International Journal of Engineering & Technology
CENTRE OF PROFESSIONAL RESEARCH PUBLICATIONS
ISSN 2049-3444

International Journal of Engineering and Technology Volume 2 No. 3, March, 2012

Assessment of Ontological Reuse versus Object Oriented Reuse Anchored in Various Reuse Subclasses

Shilpa Sharma[1], Maya Ingle[2]
MCA Department, [1]Medi-Caps Institute of Technology and Management, Indore, India
SCSIT,[2]Devi Ahilya University, Indore, India

ABSTRACT

It is extensively reckoned that the development and utilization of reusable software artifacts is necessary for improving software development efficiency and software prominence. Most software development methodologies recognize the utility of reuse, and some even provide processes and contrivances to directly support it**.** Therefore, an object-oriented software engineering methodology is grounded in the utility and benefits of the development and reuse of software artifacts. This paper examines the role of ontologies within software engineering to potentially apply the intentional development of reusable artifacts, representation and classification of artifacts into repositories, and utilization of the artifacts from repositories. Also, a range of classes of reuse are identified, inhibitors to successful reuse are pointed-out, and technical issues for achieving reuse are examined. An explicit apprehension is focused to ontological reuse amplification over the object oriented reuse in software engineering.

Keywords: *Ontological Reuse (OnR), Reuse Subclasses*

1. INTRODUCTION

In nature no object exists in isolation. 'Software reuse' is a method for developing new component, adding some extra functionality to the existing ones. Early software reuse practices focused on artifact's abstraction level such as Object Oriented Reuse (OOR) [1] [2] [3]. Conversely, reuse changed as the industry matured. Reuse became planned and systematic. Effective software reuse requires collection of designed-for-reuse software components and mechanisms to retrieve reuse candidates. Also, create new ones using the information provided by similar components [5]. In addition, there is need to bind those elements using a software process that really let to software reuse. In this context, ontologies can play an important role. Ontology is a formal explicit description of concepts in a domain of discourse [7]. An ontology may be used to enable multiple target applications to have access to heterogeneous sources of information that are expressed using diverse vocabulary or inaccessible format. Indeed, the ontology may be used as a basis for specification and development of software, allowing knowledge reuse.

Therefore, anticipated ontology is oriented toward a systematic method for reusing [16] [17]. Ontological Reuse (OnR), promotes common understanding among developers, and may be used as a basis for software specification and development. Consequently, any artifact of the software life cycle can potentially be reused such as domain knowledge, requirement specifications, architectural design, and code modules up to software applications. While it is generally accepted that building an application from scratch is a resource intensive process, the software development based OOR does not tap the full potential of existing domain-relevant knowledge sources.

Typically, implementing an OOR based application, does not resort to available knowledge and is implicitly tailored to specific application needs, which in turn means that it cannot be reused in different stipulation.
The growing demand for increasingly larger and complex systems is one of the reasons that ontology for reuse has become an alternative when dealing with these problems in software development. Thus, to effectuate the reuse, Section 2 introduces the Ontological Reuse (OnR) that promotes reuse from the beginning of the software development process. Also, an OnR is compared to OOR based on domain-specific and recreative process. In Section 3, a range of classes of software reuse are identified and a comparative study is proposed on the basis of these classes. In Section 4, we present a case study in support of OnR. Finally, benefits of using ontological reuse are draw round inn conclusion.

2. ONTOLOGICAL REUSE (OnR)

Ontological Reuse can be defined as the process in which available knowledge is used as input to generate new domain intensive application. Depending on the content of the knowledge sources and the domain overlapping, discrimination between ontological merging and integration can turn out [11].

2.1 OnR Process

Ontological reuse process starts with the identification of knowledge sources useful for the application domain, which differs both in the represented content, and in the formalization (thesauri, XML-Schemes and DTDs, UML diagrams, textual descriptions etc.). An automatic integration of the source knowledge means not only the translation of the representation languages to a common format, but also the matching of the resulting schemes. We have proposed the OnR process to be introduced early in the life-cycle of software development. It is a formal and well-documented process which is domain unambiguous and can be recreated as shown in Figure 1.

