

Utility Evaluation of Tools for Collaborative Development and Maintenance of Ontologies

Alexander Norta, Roman Yangarber
Department of Computer Science,
P.O. Box 68 (Gustaf Hällströmin katu 2b),
FI-00014 University of Helsinki, Finland
firstname.lastname@cs.helsinki.fi

Lauri Carlson
Department of Linguistics,
Siltavuorenpenger 20,
I-00014 University of Helsinki, Finland
firstname.lastname@helsinki.fi

Abstract—Ontologies lend themselves for resolving ambiguities in a wide range of applications, including mashups from diverse third-party information sources, and human- and machine-readable specifications of electronic business services (eBS). While tool support exists for the development and maintenance of ontologies, the question remains unanswered what is the degree of utility of these tools in the context of ambiguity resolution, e.g., while discovering eBS. In this paper, we fill the gap by performing an ontology-tool evaluation that allows a comparison of their utility. Based on a carefully selected set of requirements and criteria, we conduct a survey involving leading ontology-tool providers. One of the principal requirements is the collaborative ontology development and maintenance. The paper provides a detailed analysis of survey results.

Keywords—tool evaluation; collaborative ontology development; maintenance

I. INTRODUCTION

Resolving ambiguities in the informal, human-readable description and the formal, machine-readable specification is of primary importance in a wide range of state-of-the-art applications. Our particular focus is electronic business services (eBS), which is relevant for organizations that set up business-to-business (B2B) collaborations in an anonymized, electronic environment. Such B2B-collaboration is complex and it is challenging to ensure that consensus exists about the meaning of terminology between eBS-providers and consumers. For example, an eBS-provider advertises pipes for plumbing while a potential eBS-consumer looks for services about pipes for smoking. In an anonymized, electronic service-brokering environment, ontologies help to resolve such ambiguities.

For re-organizing their B2B-collaboration, many organizations introduce service-oriented architectures (SOA) [17] that Web-enable their legacy systems. Subsequently, these legacy systems become orchestratable and choreographable to support electronic B2B-collaboration. As parts of SOA, notable examples of XML-based industry standards for service orchestration are BPEL [10] and WS-CDL [11] for service choreography. UDDI [5] is instrumental to describe services for facilitating the discovery in broker systems and

service-level agreements (SLA) for the quality of service specification may be expressed in WSLA¹. For ontology specifications, WSDL-S [1], OWL-S [14], RDF [4] are some examples of available standards.

To manage the communication demands during the setup phase between potential eBS consumers and providers, intelligent broker systems should exist in between as middleware. To mention several examples, in [15], a service broker for business grids presents a backward compatible and lightweight approach that uses semantic annotations in service descriptions. A quality of service (QoS) ontology in [18] in combination with a ranking algorithm, can be used in a broker to facilitate automatic and dynamic discovery and selection of eBS. A service-broker concept in [13], bridges the integration gap between telephone companies and the IT world. Depending on the location of a mobile device, an automatic service assignment occurs for mashup creations. We define a mashup as a Web page or application that combines data or functionality from two or more external sources to create a new service. Service brokers function in [12] as an open and distributed environment of geographic information Web services that are searchable with the help of ontology-based metadata. In such broker examples, ontologies are essential for resolving ambiguities in service discovery and with tool support, ontology development and maintenance must be made effective and efficient. Semantic-Web trends in [7], cover application domains, tools, systems, languages and techniques for ontologies. The trends show that many ontology tools exist and an earlier industry-driven study evaluates² them in detail. In [9], a scientific evaluation of methodologies, tools and languages for building ontologies was conducted before the current emergence of service-oriented computing or cloud computing. On the other hand, the ontology-tool evaluation of this paper is based on an ongoing research project, ContentFactory³ (CF) that includes

¹<http://www.research.ibm.com/wsla/>

²XML.com; *Ontology Tools Survey, Revisited*; 2004, <http://www.xml.com/pub/a/2004/07/14/onto.html>

³ContentFactory, funded by Tekes (Finnish Funding Agency for Technology and Innovation), <http://www.verkko-ope.net/cf/>

alos the construction of an ontology-enabled electronic business-service brokerage (eBSB) as part of a cloud-computing ecosystem.

