SABiO: Systematic Approach for Building Ontologies

Ricardo de Almeida Falbo

Ontology and Conceptual Modeling Research Group (NEMO),
Federal University of Espírito Santo, Vitória, Brazil
falbo@inf.ufes.br

Abstract. This paper presents the new version of SABiO - a Systematic Approach for Building Ontologies. SABiO focus on the development of domain ontologies, and also propose support processes. SABiO distinguishes between reference and operational ontologies, providing activities that apply to the development of both types of domain ontologies.

Keywords: ontology engineering, domain ontology, ontology development

1 Introduction

Nowadays, the Ontology Engineering community recognizes the great help Software Engineering can provide for improving practices in the ontology engineering process. Like any complex software development activity, building quality ontologies requires appropriated methods and tools. In 1997, we defined SABiO, a Systematic Approach for Building Ontologies, whose first version was published in [1]. Since then, SABiO has been used for building several domain ontologies, such as ontologies for the software process domain [2], and cardiology domain [3]. In [4], we discussed strong points and weakness of the method, presenting some lessons learned and improvement opportunities to evolve it. Minor changes, such as using a UML profile as modeling language, were done in SABiO at that time.

SABiO was originally conceived for supporting the development of domain *reference ontologies*. By domain reference ontology we mean a domain ontology that is built with the goal of making the best possible description of the domain. It is a solution-independent specification (conceptual model) with the aim of making a clear and precise description of domain entities for the purposes of communication, learning and problem-solving. Once users have already agreed on a common conceptualization, operational versions (machine-readable ontologies) of a reference ontology can be implemented. Contrary to reference ontologies, operational ontologies are designed with the focus on guaranteeing desirable computational properties [5].

As pointed by several works, such as [5, 6], ideally domain ontologies should be developed grounded in foundational (top-level) ontologies. Concepts and relations in a domain ontology must be previously analyzed in the light of a foundational ontology. The idea behind ontological analysis is to provide a sound foundation for modeling concepts, if assumed that such concepts are aimed at representing reality [7].

Among the benefits of ontological analysis, we can cite [8, 9]: (i) the rigorous definition of models, in terms of real-world semantics; (ii) the identification of problems in the definition, interpretation or usage of concepts; and (iii) recommendations for model formality improvements. We ourselves experienced these benefits when ontologically analyzing the software process ontology [2], when several problems in the ontology were detected and the entire ontology re-engineered [10].

In order to incorporate these experiences, and the important distinction between reference and operational ontologies, we evolved SABiO. Moreover, we take the opportunity for incorporating to SABiO best practices commonly adopted in Software Engineering and Ontology Engineering. The main differences between versions 1.0 and 2.0 of SABiO are: (i) SABiO 2.0 extends the original development process to consider the design, implementation and test of operational ontologies; (ii) SABiO 2.0 considers new support processes for knowledge acquisition, reuse, and configuration management; (iii) SABiO 2.0 recognizes the importance of the use of foundational ontologies in the development of domain ontologies, and proposes applying ontological analysis during ontology capture; (v) Regarding ontology evaluation, SABiO 2.0 addresses both ontology verification and validation.

This paper presents the new version of SABiO, and it is organized as follows. In Section 2, we present an overview of SABiO 2.0. Section 3 presents SABiO's development process, describing each one of its activities, as well as the roles involved in their accomplishment, artifacts required and produced, and some techniques and guidelines for performing them. Section 4 presents SABiO's support processes. Section 5 discusses related works. Finally, Section 6 presents our final considerations.

2 An Overview of SABiO

Figure 1 presents an overview of SABiO 2.0. As this figure shows, SABiO development process comprises five main phases: (i) Purpose identification and requirements elicitation; (ii) ontology capture and formalization; (iii) design; (iv) implementation; and (v) test. Support processes are performed in parallel to the development process. Although Figure 1 suggests a somewhat sequential workflow, SABiO does not prescribe any specific life cycle model. Thus, life cycle models such as Incremental and Spiral can be adopted. We recommend incremental and iterative development.