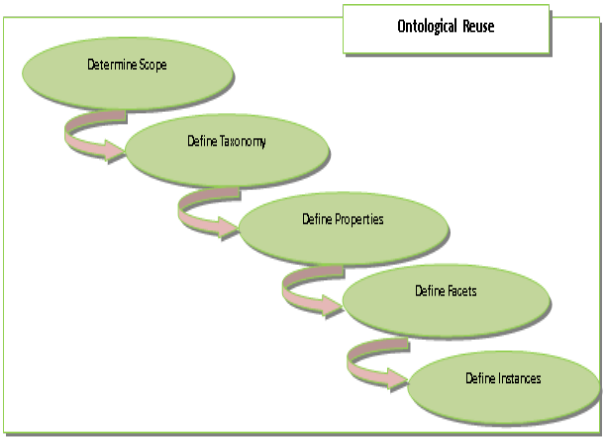

Figure 1: Ontological Reuse process

Determine Scope: It refers to defining concepts in the domain (classes). There is no correct ontology of a specific domain. Ontology is an abstraction of a particular domain, and there are always viable alternatives. This abstraction should be determined by the use to which the ontology will be put and by future extensions that are already anticipated.

Define Taxonomy: It ensures arranging the concepts in a hierarchy (subclass super class hierarchy). Opinions may differ on whether it is more efficient or reliable to do this in a top-down or a bottom-up fashion.

Define Properties: It relates with defining which attributes and properties (slots) classes can have and constraints on their values**.** While attaching properties to classes, it makes sense to immediately provide statements about the domain and range of these properties. There is a methodological apprehension here between generality and specificity such as flexibility (inheritance to subclasses) and detection of inconsistencies and misconceptions.

Define Facets: It asserts defining individuals and filling in slot values with Cardinality restrictions and Relational characteristics such symmetry, transitivity, inverse properties, functional values.

Define Instances: Filling the ontologies with instances is a step concerns with creating a knowledge base by defining individual instances of these classes and filling in specific property such as specific slot value information and additional slot restrictions.

2.2 Developing OnR

An available reuse methodology such as OOR addresses reusability issue only marginally. Though it mentions the possibility of reusing existing knowledge sources as input for the conceptualization phase, it fails to define precisely knowledge discovery and the subsequent evaluation of candidate knowledge. Also, it describes in detail to build and represent reusable artifacts, but furnish a relatively sketchy recommendation for supporting existing reusable artifacts. Figure 2 illustrates more pragmatic process an OnR, which exploit the OOR process to a maximal extent depending on the particular domain and level of formality. An OnR process extremely addresses this issue in the context of knowledge customization/pruning, explicitly extracting relevant fragments from very comprehensive, general purpose ontologies [15, 10]. In addition OnR provides a detailed description of reuse process and its implications in the overall engineering process.

We have proposed a generic and incremental process, concentrates on vocabulary of the input sources and subsequently insert additional information (semantic relationships, axioms, etc. if available explicitly) corresponding to application needs. An OnR process taps the full potential of OOR process from development and representation of reusable artifacts to supporting the reusable artifacts. First, reusable artifacts must be intentionally designed and developed. Following that, reusable artifacts must be represented, classified, and entered into and removed from appropriate repositories. Subsequently, tools and processes must be developed that support finding, understanding, modifying, and composing artifacts.

Figure 2 illustrates the mapping of OOR notions analogous to each of OnR notions. The development and representation of reusable artifacts of OOR are concerned with the identification, modeling, cataloging and disseminating a set of software artifacts that can be applied to existing and future software in a particular application domain. Besides, encodes the semantics of artifacts in such a way that a user, trying to solve a problem with own knowledge and semantics can locate an appropriate reusable artifact. Therefore, these notions mapped with determining the scope, taxonomy and properties. In OOR, once artifacts are represented, they must be classified and entered into repositories. As such, artifacts are classified in order to indicate the type and relation to other artifacts. There are two well-known schemes for doing this repository classification: enumerative and faceted [8]**.** Formerly artifacts have been developed, represented and categorized into repositories; the next concern is to utilize

this wealth of information specifically supporting the reusable artifacts. Thus, relates with defining the facets and instances of an OnR.