For the purpose of eBS brokering, we assume the development and maintenance of required ontologies for resolving ambiguity issues, requires a collaborative effort of experts from various domains. For example, new terms that enter an ontology may require several terminology experts and a technology expert integrates the resulting term definition in the hierarchy of related ontology concepts and properties. Note, we assume the repositories forming the hierarchy emerge in a bottom-up way that are harmonized into a hierarchy. Thus, a tool for such ontology development and maintenance must offer collaborative functionality. To the best of our knowledge, no survey exists that focuses on the utility of tools for collaboratively developing ontologies for the purpose of brokering eBS. In this paper we fill the gap by carefully extracting focused requirements to perform a targeted evaluation. The evaluation is useful for ontology developers to assess whether a specific tool should be used for certain applications. To do so, the set of requirements and their associated weights may be adjusted in order to quickly establish the respective utility.

The structure of the paper that simultaneously reveals the evaluation method we apply, is as follows. Section II demarcates the domain context that ontology tools must support for this study. Taking into account the application-context features, Section III shows a set of extracted functional and non-functional requirements that ontology-development and maintenance tools should adhere to. The requirement hierarchy has *utility* as root and the leaves are broken down to an operationalized level for the evaluation. Section IV presents the evaluation that starts with a triage of tools, which provide functionality for collaborative ontology development and that are actively maintained. In this paper, we define triage as a categorization of a tool-subset based on the adherence to a criteria for the purpose of more detailed evaluation. Finally, Section V concludes the paper and presents future work.

II. THE TOOL-APPLICATION CONTEXT

For the CF-research project, we develop an ontology-enabled eBSB that can not be fully presented in this paper because of page limitations. Instead, we refer to [16] for further details. Briefly, eBSB allows service consumers and providers to register their service requests and service offers respectively together with assigned personnel responsible for managing the services.

The services exist in the eBSB in a human-readable and also in a complementary machine-readable format like XML-based WS-* languages. In both description representations exist terminology that is potentially ambiguous for parties who want to perform B2B-collaboration. A certain application context for the eBSB exists, e.g., a specific

supply chain around an original equipment manufacturer, geographic region, service domain, and so on, triggers the establishment of one or several dedicated ontology libraries for resolving such ambiguities. A description of specific ontology-library features follows in the sequel.

The eBSB is an anonymized, electronic platform and it is a challenge for users to estimate the trustworthiness of service offers and requests, their issuing organizations and service-assigned personnel. Hence, the eBSB automatically pulls in related third-party information from blogs, wikis, news, specific company registries in the Web, and so on, for a mashup to allow a targeted and speedy estimation of trustworthiness.

The pulled in third-party information from the Web for the mashup may contain a lot of irrelevant content. The earlier established ontology library for the collaboration context serves for an automated ontological pruning, ranking and aggregation of information in the mashup. Consequently, the eBSB-user decides to either provide for a service request or consume a service offer.

A. Specific Features of Ontology Management

Based on a lifecycle of ontology development and maintenance in Figure 1, we explain the specific features of ontology management in the framework of the eBSB. The lifecycle starts with a user who searches for service offers/requests. The human-readable service description shows automatically extracted terminology that is the basis for dynamically setting up an ontology pyramid. The machine-readable service specification may be another source of such terminology, e.g., properties of message specifications.

The first step to choose the root for the ontology pyramid that contains the highest-level concepts for the service context. Next, ontology libraries with refining concepts and properties form the full ontology pyramid. Attached meta ontologies visualized as sheet icons, allow a classification of the ontology root and refining libraries for speedy discovery. Note that the arcs of Figure 1, indicate it is possible to first establish in a bottom-up way the ontology libraries locally that trigger later the introduction of a pyramid root to cater for ontology-engineering flexibility.

After assembling the ontology pyramid, the eBSB is able to resolve ambiguities in service descriptions that contain specific keywords, i.e., for keywords in the service description ontological concepts and properties serve as clarification. The completeness check means the individual keywords must be present in the existing libraries. If all keywords are already part of ontology libraries, the pyramid is ready and requires no further changes. Otherwise, terminologists engage in a discussion for establishing a term definition that is translatable into an ontology update.

Based on the the application context, we deduce the a set of requirements for the ontology-tool evaluation.

