Ontology in Software Engineering

Salvatore F. Pileggi
School of Information, Systems and Modelling
University of Technology Sydney
Australia
Email: SalvatoreFlavio.Pileggi@uts.edu.au

Antonio A Lopez-Lorca
School of Computing and Information Systems
University of Melbourne
Australia
Email: antonio.lopez@unimelb.edu.au

Ghassan Beydoun
School of Information, Systems and Modelling
University of Technology Sydney
Australia
Email: Ghassan.Beydoun@uts.edu.au

Abstract
During the past years, ontological thinking and design have become more and more popular in the field of Artificial Intelligence (AI). More recently, Software Engineering (SE) has evolved towards more conceptual approaches based on the extensive adoption of models and meta-models.

This paper briefly discusses the role of ontologies in SE according to a perspective that closely matches the theoretical life-cycle. These roles vary considerably across the development lifecycle. The use of ontologies to improve SE development activities is still relatively new (2000 onward), but it is definitely no more a novelty. Indeed, the role of such structures is well consolidated in certain SE aspects, such as requirement engineering. On the other hand, despite their well-known potential as knowledge representation mechanisms, ontologies are not completely exploited in the area of SE.

We first (i) propose a brief overview of ontologies and their current understanding within the Semantic Web with a focus on the benefits provided; then, the role that ontologies play in the more specific context of SE is addressed (ii); finally, we deal with (iii) some brief considerations looking at specific types of software architecture, such as Multi-Agent Systems (MAS) and Service-Oriented Architecture (SOA).

The main limitation of our research is that we are focusing on traditional developments, where phases occur mostly sequentially. However, industry has fully embraced agile developments. It is unclear that agile practitioners are willing to adopt ontologies as a tool, unless we ensure that they can provide a clear benefit and they be used in a lean way, without introducing significant overhead to the agile development process.

Keywords Ontology, Software Engineering.
1 Introduction

During the past years, ontological thinking and design have become more and more popular in the field of Artificial Intelligence (AI). More recently, Software Engineering (SE) has evolved towards more conceptual approaches based on the extensive adoption of models and meta-models (Henderson-Sellers 2011). A recent survey on applications of ontologies for SE (Bathia et al. 2016), provides a broad analysis of current issues and challenges for the years to come. The authors define a fine-grained classification of the ontologies based on their type and scope. The main categories considered within the survey are upper ontology, software process ontology, domain ontology, requirement ontology, architecture and design ontology, pattern ontology, implementation ontology, documentation ontology, quality ontology, maintenance ontology and technology ontology.

In this paper we reflect on the adoption of ontologies in the field of SE according to a role-based perspective that maps the SE life-cycle much closely than in Bathia et al. (2016). We focus our discussion on Requirements Engineering (RE). Furthermore, we briefly discuss the use of ontologies in the context of Multi-Agent Systems (Van der Hoek and Wooldridge 2008) and Service-Oriented Architecture (Stantchev and Malek 2010) development. As far as the authors know, there are no surveys or papers on the topic adopting a similar approach. However, we have identified a number of works that focus vertically on specific aspects or sub-problems. For instance, Gigante et al. (2015) deal with the role of semantics in requirements verification, while Ding et al. (2014) review knowledge-based approaches in software documentation. These contributions are valuable and informative, however, they lack the big picture view that holistic approaches can provide. This paper first proposes a brief overview of ontologies and their current understanding within the Semantic Web (Berners-Lee et al. 2001) with a focus on the benefits provided. Then, the role that ontologies play in the more specific context of SE is addressed. We focus on the RE activities while maintaining a holistic view of the SE life-cycle. Finally, the paper briefly considers the cases of Multi-Agent Systems (MAS) and Service-Oriented Architecture (SOA).

2 Ontology and Web Semantics

The term “ontology” was originally coined in ancient Greece in metaphysics, a major branch of philosophy at the time, to mean “the study of what is there”. This included for example, the question of whether or not there is a God (Hofweber 2018). AI community adopted the term in the 1980’s to describe a central component of knowledge-based systems (Akerkar and Sajja 2010), to conceptualise and to theorise a modelled world (Gruber 2009). In the early 1990’s Gruber (1993) defined ontology as an “explicit specification of a conceptualisation”, a formalization of the definition suitable to many systems.