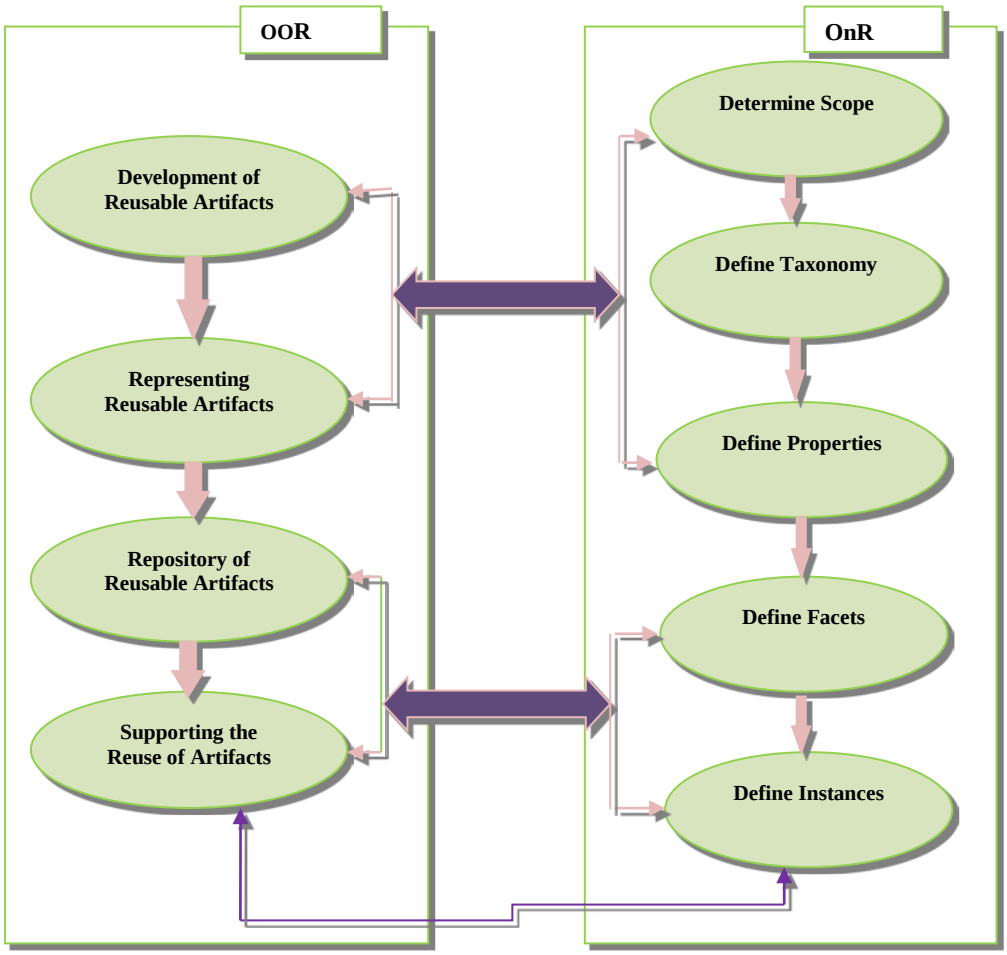

Figure 2: OOR to OnR

3. CATEGORICAL COMPARISON OF OOR AND ONR

Ontologies have great potential to deal with software reuse predicaments of various aspects such as domain specificity, fixed functionality, well bounded interfaces, performance expectations, and demonstrable excellence. In this section, we have identified range of subclasses of reuse which are described as follows:

- **Software component reuse subclass:** It concerned with a search for components that supply the functionality needed by the user. The underlying idea is to build a software component assortment with reference to the domain.

- **Software architecture/ design reuse subclass:** This class claims to be more than just component reuse since it is one of the software elements to be reused during the software process.

- **Software requirements reuse subclass:** This class in a system identifies capability, characteristic or quality factor in a system in order for it to have value and utility for sharing the requirements across various domains.

- **Software process reuse subclass:** It is a kind of reuse class that deals with the construction of reusable software processes as a means of improving the organization's software process.

- **Software technologies reuse subclass:** This subclass has been devised to investigate the application domains of software technologies.

- **Software experiences reuse subclass:** It describes the methods that try to reuse every useful experience in software systems development.

On the basis of these subclasses, we have presented a comparative study to systematically exemplify the object oriented reuse versus ontological reuse as shown in Table 1.