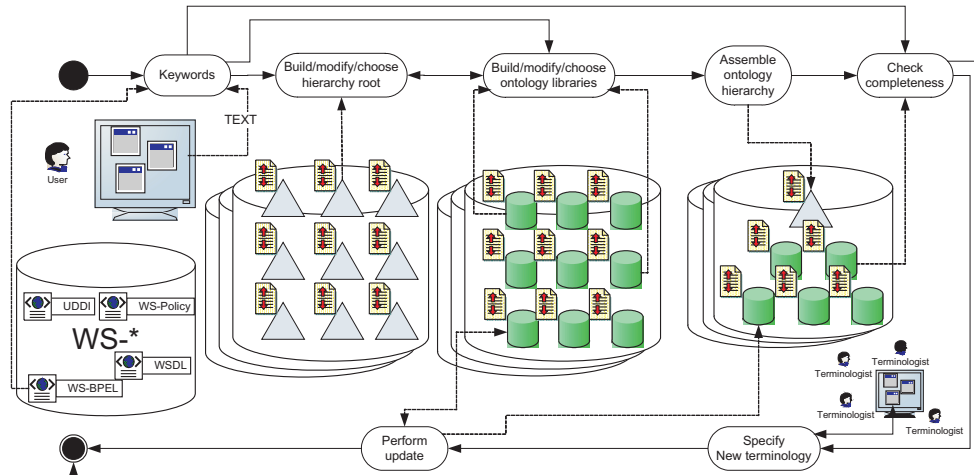


Figure 1. Lifecycle of ontology development and maintenance.

III. DEDUCED REQUIREMENTS FOR ONTOLOGY-TOOL EVALUATIONS

The ontology-tool evaluation, considers an exhaustive set of functional and non-functional requirements extracted from the ContentFactory-project needs [16] and also from [3] software-engineering literature. The first cover desirable behaviors of ontology tools while the latter specify criteria to judge the operation of an ontology tool, rather than specific behaviors. Other terms for non-functional requirements are quality attributes and quality of service requirements.

The depiction of the evaluation-requirements hierarchy in Figure 2 shows *utility* as the root. The leaf requirements have annotations with categorizers of which *r* stands for triage, *m* for must have, *n* for need to have and *i* for nice to have. The depiction shows with an arrow that *performance* references to *verification and evaluation*, i.e., functionality for correctness-checking ontologies, should not take longer than several seconds to produce results. Below, we explain the requirements. Their specific definitions for the context of this evaluation represent a tradeoff with the objective allowing an evaluation of all tools. Hence, too strict requirement definitions may result in a bias for one tool and disadvantage others.

A. Functional Requirements Specification

For the set of functional requirements below, it is important to understand how ontology tools play a role in clarifying ambiguities in that lifecycle. Thus, the ontology tool must support the development of ontology hierarchies with a top level either entirely or partly versioned and attached lower-level ontologies. The libraries of the ontology hierarchy may be formulated in differing XML-based specification languages, e.g., OWL variations, RDF, and linked in ontologies either from local or external

libraries. Terminology definitions created by deliberations of terminology experts may span several human languages and form the basis for building ontology libraries collaboratively.

a. Collaborative ontology development: The multi human-language terminology is part of a collaboratively refined ontology hierarchy and created with a consensus by domain experts. Hence, proposal and voting functionality must be provided for users that slip into specific roles with competencies and permissions, e.g., permitting the initiation of terminology voting. Out of this requirement scope are automated voting initiation, automated ontology creation and specific workflows for ontology development and maintenance.

b. Maintaining collaboratively refined hierarchies of ontologies: It is important to change on the fly the libraries that make up the ontology hierarchy, i.e., refined subsets of the ontology hierarchy must be exchangeable with alternative ontology refinements.

c. Ontology development in different human languages: In the CF project, we consider ontologies for terminologies represented in human languages different than English, e.g., French, Chinese, and so on. Hence, the tool must support the development and maintenance of ontologies in other languages than just English.

d. Multi ontology-language support: Although ontology libraries may be formulated in different machine-readable languages, integrate the libraries into the same ontology hierarchy needs to be possible. The languages we consider are OWL-DL, OWL-FULL, OWL-LITE, RDF. The CF-project focus for ontologies is OWL-DL as it allows representations in 1st order logics together with restrictions.

e. Verification support: The ontology hierarchy needs to be verifiable in its partial libraries and also in its entirety. Verification properties are, i.e., the syntactic correctness for