More recently, ontology has become one of the key concepts underpinning the Semantic Web model (Chandrasekaran et al. 1999) where, unlike other data models, ontology is supposed to address the “meaning” of the target data, information or knowledge. Indeed, in a Web context, ontology is understood as a rich data model, developed upon the Web infrastructure, able to represent complex resources and the relations among them. As resources are univocally identified worldwide by adopting IRIs (Web resources), ontologies work according to an enhanced model of interoperability, commonly referred to as Semantic Interoperability (Obrst 2003). Furthermore, ontology overcomes the most basic understanding of the Semantic Web (e.g. Linked Data (Bizer et al. 2011)) by providing capabilities for standard reasoning, normally based on the specification of inference rules.

Ontologies may have different scopes and purposes. They are currently adopted in a wide range of domains and discipline. In the context of this work, we mostly refer to domain ontologies which, specify a given-domain by defining its objects, the relationships among them and axioms that govern the domain. The main goal of a domain ontologies is to represent a shared view of a domain. Generally speaking, the model of a domain assumes all parties involved in their use agree on the represented conceptualisation, meaning that the concepts adopted are expected to be well-known and accepted.

Ontologies can be defined as Web ontologies using the OWL language (McGuinness and Van Harmelen 2004). This formalisation relies on Axioms which define facts involving Web resources and literals that are supposed to be always true in the considered context. Web ontologies operate under the assumption of an open world, the falsehood of unknown facts is not assumed. Key elements of OWL ontologies are:

- **Classes and class hierarchy** which determine the type of concepts defined in the ontology, i.e. that exist in the domain. For example, a generic class “person” belonging to a given domain can assume two sub-classes, e.g. “worker” and “student”, defining a hierarchy.

- **Individuals** are normally instances of classes, although it is possible to define un-typed individuals.
Most software development processes share a core set of phases (Tsui 2009) (Figure 1, left):

- **Analysis phase** is aimed at extracting the system requirements from the customer and at building the models to understand the problem.
- In the **design phase**, designers think in terms of the solution to define the architecture of the software without concerning themselves with low-level operational detail.
- Within the **codification phase**, programmers materialize the architecture defined in the previous step by using a concrete programming language.
During the testing phase, the different artefacts produced during the software development are verified and validated to assure their quality and compliance with the original requirements.

Finally, once the product has been delivered to the client, it starts the last phase, maintenance, where errors are identified and fixed. Furthermore, new requirements may be eventually added to the system.

The analysis phase is commonly known as requirement analysis phase and, because of its importance, is often object of special attention. It can be further divided in sub-tasks, each one characterised by the following activities (Sadraei et al. 2007):

- **Elicitation** identifies high level goals of the target system, requirements for different groups of users, and the tasks to be accomplished, along with system boundaries. The normal outcome is the early requirements document.

- **Requirement Analysis** analyses the requirements to uncover conflicts, ambiguities, missing or duplicate requirements in order to identify alternatives and convert them into a structured and unambiguous representation. The analysis outcome is an early requirements model.

- **Negotiation** addresses key trade-offs to achieve agreement between stakeholders.

- **Verification and Validation** examine requirements to find any deficiencies in consistency, accuracy and adequacy. It may also include a feasibility analysis to verify the cost of development. After verification and validation, the requirements model should be understood by systems analysts.

Concerning the maintenance phase, two commonly accepted sub-phases may be identified:

- **Change Management** recognizes changes through continuous requirements elicitation, re-evaluation of risk and evaluation of the system in its operational environment (Nuseibeh and Easterbrook 2000), to assure that all relevant information for each change is collected.

- **Requirement Tracing** manages the evolution of requirements, maintaining traces about its history to track the origins of each requirement, so that if a change has to be made to a design component, the original requirement can be located (Davis 1993).

Looking at the typical SE life-cycle with a focus on analysis and maintenance (Figure 1, left), we have identified four different major roles (Figure 1, right) as follows:

- **Consistency and completeness checking.** Ontologies define formal semantics based on some logic. It contributes and can facilitate the automation of important modelling tasks, such as consistency and completeness checking. Consistency checks avoid inconsistent use of knowledge, i.e. to assert truth and falsehood of a given fact at the same time. Automated consistency checking enables the
detection of such assertions through a chain of systematic logical steps. During consistency checking implicit knowledge can also be explicated and modelled. Ontologies can be used this way to complete (Kaiya and Saeki 2005) models acting as reference frames to identify modelling gaps or lacks.