Table 1: OO Reuse vs. Ontological Reuse

S. No.	Reuse Subclass	Object oriented Reuse	Ontological Reuse
1.	**Software component reuse**	• Reduce the number of parameters • Avoid using options in constraints • Prohibit the direct access to instances • Implies more cohesive and primitive operations	• Enables multi faced description of components by lexically analyzed, stored, and indexed using the tokenization and indexing mechanisms • Generating semantic instances for the concepts and relations • Atomizes the retrieved instances based on a mixture of traditional IR and user context heuristics • Allows semantic matching based upon signature-based queries and metadata keyword queries.
2.	**Software architecture/ design reuse**	• Identify the system's responsibilities at a given level of abstraction • Scrutinize the system's environment to produce classes and objects • Partitioning the class and the object structure into larger units for various attributes and services • Identify generalization-specialization structure to capture inheritance.	• Allows analysis and comparison of different domain theories; • Structures knowledge acquisition for entities and relations • Provide a metalevel view (vocabulary and structure) on their • facilitates adequate system documentation; • Define assumptions that enable knowledge exchange between different agents.
3.	**Software requirement reuse**	• Do domain analysis and prototyping • Perform incremental development • Reconcile conflicting sets of requirements • Build flexibility by regular reviews	• Defines the optional parts of candidate system • Models the complex and alternative courses which seldom occur • Outline separate sub courses which are executed only in certain cases • Models the situation in which different modes can be inserted
S. No.	**Reuse Subclass**	**Object oriented Reuse**	**Ontological Reuse**
4.	**Software process reuse**	• Recognition of objects and operations • Grouping the objects and operations • Presents the pictographic description of objects • Justifying the design decisions	• Generate implementation for the Knowledge Processor • Generate a SIB subscription from the Ontology • Import /export model elements from /to the Repository • Generate a generic Task from the Ontology • Find an implementation from a Repository
5.	**Software technology reuse**	• Provides graph of each class hierarchy and collaboration for each subsystem • Specifies contracts supported by each class and subsystem • Presents computational description of each public object module.	• Based on open-source components and will be published as open-source • Support interoperability, and standard-based solutions are preferred. • Ready made Tool-level extensibility
6.	**Software experience reuse**	• Specify a set of basic building blocks for constructing primitive terms • Assembling and organizing structuring mechanisms; • Constructing and querying primordial operations	• Automating the process of creating application ontologies. • Support query formulation in an intuitive way to users • Provide means for visualizing, browsing and exploring domain • Provide adapted evaluation methods

		• Define the set of consistent symbol structure states, or changes of states accompanied by interpretation rules and usage guidelines.	

4. A CASE STUDY

As indicated in Table 1, it is vital to practice ontological reuse in software development for the precise granularity and for a high level of stability. The object oriented reuse practiced on domain level which restrains basic terms of a domain that combined and extended in the ontological reuse in order to describe more complex semantics. In allusion with our hydrology case, the concepts water level, water body and discharge are formalized. It is stated that every water body has a water level and a discharge and that these qualities can be observed and measured. This general description provides an entry point for the semantic search on *software requirement reuse* level. How measuring and representing is done for a specific water level measurement service is then sanctified at *software process reuse* level.

Once the domain level is settled, application ontologies can be added or removed without the need of modifications on domain level which makes the application level highly flexible and consequently referred to *software component reuse* level. The commitment to the same domain level makes the application ontologies comparable. Also, ad-hoc concepts (like query concepts) that are build on basis of the domain ontologies become comparable to all application concepts. The task of constructing application ontology lies in the responsibility of the provider of the information source whereas the construction of domain ontologies is a joint effort of domain experts that propose *software experience reuse* level. In our hydrology example, the fact that the water level is measured in centimeter and not in feet is stated on application level. Additionally all other peculiarities of this specific water level measurement service are describe on *software technology reuse* level. This could include legal information to use the provided information and data representation issues. The concepts used to describe non geospatial aspects are taken from other types of domain ontologies such as measurement ontologies or data representation ontologies that contaminate *software architecture/ design reuse* level.