SABiO focuses on the development of domain ontologies. Two types of domain ontologies can be developed: reference and operational ontologies. As aforementioned, a reference ontology is a special kind of conceptual model. An operational ontology, in turn, is a machine-readable implementation version of the ontology, designed with the focus on guaranteeing some desirable computational properties [5]. Thus, before implementing an operational ontology, a design phase should be accomplished taking technological non-functional requirements and the ontology implementation environment into account. If someone is interested in building a reference ontology, then she shall accomplish only the first three activities of the development process. If someone wants to build an operational ontology, then she has to perform the entire development process.

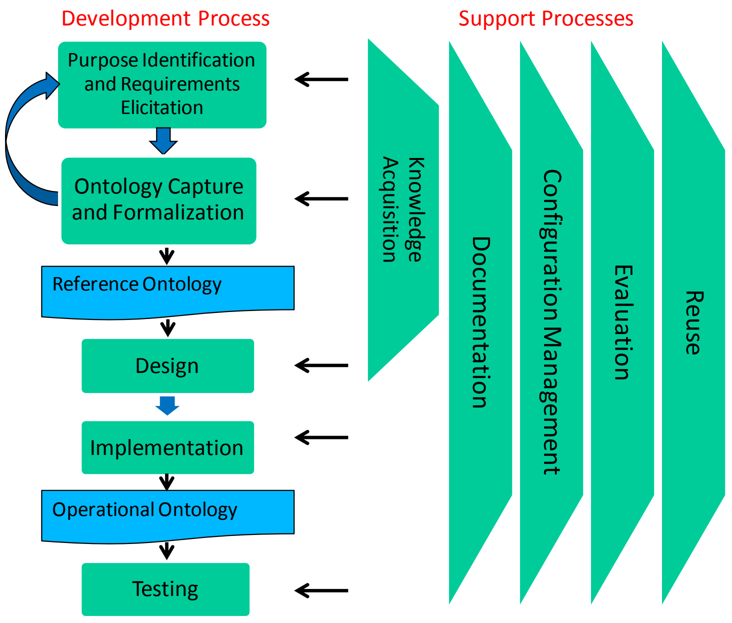

Fig. 1. SABiO's Processes

3 SABiO's Development Process

SABiO development process comprises five main phases. Each phase is composed of activities, which are to be performed by workers playing certain roles. The main roles considered in SABiO are: (i) domain expert, who is a specialist in the ontology domain, and provides the knowledge that is to be modeled and implemented in the domain ontology; (ii) ontology user, representing someone who intends to use the ontology for a given purpose; (iii) ontology engineer, who is responsible for the reference ontology, and thus is responsible for the initial phases of the ontology development process; (iv) ontology designer, who is responsible for the design of an operational ontology; (v) ontology programmer, who is responsible for implementing an operational ontology; (vi) ontology tester, who is responsible for testing an operational ontology. It is worthwhile to point out that a given worker can play several roles in a given project. For instance, generally the roles of ontology designer and ontology programmer are played by the same worker, since both roles require great knowledge regarding the ontology implementation environment.

Ontology Purpose Identification and Requirements Elicitation

The first phase in SABiO's development process is Purpose Identification and Requirements Elicitation. As Figure 2 shows, this phase comprises four activities that occur in an iterative way, all of them involving the participation of the ontology engineer, domain experts and potential ontology users.

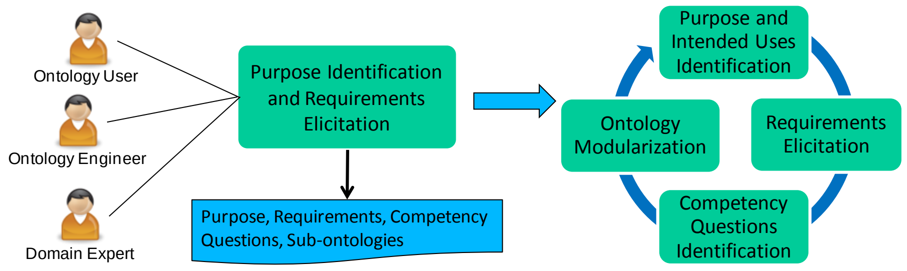
Ontology User
Ontology Engineer
Domain Expert
Purpose Identification and Requirements Elicitation
Purpose, Requirements, Competency Questions, Sub-ontologies
Purpose and Intended Uses Identification
Requirements Elicitation
Competency Questions Identification
Ontology Modularization

Fig. 2. The Purpose Identification and Requirements Elicitation Phase.