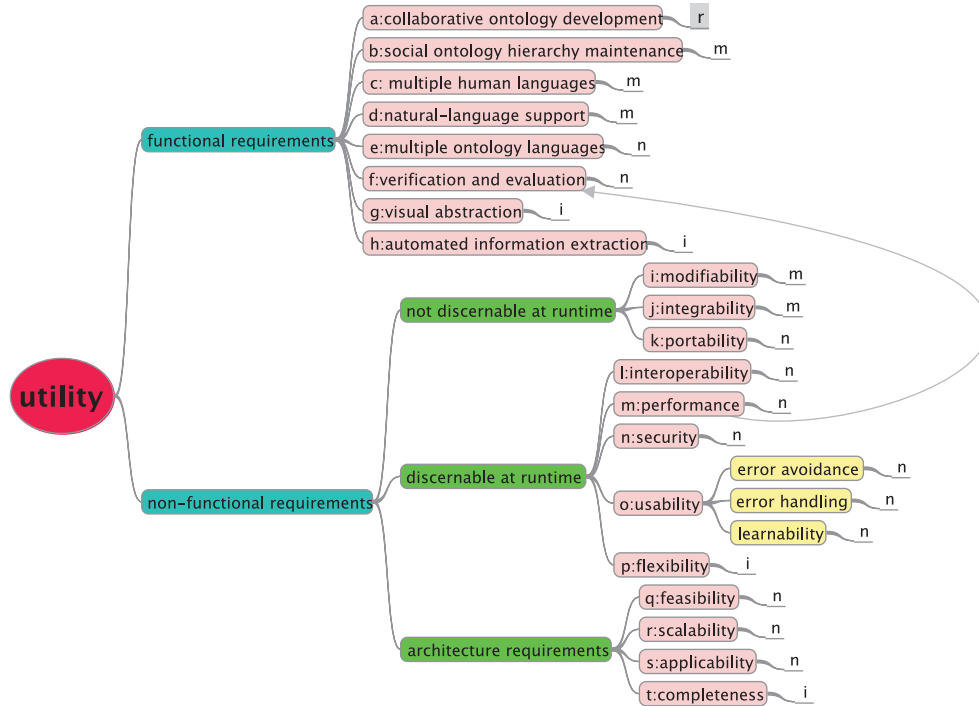


Figure 2. Hierarchy of evaluation requirements.

the description logics of, for example, an OWL instantiation; and ontological consistency in which an ontology is extended to a canonical model using tableau algorithms [2]. An example for consistency violation would be somebody in an ontology who is married multiple times concurrently. Note that we do not narrow down to one particular formalism. Instead, the ontology tool needs to demonstrate adjustment flexibility towards verification based on arbitrary formalisms.

f. Visual abstractions for facilitating ontology development and maintenance: It is nice to provide intuitively comprehensible visual abstractions for ontologies that also allow laymen to engage in development and maintenance. Given the nature of ontologies, a class-hierarchy view as a tree is a minimum requirement. Additionally, it is nice to have graph representations with separate visibility options, e.g., zooming, cropping, filtering, choosing specific properties. The graph view should be editable.

g. Natural-language support: Terminology in ontology libraries may comprise synonyms or representations in different human languages. Hence, it is nice for users to support the management of such term similarities, lexical referencing of ontology elements, searching, filtering, and so on.

h. Automated information extraction: The tool should be able to automatically extract information from unstructured text that is useful for the development and maintenance of ontology libraries. Hence, functionality for such knowledge acquisition is nice to have available.

B. Non-Functional Requirements Specification

For the ontology-tool evaluation, the division of non-functional requirements distinguishes from [3] between requirements that are observable via execution and those that are not. In the first case, the requirements are performance, security, availability, functionality, and usability. In the latter case, the requirements are modifiability, integrability, portability. Finally, we consider requirements for the ontology-tool architecture, namely feasibility, scalability, applicability, completeness. First, we specify ontology-tool requirements that are not *discernible at runtime*.

i. Modifiability: Ontology tools must be able to adopt new functionality quickly to remain relevant for rapidly changing application domains. The ontology tool must allow for rapid internal functionality adjustment triggered by changes that occur in the real-world application context.

j. Integrability: For an ontology tool, it must be possible to also comprise components that may be from third parties separately developed and integrated at a later stage.

k. Portability: Ontology tools need to run on heterogeneous system infrastructure, i.e., hardware, software, or a combination of both. Hence, the architecture level encapsulates platform-specific considerations that enables portability by giving the application software an abstract interface to its environment.

Next, we specify the system requirements for ontology tools

discernible during runtime.

l. Interoperability: The ontology tool must be loosely coupled with a collection of services that are part of a larger service-oriented application architecture. Hence, the ontology tool must be Web-addressable and use XML-based languages for exchanging information with other loosely coupled services.

m. Performance: The computational and communicational load in collaborative ontology development and maintenance needs to be manageable by tools within acceptable time phases. Service discovery in the context of emerging SOC, is a constantly changing domain. Hence, the execution of any tool functionality always needs to remain below a minute. That time limitation includes specifically ontology verification.

