DOLCE. In *The Thirteenth International World Wide Web Conference Proceedings*, pages 563–572. ACM, May 2004.

[4] B. Grosof, I. Horrocks, R. Volz, and S. Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In *The Twelfth International World Wide Web Conference Proceedings*. ACM, May 2003.

[5] D. Oberle, S. Staab, R. Studer, and R. Volz. Supporting Application Development in the Semantic Web. *ACM Transactions on Internet Technology (TOIT)*, 4(4), Nov 2004.

[6] D. Oberle, A. Eberhart, S. Staab, and R. Volz. Developing and managing software components in an ontology-based application server. In *5th International Middleware Conference*, LNCS. Springer, 2004.

[7] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, and A. Oltramari. Ontology Library (final). *WonderWeb Deliverable D18*, Dec 2003.

[8] D. Oberle, S. Staab, R. Studer, and R. Volz. KAON SERVER Demonstrator. *WonderWeb Deliverable D7*, Dec 2003.

[9] H. Stuckenschmidt, and M. Klein. Ontology Refinement Towards Structure-Based Partitioning of Large Ontologies. *WonderWeb Deliverable D22*, Jun 2004.

[10] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider (eds). *The Description Logic Handbook.*. Cambridge University Press, Jan 2003.

[11] A. Hotho, S. Staab, and G. Stumme. Ontologies Improve Text Document Clustering. In *Proc. of the ICDM 03, The 2003 IEEE International Conference on Data Mining*, 541-544. 2003.

[12] A. Maedche and S. Staab. Ontology Learning. In *Handbook on Ontologies*, 173-190. Springer, 2004.

[13] R. Volz, S. Handschuh, S. Staab. L. Stojanovic and Nenad Stojanovic. Unveiling the hidden bride: Deep Annotation for Mapping and Migrating Legacy Data to the Semantic Web. In *Journal on Web Semantics*, 2004.

[14] R. Volz, S. Staab and B. Motik. Incremental Maintenance of Materialized Ontologies. In *Proc. of CoopIS/DOA/ODBASE 2003*, 707-724, LNCS. Springer, 2003.

[15] R. Volz, D. Oberle, and R. Studer. Views for Lightweight Web Ontologies. In *Proceedings of the 2003 ACM Symposium on Applied Computing (SAC)*, 1168-1173. ACM, 2003.

Kontakt

Daniel Oberle
Universität Karlsruhe
Institut AIFB
76128 Karlsruhe
oberle@aifb.uni-karlsruhe.de
http://www.aifb.uni-karlsruhe.de/

Steffen Staab
Universität Koblenz
Institut für Informatik
56070 Koblenz
http://www.uni-koblenz.de/ifi

Bild

Daniel Oberle studied computer science both at the University of Technology (FH) and the University (TH) of Karlsruhe receiving diploma with majors in distributed systems and data modelling as well as knowledge representation and logics, respectively. He is currently working on his Ph.D. thesis with focus on semantic management of middleware, i.e. the application of semantic technologies in Application Servers and Web services.

Bild

Steffen Staab received his Dr. rer. nat. from the Univ. of Freiburg and his habilitation from the Univ. of Karlsruhe. In 1999, he has co-founded Ontoprise GmbH, the leading company specialized in ontology technology. Recently he has accepted an associate professorship for databases and information systems at the University of Koblenz — founding the research group ISWeb (Information systems and Semantic Web).

Bild

Raphael Volz studied informatics and life sciences at the Universities of Heidelberg and Karlsruhe. Visiting the Information Technologies Research Lab of Swiss Life, Zuerich, he wrote his master thesis on "Acquisition of ontologies using Text-Mining". He received his Dr. rer. pol. from the Univ. of Karlsruhe in 2004 and is now a consultant with Booz, Allen & Hamilton.