- **Understanding.** By formally defining relations in a domain, ontologies can facilitate the understanding of the target system (e.g. Graja et al. 2011). An ontological approach allows to deconstruct the problem, splitting it into fragments that are easier to understand and analyze. Furthermore, these fragments can be classified, which helps comprehending the underlying structure and eventually defining further relations. Annotation can then be used to augment the concepts of an ontology with metadata that describe them and give further information. These annotations combined with formal relations and structural knowledge, make possible to address a subsequent complexity, suitable for humans and machines both. Semantic query (Ray 2009) provides a powerful interface that enables flexible interaction with ontologies.

- **Integration.** The features of semantic technologies intrinsically enable further capabilities in terms of integration (e.g. Paulheim 2009) within heterogeneous systems, i.e. facilitating that different systems to interoperate and share knowledge. Ontologies are technology-independent knowledge modelling artefacts and can be used in heterogeneous contexts in a wide range of technologies.

- **Reuse.** As previously discussed, ontologies holistically define knowledge by relating atomic concepts. In the context of SE, ontologies are usual to separate domain knowledge from operational knowledge. This facilitates the reuse (Uschold et al. 1998) at two levels: first of all, the ontology schema can be reused as a shared knowledge conceptualization; moreover, it can facilitate the reuse of the concrete artefacts that they describe.

### 3.1 Ontology-driven MAS and SOA Development

The effectiveness of ontology-driven development techniques becomes evident in the moment in which more specific paradigms are considered. Figure 2 shows the theoretical life-cycle for MAS (Van der Hoek and Wooldridge 2008) and SOA (Stantchev and Malek 2010). An exhaustive analysis is out of the scope of the paper. However, it is possible to identify a number of contributions in literature, that adopt an ontology-driven approach for the development of a specific kind of software architectures.

There are cases in which ontologies are applied to generic aspects. That is, for example, the case of the methodology adopted by Cossentino et al. (2010) which adopts an ontological approach in early stages of analysis to describe the problem domain concepts and to complement the requirements description in terms of use cases in MAS. As well as, Tran and Low (2008) focus on ontological capabilities to produce MAS whose components are interoperable and reusable.

Some other works attempt to use ontologies to improve aspects, specific of a kind of architecture. For instance, Girardi and Leite (2008) address explicitly the model of the MAS domain. Nyulas et al. (2008) present an ontology-driven framework for deploying MAS upon JADE (Bellifemine et al. 2007). Sensoy et al. (2010) and Fornara and Colombetti (2009) use OWL ontologies to define the policies that rule agent behaviour.

![Figure 2: Theoretical MAS (left) and SOA (right) lifecycle.](image-url)
4 Conclusion and Future Work

This paper briefly discusses the role of ontologies in SE according to a perspective that closely matches the theoretical life-cycle. These roles vary considerably across the development lifecycle. The use of ontologies to improve SE development activities is still relatively new (2000 onward) but it is definitely no more a novelty. Indeed, the role of such structures is well consolidated in certain SE aspects, such as requirement engineering. On the other hand, despite their well-known potential as knowledge representation mechanisms, ontologies are not completely exploited in the area of SE.

We believe that ontology-driven software development will be consolidated in the next future, becoming more popular and, eventually, extensively adopted in fact. To contribute to this goal, we seek to explore uses of ontologies for the development of both MAS and SOA. We will attempt to harness their potential, including automatic reasoning, as a tool to enable the verification and validation of these systems. This is, ensuring consistency and traceability of artefacts along the development life-cycle, and consistency against the client needs, respectively.

One limitation of our research, is that we are focusing on traditional developments, where phases occur mostly sequentially. However, industry has fully embraced agile developments. It is unclear that agile practitioners are willing to adopt ontologies as a tool, unless we ensure that they can provide a clear benefit and they be used in a lean way, without introducing significant overhead to the agile development process.

5 References


Copyright: © 2018 authors. This is an open-access article distributed under the terms of the Creative Commons Attribution-NonCommercial 3.0 Australia License, which permits non-commercial use, distribution, and reproduction in any medium, provided the original author and ACIS are credited.