n. Security: Refers to the ability of resisting unauthorized attempts of usage of ontology libraries while only permitting authorized ones. For example, an authentication server needs to reside between collaborating parties. Monitors need to be used for inspecting and logging network events. The communication of a system needs to be placed behind a firewall, and so on.

o. Usability: Refers to a tool being easy to use for collaboratively developing and maintaining multi-lingual ontologies in differing languages. We split usability into the following three requirements [6] that are relevant for ontology tools. *Error avoidance* means that functionality needs to be in place for preventing and anticipating common errors that occur during ontology development and maintenance. Closely related is the issue of *error handling*, which is satisfied when a tool helps a user to recover from errors. *Learnability* refers to how quickly users can learn using the tool.

p. Flexibility: It is nice if ontology tools are so versatile that they support the execution of diverse activities, the participation of diverse partners, and the exchange of diverse data. Thus, this requirement addresses the unpredictable conceptual and technological heterogeneity of the ontology-tool application context.

Additionally, we consider *requirements on architecture*. The requirement of

q. feasibility means that it needs to be possible with a tool to collaboratively develop and maintain ontologies within acceptable time and cost limits, i.e., costs for hardware, time to learn the tool, and so on. **r. Scalability** refers to the ability of a tool to combine more than two collaborating parties into ontology development and maintenance. **s. Applicability** states that as described in Figure 1, a tool is instrumental for guiding the collaborative development and maintenance of ontologies. **t. Completeness** means an ontology tool needs to comprise the components required in accordance with Section III-A for satisfactorily developing and maintaining ontologies collaboratively.

IV. EVALUATION OF ONTOLOGY-TOOLS

Below, we first show how the utility of a tool is computed, followed by a description of the tools that remained after the initial triage check. Next, tables show functional and non-functional evaluation results. We stress that the utility equation suits the needs of the CF-project. In other application contexts, the utility computing may use different sets of requirements with their own requirement definitions and different weights.

$$u^t = \theta_T T^t + \theta_M \sum_i M_i^t + \theta_N \sum_j N_j^t + \theta_C \sum_k C_k^t \quad (1)$$

For computing the utility u of a tool t , we consider Equation 1 in which the classes of requirements T, M, N and C (triage, “must”, “need” and “nice”, respectively) receive scores on the scale $\{3 \mid 2 \mid 1 \mid 0\}$. I.e., 3 means an ontology tool has a strong focus on a requirement, 2 means there largely exists support, 1 means a requirement is somewhat supported and 0 means a requirement is not applicable for a respective tool evaluation. Note that after the triage we only consider tools for the evaluation that do support collaboration, for which free versions of their tools are in maintenance. We do not consider tools that are purely commercial and can therefore not be freely used in publicly funded research projects. Consequently, all other tools are not considered in this paper.

The respective weights (θ 's) for the requirements—the triage T , the must requirement M , need N , and nice C —can be set in many ways, our main concern being that $\theta_T > \theta_M > \theta_N > \theta_C$. Therefore, in our evaluation, we fixed them at 4, 3, 2 and 1, respectively. The overall utility of a tool results from summing up all values.

A. Tool Triage Results

For the detailed evaluation, the triage result presented below comprise actively maintained, freely downloadable tools with collaborative ontology development capabilities

- NeOn: The NeOn toolkit⁴ is a state-of-the-art, open-source, multi-platform ontology-engineering environment that aims to provide comprehensive support for all activities in the ontology engineering life-cycle. The toolkit is based on the Eclipse platform and provides an extensive set of plug-ins covering all aspects of ontology engineering, including relational database integration, modularization, visualization, alignment, and project management.
- Protégé: As a free, open-source ontology editor and knowledge-based framework, Protégé⁵ allows the collaborative development of ontologies. Protégé ontologies can be exported into a variety of formats including RDF(S), OWL, and XML Schema.

