Ontology Based Online Knowledge

K.Vanathi[1], D.Muthusankar[2]
[1]PG Scholar, K.S.Rangasamy College of Technology, Tiruchengode, India.
Email: vanathi.vanu@gmail.com, Mobile No: +91 9003831771.
[2]Assistant professor (Academic), K S Rangasamy College of Technology, Tiruchengode, India.
Email: muthusankar@ksrct.ac.in

Abstract-- **A key goal of the Semantic Web is to shift social interaction patterns from a producer-centric paradigm to a consumer-centric one. Treating customers as the most valuable assets and making the business models work better for them are at the core of building successful consumer-centric business models. It follows that customizing business processes constitutes a major concern in the realm of a knowledge-pull-based human semantic Web. This work conceptualizes the customization of service-based business processes leveraging the existing knowledge of Web services and business processes. It is represented with this the conceptualization as a new Extensible Markup Language (XML) markup language Web Ontology Language-Business Process Customization, based on the de facto semantic markup language for Web-based information Web Ontology 2 DL Language (OWL 2DL)]. Furthermore, it is reported with a framework, built on OWL-BPC, for customizing service-based business processes, which supports customization detection and enactment. Customization detection is enabled by a business-goal analysis, and customization enactment is enabled via event–condition–action rule inference. Our solution and framework have the following capabilities in dealing with inconsistencies and misalignments in business process interactions: 1) resolve semantic mismatch of process parameters; 2) handle behavioral mismatches which may or may not be compatible; and 3) process misaligned rendezvous requirements. Such capabilities are applicable to business processes with heterogeneous domain ontology. We present an architectural description of the implementation and a walk-through of an example of solving a customization problem as a validation of the proposed approach.**

Keywords— **Information system, ontology, Web Ontology Language, OWL 2DL, XML.**

I. Introduction

An ontology formally represents knowledge as a set of concepts within a domain, and the relationships among those concepts. It can be used to reason about the entities within that domain and may be used to describe the domain.

In theory, an ontology is a "formal, explicit specification of a shared conceptualisation". An ontology renders shared vocabulary and taxonomy which models a domain with the definition of objects and / or concepts and their properties and relations.

Ontologies are the structural frameworks for organizing information and are used in artificial intelligence, the Semantic Web, systems engineering, software engineering, biomedical informatics, library science, enterprise bookmarking, and information architecture as a form of knowledge representation about the world or some part of it. The creation of domain ontologies is also fundamental to the definition and use of an enterprise architecture framework.

Web mining techniques have shown promising performance in research experiments. Their actual deployment in live Web data, in contrast, has been fairly limited due to a lack of background semantics required for processing the text data, links, and other elements in Web pages. In this respect, an *ontology* which gives a conceptual description of the background semantics can serve as a very useful input to the Web mining problems. An ontology refers to a set of concepts and the relationships, together known as ontology entities, describing the information within an application domain.

When an ontology is used in solving a Web classification or extraction problem, the results obtained can be associated with the ontology entities making them easier to understand. This is a big advantage because each ontology often represents knowledge agreed upon by users and applications of a domain. For example, within the University domain, {Professor, Student, Course} and {Teach, Register, Supervise} are the common concepts and relationships respectively. University Web pages are likely to centered around these concepts and related concept instances are likely to be linked in one way or another. As the languages for defining ontologies and using the latter in marking up Web content become well accepted [8], we see an increasing use of ontology in Web mining. In this paper, we will give an overview of *ontology-based Web mining*. In ontology-based Web mining, we are often interested in discovering the instances of concepts and relationships in a given ontology, or using them to discover other useful knowledge. These Web mining techniques can potentially be deployed in a digital library system to enhance the access to Web content.

In this work, we offer a technique that solves the problems identified in the last paragraph for OWL 2 ontologies. The technique introduces 1) a formal contract that captures designer's assumptions he/she made about an ontology during development of the business logic; 2) process of its validation; and 3) transactional support aimed at keeping the ontology consistent and optimizing the access to the ontology. Furthermore, our approach allows competencies to be clearly distinguished in the application development process. Ontology and formal contract maintenance is the responsibility of the knowledge engineer educated in ontologies, but not necessarily having sufficient programming skills. On the other hand, the application developer(s) need not to be educated in ontological reasoning and knowledge engineering. Hence, this approach tends to significantly reduce the application development costs.

Furthermore, our approach makes use of an object-ontology mapping, similarly to approaches. However, on comparison with them, our approach brings two novel features:
1) explicit formal contract between the application and the ontology that enables applications to run over an evolving ontology;
2) transaction support that optimizes access to the ontology and ensures the application does not cause ontology inconsistency.

In this work it is proposed with a novel algorithm for generating knowledge in the web using an algorithm called OWL2-DL Language which includes the following features.

1) Contract stability—The contract has to be static or slowly evolving comparing to the ontology. The interface shall survive most ontology refinements.

2) Contract maintainability—The contract between an ontology and the respective object model has to be easy to establish and maintain.