5. RESULTS AND CONCLUSION

One of the main advantages of the ontology is its comprehensibility. The ontological reuse achieves some lucidness of unclear concepts related with software reuse. Besides, the concepts were linked rigorously. A significant aspect of the ontological reuse is it is independent from implementations or technological aspects. The proposed ontological reuse allocates various software reuse subclasses with ensuing benefits such as

- Cost reduction: Smaller number of software requirements has to be specified, designed, implemented and validated.
- Higher reliability and quality: Components tested in previously functioning systems are more reliable than new ones.
- Risk reduction: A previously existing process implies a lesser degree of uncertainty with respect to cost estimation for the project.
- Accelerated system development: Software architecture/ design reuse results in shorter development and validation times.
- Effective use of specialists: Instead of application specialists doing the same work in different projects, these specialists can develop software that encapsulates their knowledge.
- Standards compliance: Some standards, such as interface standards, can be implemented as a set of standard reusable technologies.

REFERENCES

[1] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. *Object-Oriented Modeling and Design*. Prentice Hall, 1991.

[2] Shlaer, S. and Mellor, S. *Object-Oriented Systems Analysis: Modeling the World in Data*. Prentice Hall, 1998.

[3] Coad, P. and Yourdon, E. *Object-Oriented Analysis.* Yourdon Press, Englewood Cliffs, NJ, 1990.

[4] Dardenne, A., van Lamsweerde, A., and Fickas, S. Goal-directed requirements acquisition. *Science of Computer Programming 20* (1993), 3–50.

[5] M. Ramachandran, "Software Reuse Guidelines", *ACM SIGSOFT Software Engineering Notes*, 2005, Vol. 30, No.3, pp. 1-8.

[6] W. B. Frakes and K. Kang, "Software Reuse Research: Status and Future", *IEEE Transactions on Software Engineering*, 2005, Vol. 31, No. 7.

[7] N. Noy. and D. McGuinness. "Ontology Development 101: A Guide to Creating Your First Ontology", Protégé Documentation, Stanford University: Stanford, CA, 2001.

[8] SEI, "Domain Engineering", Carnegie Mellon University, 2004, retrieved online, available in:<http://www.sei.cmu.edu/domain-engineering/domain_eng .html>.

[9] R. Pietro-Díaz, "Domain analysis: an introduction", ACM SIGSOFT Software Engineering Notes, ACM Press, New York, 1990, pp. 47-54.

[10] R. S. Pressman, Software Engineering: a practitioner's approach, *Ingeniería del Software, Un enfoque práctico*, Mc Graw Hill, New York, 2005.

[11] J. Sodhi and P. Sodhi, *Software Reuse, Domain Analysis and Design Process*, Computing McGraw Hill, 1999. I. Sommerville, *Software Engineering*, Pearson Education, 2004.

[12] D. J. Reifer, *Practical Software Reuse, Strategies for introducing reuse concepts in your organization*, John Wiley & Sons Inc, 1997.

[13] K.C. Kang, S.H. Cohen, J.A. Hess, W.E. Novak and A.S Peterson, "Feature-oriented domain analysis feasibility study: interim report", *Technical Report CMU/SEI-90-TR 21*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1990.

[14] M.D. Lubars, "Domain analysis and domain engineering in IDeA", *IEEE Tutorial: Domain Analysis and Software Systems Modeling*, IEEE Computer Society Press, California, 1991, pp.163-178.

[15] R.Prieto-Díaz, "Domain analysis process model, a domain analysis method developed by IBM federal sector division and available in reuse library process model", *Report IBM STARS CDRL 03041-002*, U.S. Air Force Electronic Systems Center, Hanscom Air Force Base, MA, 1991.

[16] OMG, "Reusable Asset Specification, OMG Available Specification Version 2.2", formal 05-11- 02, www.omg.org.

[17] P. Shoval, V. Maidel, Bracha Shapira, "An ontology content-based filtering method", I.Tech-2007 –Information Research and Applications.

[18] Shilpa Sharma, Maya Ingle , *"Study of Object Oriented Software Engineering versus Ontology Engineering"*, National Conference on Emerging Technologies in Electronics, Mechanical and Computer Engineering, IIST, Indore, 2010

[19] Shilpa Sharma, Maya Ingle, "An Ontology Aided Requirement Engineering Framework", International Journal of Advanced Research in Computer Science, Volume 2, No. 1, Jan-Feb 2011