⁴<http://neon-toolkit.org/>

⁵<http://protege.stanford.edu/>

- CmapTools Ontology Editor: Concept maps [8] are an effective way of representing the understanding of a person’s domain of knowledge. CmapTools⁶ is a software environment developed at the Institute for Human and Machine Cognition (IHMC) that empowers users, individually or collaboratively, to represent their knowledge using concept maps, to share them with peers and colleagues, and to publish them. Additional CmapTool extensions support roundtrip translation for ontology⁷ representations.
- TopBraid Composer: For developing Semantic Web ontologies and building semantic applications, TopBraid Composer⁸ is an enterprise-class modeling environment. Fully compliant with W3C standards, TopBraid Composer offers comprehensive support for developing, managing and testing configurations of knowledge models and their instance knowledge bases. TopBraid Composer incorporates a flexible and extensible framework with a published API for developing semantic client/server or browser-based solutions that can integrate disparate applications and data sources. Note, for the evaluation we consider the free edition with only a subset of functionality that the commercial editions offer.
- HOZO: The development environment, named HOZO⁹ for building ontologies comprises of Ontology Editor, Onto-Studio and Ontology Server. Ontology Editor provides users with a graphical interface, through which they can browse and modify ontologies. This system manages properties between concepts in the is-a hierarchy. Onto-Studio helps users design ontologies from technical documents. Ontology Server manages the built ontologies and models.
- OntoBroker: The OntoBroker¹⁰ is a comprehensive and scalable semantic Web-middleware. It is an inference machine for the processing of ontologies that supports all W3C Semantic Web recommendations: OWL, RDF, RDFS, SPARQL and, in addition, the industry standard F-Logic. Note that OntoBroker serves as a foundation for the development of NeOn and thus, we omit this tool in the detailed evaluation as covers it.

In the complementary survey document¹¹ for this evaluation, two tables list all the ontology tools with collaborative capabilities we considered for the triage. The last column of these tables gives reasons for not adopting a particular tool.

⁶<http://www.ihmc.us>

⁷<http://coe.ihmc.us/groups/coe/>

⁸http://www.topquadrant.com/products/TB_Composer.html

⁹<http://www.hozo.jp/>

¹⁰<http://www.ontoprise.de/>

¹¹<http://www.cs.helsinki.fi/u/anorta/publications/OntologyToolEvaluationStatements.pdf>

B. Functional Requirements Evaluation

The evaluation results stem from a survey we requested tool representatives to fill in. Note, grey colored cells with numbers in Table I and Table II indicate we performed the evaluation ourselves to the best of our knowledge. While in both cases of our own evaluation, we integrate feedback from ontology-tool representatives, it is important to stress that the results in grey colored cells and the justification remarks contained in the complementary survey document (referenced in the footnote) represent our own evaluation experience from working with the tools and also studying available white papers.

After the first set of evaluation results were generated, tool-representatives cross-checked each other’s evaluation scores. We hope to have succeeded in integrating all cross-checking inputs in the final tables depicted below. With respect to functional requirements, the evaluation results in Table I indicate that the best scoring tools are NeOn and Protege, followed by CmapTools Ontology Editor in third position. The remaining tools also pass the triage (*a*) requirement.

Requirements	Ontology Tools					Weight	
	NeOn	Protege	CmapTools Ontology	TopBraid Composer	HOZO		
functional requirements	a	3	3	3	3	1	4
	b	3	1	3	3	1	3
	c	3	3	3	3	3	3
	d	2	3	2	2	1	3
	e	3	3	1	1	2	3
	f	3	3	3	1	3	2
	g	3	3	1	1	3	2
	h	3	2	0	0	0	1
utility f	60	55	47	43	37		

Table I
HIERARCHY OF EVALUATION REQUIREMENTS.

In Table I, NeOn scores reveal a strong focus in all functional requirements with the exception of multi-ontology language support. The given reason is the focus on OWL2 that results in a slightly less complete coverage of earlier ontology languages.

Protégé, does not fully cover three functional requirements completely. Firstly, there is no explicit support for maintaining collaboratively refined hierarchies of ontologies. Protégé also does not fully support natural-language and automated information-extraction functionality. However, differently to NeOn, there is full coverage of multi-ontology language support.

CmapTools Ontology Editor (COE) fully supports the collaborative development of ontology hierarchies that involve different human languages. With the use of concept maps for the user interface in ontology development, COE is a suitable option for laymen who are not technical ontology experts. COE does not support with a strong focus multi-ontology language support, verification and natural-language

support. There is no support at all for automated information extraction.

TopBraid Composer supports the triage and all functional must requirements fully with the exception of multi ontology-language support. Full support requires additional open-source components. However, the tool somewhat supports verification support, visual abstractions, and natural-language support while automated information extraction is not applicable.

With respect to the triage requirement, HOZO satisfies it somewhat while the tool only fully supports ontology development in different human languages out of the must requirements. Further strengths of HOZO are visual abstractions and natural-language support.

C. Non-Functional Requirements Evaluation

The second set of evaluation results for non-functional requirements depicted in Table II, also sums up the summed up evaluation results at the bottom of the table. NeOn scores well in Table II but not perfectly in all requirements. According to the tool developers, NeOn pays attention to convenient plugin management but does not have a Web version of its tool. For the same requirement, Protégé does not provide interoperability support, no Web version exists of the tool and no information is given to which extent the architecture supports loose coupling. COE, on the other hand, provides access to and integration with a larger SOA architecture.