3) Non-restrictive—The interface has to provide full access to the ontological knowledge, that includes entailment checking and query answering.

4) Validation—The interface has to ensure that modification of the ontology by the application violates neither the consistency of the ontology nor the contract between the application and the ontology.

In a service-based business process, customization may be enabled by automatically adapting the process to match the business partner’s practice indicated by their business processes. Such practice includes service interface specifications, Web Ontology Language (OWL)-service profiles, process models, and grounding. We would like to point out that, in this paper, we focus on the business scenarios where the business processes can be supported by dynamic and automatic service composition. In such scenarios, the instantiation of business processes allows a certain degree of flexibility in selecting business partners and adjusting the process parameters for the partners. In other words, here, we will only discuss service based business processes where the idea of automation of adaptation is applicable. We do not address many other circumstances where the choreography between processes must be well defined before execution in order to avoid any unacceptable conflict and loss.

Our experience proves that the decision to use OWL syntax for ontological axioms and integrity constraints at the same time is an advantage as it significantly simplifies integrity constraint authoring and management by the usage of state-of-the-art OWL tools, provided the original OWL ontology. For example in Protege, the integrity constraint designer creates a new OWL document for storing integrity constraints for the developed application. Then, by the usage of the import mechanism of OWL the original ontology is imported and new integrity constraints could be easily constructed that is based on the vocabulary and ontological axioms of the original ontology. In our methodology, integrity constraints are stored in a different OWL document than ontological axioms and, additionally, are distinguished using OWL annotations.

To ensure stability of the interface, the contract should be designed with as few integrity constraints as possible for proper application functionality. The more integrity constraints the contract contains, the less stable the contract is, resulting in more frequent object model revision.

Two important elements that enable an automatic way of processing business process customization are upper ontology of a business process and upper ontology of the process customization. Both upper ontologies have to interface with domain ontology during the inference by an inference engine.

Basically, three types of integrity constraints with regard to their relation to the object model and their evaluation strategy can be distinguished.

1) Compile-time constraints: They are those that can be compiled into the object model of the OO language under consideration. These constraints have the form α1 and α2. Integrity constraints of this type restrict the type of OO data field L(S) within an OO class L(A1):
a) when both α1 and α2 are present, the data field L(S) is of type L(A2),
b) when only α1 is present, the type of L(S) is a set of L(A2),
c) when only α2 is present, the type of L(S) is L(T).

2) Run-time constraints: They cannot be compiled directly into the object model, but their validation can be optimized in run-time by cheap procedural prechecks within the object model without the need to evaluate a DCQNOT query (see Section II). Currently considered run-time constraints have the form of cardinality constraints α3 and α4 from Table II, but we anticipate extending the run-time integrity constraint types as well as further optimizations of currently considered run-time constraint evaluation.

3) Reasoning-time: These constraints are all other integrity constraints that cannot be reduced4 to integrity constraints of aforementioned types. Reasoning-time integrity constraints cannot be cheaply checked by the OPL itself, but have to be evaluated by the usage of the DCQnot query engine.

Note that prechecking run-time cardinality constraints by simple counting of individuals introduced earlier fails whenever two or more individuals could be inferred to be the same, i.e., O |= i1 = i2 for different individuals i1 and i2 , (in addition to asserted i1 = i2 axioms). Fortunately, run-time cardinality restrictions are allowed in integrity constraints only in aforementioned option 2), which enforces the ontology to be at most SRI. In this case, no individuals can be inferred to be the same (as the Merge operation in the SROIQ tableau algorithm is not applicable), except individuals in the transitive closure of the = relation (i.e., assertions of type i1 = i2). This transitive closure is maintained by OPL and used to distinguish same individuals during cardinality restrictions checking.

The upper ontology of services, such as OWL-S, refers to the types of knowledge about a service such as what the service provides for prospective clients and how it is used. For a service-based business process, the classes and properties defined within the upper ontology of services directly convey all important information of a business process. Together with the structural constructs of the process, they facilitate the automation of executing the business process. Furthermore, with an upper ontology on business process customization like OWL-BPC, automatic customization of business processes is made possible due to the presence of a machine-understandable knowledge framework on what, when, and how customization should be performed.

To provide expressive query language for accessing ontological knowledge from ontology access layer, we propose the SPARQL − DLNOT language. Its advantage to other OWL query languages is that it is a direct extension of DCQNOT that supports additionally undistinguished variables (i.e., variables that do not need to be matched against existing individuals in the ontology, but to the inferred ones) and expressive meta-query atoms that allow us to retrieve OWL classes and properties in addition to individuals. An example of such query atoms are SubPropertyOf(x, y) for retrieving all sub property pairs, or DisjointWith(x, y) for retrieving pairs of disjoint classes.