Requirements		Ontology Tools					Weight	
		NeOn	Protege	CmapTools Ontology	TopBraid Composer	HOZO		
non functional	runtime	i	3	3	2	3	2	3
		j	3	3	2	3	1	3
		k	3	3	3	3	1	2
	non runtime	l	2	1	1	0	1	2
		m	3	3	3	1	2	2
		n	1	3	1	1	2	2
		o	3	3	3	2	2	2
		p	2	2	0	2	1	1
	architecture	q	3	3	3	3	2	2
		r	3	3	3	3	2	2
		s	3	2	3	1	1	2
		t	2	3	3	2	1	1
	utility nf		64	65	55	50	37	
	utility Σ		124	120	102	93	74	

Table II
HIERARCHY OF EVALUATION REQUIREMENTS.

NeOn and COE provide partial security support while Protégé has a strong security focus. The flexibility requirement is not fully supported by NeOn although the tool offers some update notification for new plugins. Protégé relies on being open-source and covers flexibility with third-party components. Finally, NeOn with its plugin architecture offers essential components by default and counts on additional plugin components that allows the tool to evolve for specific application contexts. Based on the longer existence of the tool, Protégé and COE claim an empirically

proven strong focus for completeness based on its record of project applications. For the remaining evaluation results, we refer readers to the justification comments¹² in the survey document.

The evaluation shows the free edition of TopBraid Composer scores well in the runtime section of non-functional requirements and covers the architectural requirements with the exception of applicability that is somewhat satisfied. For the free edition, weakest scores are in the block for non-runtime requirements. HOZO, on the other hand, scores in all non-functional requirements, although it has nowhere a strong focus. Just as COE, HOZO does not have a strong focus on any non-functional must requirements that matter for the CF-project context.

V. CONCLUSION AND FUTURE WORK

This paper presents a tool evaluation for the development and maintenance of ontology hierarchies that may involve different human and ontological languages. The evaluation context is an evolving system for business-service brokerage that uses ontologies to resolve ambiguities in the human- and machine-readable specifications of electronic business services. On the other hand, such ambiguities need to be resolved for service-related information that is pulled in from the Web cloud and displayed as a mashup so that users develop trust in their service choices. Based on the application context, a hierarchy of functional and non-functional evaluation requirements refines the utility notion for ontology tools.

A triage reduces the set of investigated ontology tools based on their ability to develop and maintain hierarchies of ontologies collaboratively. These tools must be available for evaluation and still actively maintained by an existing organization. The result of this triage leaves over the tools NeOn, Protégé, CmapTools Ontology Editor, OntoBroker, HOZO, the free version of TopBraid Composer. Since OntoBroker is the foundation for NeOn, the first tool may be omitted for the detailed evaluation. Tables for the functional and non-functional requirements show the evaluation results and the overall utility.

Representatives evaluate their own tools NeOn, CmapTools Ontology Editor and HOZO. The evaluation of Protégé and TopBraid Composer was carried out by the paper authors and corrected based on feedback from a representative. Evaluating the tools by representatives may be considered a weakness of the evaluation. However, the final results are cross-checked by all representatives from whom tools are part of the detailed evaluation.

With respect to overall utility, the study suggests that NeOn is slightly ahead of Protégé and CmapTools Ontology Editor. Note that all tools in this detailed evaluation are

¹²<http://www.cs.helsinki.fi/u/anorta/publications/OntologyToolEvaluationStatements.pdf>

excellent in their own right and the ranking of tools emerges from the specific needs of the application context. The ranking may change in other application contexts that lead to different requirement definitions and weights. All tools share OWL-DL as a "lingua franca" and can be used in combination for the development and maintenance of ontologies. That way the respective strengths and weaknesses of ontology tools complement each other. For example, in the context of the ContentFactory project, CmapTool Ontology Editor allows laymen without ontology knowledge to take advantage of the intuitive user interface that applies concept maps. When translated to OWL-DL, NeOn proposes itself as the ontology tool with the highest degree of utility. However, the evaluation suggests that Protégé is a better choice for ontology work that must pay attention to security. HOZO offers multi-perspective ontology visualizations and Asian language support. TopBraid Composer offers a limited subset of functionality of its free edition that we considered for the detailed evaluation, compared to the pay-editions. The latter edition offers higher utility with more functionality.

In the future, we will study how to take advantage of the pre-existing tool functionality so that we minimize new implementation efforts in the ContentFactory project. Based on ongoing experiences from ContentFactory system refinements, functional and non-functional gaps in existing ontology tools become apparent during their application. That way we hope to provide inspiration to the ontology-tool developers for initiating research and development projects for future extensions.

ACKNOWLEDGMENT

This research is conducted in the ContentFactory research project and funded by the Finnish Funding Agency for Technology and Innovation (Tekes). We thank Martin Dzbor from NeOn, Tom Eskridge from COE, Kouji Kozaki from HOZO, for filling in the survey and discussing the results. We thank TopBraid Composer and Protégé for providing valuable feedback to our evaluation.

REFERENCES

- [1] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M. Schmidt, A. Sheth, and K. Verma. *Web Service Semantics - WSDL-S*. <http://www.w3.org/Submission/WSDL-S/>, 2005.
- [2] F. Baader, I. Horrocks, and U. Sattler, editors. *Chapter 3 Description Logics*. In *Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, Handbook of Knowledge Representation*. Elsevier, 2007.
- [3] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley, 1998.
- [4] D. Beckett and B. McBride. *RDF/XML Syntax Specification (Revised)*. <http://www.w3.org/TR/rdf-syntax-grammar/>, 2004.
- [5] T. Bellwood, L. Clment, and D. Ehnebuske et al. *UDDI Version 3.0, Published Specification*. <http://uddi.org/pubs/uddi-v3.00-published-20020719.htm>, 2003.
- [6] P.O. Bengtsson. *Architecture-Level Modifiability Analysis*. PhD thesis, Department of Software Engineering and Computer Science, Blekinge Institute of Technology, Sweden, 2002.
- [7] Jorge Cardoso. The semantic web vision: Where are we? *IEEE Intelligent Systems*, 22(5):84–88, 2007.
- [8] A.J. Cañas, G. Hill, R. Carff, N. Suri, J. Lott, T. Eskridge, G. Gómez, M. Arroyo, and R. Carvajal. Constructing process views for service outsourcing. In A.J. Cañas, J.D. Novak, and F.M. González, editors, *In: Concept Maps: Theory, Methodology, Technology, Proceedings of the First International Conference on Concept Mapping*, pages 125–133. Universidad Pública de Navarra: Pamplona, Spain, 2004.
- [9] Oscar Corcho, Mariano Fernández-López, and Asunción Gómez-Pérez. Methodologies, tools and languages for building ontologies: where is their meeting point? *Data Knowl. Eng.*, 46(1):41–64, 2003.
- [10] D. Jordan, J. Evdemon, A. Alves, and A. Arkin. *Business Process Execution Language for Web-Services 2.0*. <http://www.oasis-open.org/committees/download.php/10347/wsbpel-specification-draft-120204.htm>, 2007.
- [11] D. Jordan, J. Evdemon, A. Alves, and A. Arkin. *Web Services Choreography Description Language 1.0*. <http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/>, 2007.
- [12] E. Klien, M. Lutz, and W. Kuhn. Ontology-based discovery of geographic information services? an application in disaster management. *Computers, Environment and Urban Systems*, 30(1).
- [13] Salvatore Loreto, Tomas Mecklin, Miljenko Opsenica, and Heidi Maria Rissanen. Service broker architecture: location business case and mashups. *Communications Magazine*, 47(4):97–103, 2009.
- [14] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, and S. McIlraith. *OWL-S: Semantic Markup for Web Services*. <http://www.w3.org/Submission/OWL-S/>, 2004.
- [15] Henar Muñoz Frutos. Towards a semantic service broker for business grid. In *ESWC 2009 Heraklion: Proceedings of the 6th European Semantic Web Conference on The Semantic Web*, pages 939–943, Berlin, Heidelberg, 2009. Springer-Verlag.
- [16] A. Norta. A HUB ARCHITECTURE FOR SERVICE ECOSYSTEMS: Towards Business-to-Business Automation with an Ontology-Enabled Collaboration Platform. In *Proc. of 6. International Conference on Web Information Systems and Technology (WEBIST) 2010*, volume 2, pages 240–243. INSTICC, 2010.
- [17] M.P. Papazoglou and P.M.A Ribbers. *e-Business: organizational and technical foundations*. John Wiley & Sons, Ltd., 2006.
- [18] Vuong Xuan Tran, Hidekazu Tsuji, and Ryosuke Masuda. A new QoS ontology and its QoS-based ranking algorithm for Web services. *Simulation Modelling Practice and Theory*, 17(8):1378–1398, 2009.