II. Ontology access layer

The ontology access layer executes ontological queries / updates requested by the application logic. The main task of the layer is to provide transactional access to the ontology, with respect to the ACID (atomicity, consistency, isolation, durability) requirements. Atomicity, consistency and isolation is ensured by the transaction processing mechanism. Durability is ensured by creating a simple transaction log. As all object model changes can be expressed in terms of 1) OWL axiom additions, or 2) OWL axiom removals, the transaction log can be just a list of change records of these

Since ontology consistency checking and evaluation of ontology queries is time consuming, the front-end layer optimizes integrity constraint checking (FA2 and FA3) and keeps cached the ontology changes that are propagated to the ontology at transaction commit (FS4). Because of the time and space complexity of the ontology consistency check, it is not feasible that each transaction owns a separate instance of an OWL reasoned for its whole life duration.

To handle this problem, all transactions share one instance *R* of an OWL reasoner. During the commit phase, at the beginning of BA3, a new reasoner instance *R_* is created. When the commit succeeds (the changes violate neither integrity constraints nor ontology consistency), data are propagated to the ontology, *R_* becomes shared (the back-end layer benefits from serialized transactions, thus also achieving transaction isolation), replaces *R* (BA5), and all pending transactions are informed about reasoner change (BS3). On the other hand, transaction rollback causes *R_* to be disposed.

The transaction scenario requires that all queries in a transaction are evaluated (FR2, FS2, BR2, BA2, BS1, FR3, and FS3) before the first data modification (FR5) in the object model to prevent reading outdated data—for the majority of practical scenarios, this restriction is not serious, as data-editing clients fetch data first, provide them to a user interface and take back changed data to be stored in the ontology.

There are currently two back-end OWLAPI-based implementations:
1) a simple implementation that accesses ontologies in OWL files, and
2) a database-backed implementation that uses OWLDB9 to store OWL ontologies in a relational database. In both variants, any OWLAPI-compliant OWL2-DL reasoner can be used to check consistency, validate integrity constraints, and evaluate SPARQL − DLNOT queries (flow from FR2 to FS3, although Pellet [19] is currently preferred as it provides built-in optimized SPARQL − DLNOT support. In case of another OWL reasoner is used, we provide our out-of box SPARQL − DLNOT implementation OWL2Query10 as a part of JOPA. It can be used on top of any OWL2-DL reasoner to validate DCQNOT integrity constraints and evaluate SPARQL − DLNOT queries.

III. Conclusion

In this paper, we have presented a conceptualization of customizing service-based business processes according to the discrepancies and misalignments discovered in their business process description documents. We have also presented an ontology, i.e., the OWL 2-BPC, which we have developed for this purpose. Our main contribution in this paper is to tackle the problem of possible inconsistencies of collaborating business processes, including the following:

1) possible semantic inconsistency such as semantic mismatching on process parameters,
2) behavioral mismatches which may or may not be
3) compatible and
4) misaligned rendezvous requirements.

This ontology is used to build our framework for detecting and performing service-based business process customization. We have presented the detailed algorithm of the framework and its architecture.

REFERENCES

[1] Carroll. J. J, Dickinson. I, Dollin. C, Reynolds. D, Seaborne. A, and Wilkinson. K, (2004) “Jena: Implementing the semantic web recommendations,” in Proc. WWW, pp. 74–83.

[2] Ermolayev. V, Keberle. N, and Matzke. W. E, (2008) “An upper level ontological model for engineering design performance domain,” in ER, (Lecture Notes in Computer Science Series). Berlin: Springer, pp. 98–113.

[3] Horridge. M and Bechhofer. S, (2011). The OWL API: A java API for OWL ontologies. Semantic Web [Online]. 2(1), pp. 11–21. www.semanticwebjournal.net/sites/default/files/swj107_2.pdf

[4] Maedche. A, Motik. B, Stojanovic. L, Studer. R, and Volz. R, (2003) “Ontologies for enterprise knowledge management,” Intell. Syst., IEEE, vol. 18, no. 2, pp. 26–33.

[5] Motik. B, Patel-Schneider. P. F, and Grau. P. C, Eds. (2009 Oct.). OWL

[6] Story. H, (2008). “Semantic Object (Medata) Mapper,” (cited 6Oct.2010). [Online].Available:http://sommer.dev.java.net/sommer.

[7] Tran. T, Haase. P, Lewen. H, O´Mun˜oz-Garc´ıa, Go´mez-Pe´rez. A, and Studer. R, (2007) “Lifecycle-support in architectures for ontology-based information systems,” presented at the Int. SemanticWeb Conf./Asian Semantic Web Conf., Berlin, Heidelberg, Germany: Springer-Verlag, pp.508–522.

[8] Von Malottki. J, (2009) “Java OWL APIs.” (cited 6 Oct. 2010). [Online]. Available: http://wiki.yoshtec.com/java-owl-api

[9] Wang. Y, Liu. X, and Ye. R, (2008) “Ontology evolution issues in adaptable information management systems,” in Proc. IEEE Int. Conf. e-Business Engineering. Los Alamitos, CA: IEEE Computer Society, pp. 753– 758.

[10] Web Ontology Language Direct Semantics, ser. W3C RecommendationW3C. [Online].Available:http://www.w3.org/TR/2009/REC-owl2 direct-semantics- 20091027/

[11] (2007) [Online]. Available: http://protege.stanford.edu