Initially, we need to identify the ontology purpose and its intended uses. Once defined the ontology purpose and intended uses, we should elicit its requirements. Ontology requirements, like software requirements, can be divided into functional and non-functional requirements. Functional requirements refer to the knowledge (content) to be represented by the ontology, and can be stated as competency questions, i.e. the questions that the ontology should be able to answer [11]. By establishing the competency questions, we reach an effective way to determine what is relevant to the ontology and what is not, i.e. we define its scope. Moreover, we provide a way for its evaluation. On the other hand, non-functional requirements refer to the characteristics, qualities and general aspects not related to the ontology content [12]. Non-functional requirements can be divided into: (i) ontology quality attributes, which refer to characteristics that the ontology, as a software product, should present, such as reasoning performance and availability (for the case of operational ontologies), usability (e.g., understandability), maintainability (e.g., extensibility); (ii) project requirements, which are requirements derived from the ontology project, such as process requirements (e.g., adherence to process models and document templates), implementation requirements (e.g., whether the operational ontology must be implemented in a given machine-readable language), delivery requirements (e.g., time to market), consensus-related requirements (e.g., who must agree with the ontology); (iii) intended uses-related requirements, which take the intended uses for the ontology into account, such as knowledge sources-related requirements (e.g., whether the terminology to be used in the ontology must be taken from standards), interoperability requirements (e.g., whether the ontology must be grounded in a specific foundational ontology in order to easy its integration with other ontologies already existing).

Functional requirements should be written in the form of competency questions (CQs). CQs can be identified using different strategies, and written in different granularity levels. Some strategies for identifying CQs are [12]: (i) Top-down: the ontology engineer starts with complex questions that are decomposed in simpler ones; (ii) Bottom-up: the ontology engineer starts with simple questions that are composed to create complex ones; (iii) Middle out: the ontology engineer starts just writing down important questions that are composed or decomposed later on to form abstract and simple questions, respectively. Whatever strategy used, in the end, we need to achieve a set of CQs in both levels. Simple CQs are important for deriving test cases; complex and more abstract CQs are important to guide ontology modularization.

If the domain of interest is complex, we need to modularize the ontology. Ontology modularization consists in identifying modules (or sub-ontologies) that can be considered separately while they are interlinked with other modules [12]. The benefits of modularizing ontologies include [12]: (i) to facilitate the development and maintenance of the ontology by dividing it in loosely coupled, self-contained modules; (ii) to facilitate the reuse of parts of the ontology; (iii) to improve performance by enabling distributed processing. However, there is no universal way to modularize an ontology and the choice of a particular approach should be guided by the ontology requirements [12]. SABiO suggests decomposing the ontology into sub-ontologies. Techniques and criteria for ontology partitioning, such as the ones proposed in [13], apply. Regarding the criteria, we suggest considering at least the following ones: independence, cohesion, and size. UML package diagrams should be developed for graphically showing the sub-ontologies and the dependencies between them. If a sub-ontology *so2* refers to a concept defined in another sub-ontology *so1*, then *so2* depends on *so1*. The four rules for partitioning ontologies proposed in [12] are also useful here.

The four activities in Figure 2 must be performed in an iterative way. Based on the purpose initially established for the ontology and the intended uses elicited with potential users of the ontology, requirements are elicited and competency questions are outlined. The ontology engineer may then identify sub-ontologies and allocate the competency questions to the sub-ontologies identified. The growing understanding of the domain can lead to a better understanding of the purpose of the ontology and the identification of new intended uses, starting a new cycle.

Ontology Capture and Formalization

The main goal of this phase is to capture the domain conceptualization based on the competency questions. As discussed in the Introduction of this paper, SABiO suggests that concepts and relations in a reference domain ontology are analyzed in the light of a foundational ontology. Ontological analysis can be applied in several scenarios: portions of the ontology that are developed from scratch should be analyzed, as well as non-ontological and ontological resources to be reused. A foundational ontology must be selected for performing this task.

The relevant concepts and relations should be identified and organized. A graphical model is a key instrument for supporting communication, meaning negotiation and consensus establishment with domain experts. For building reference domain ontologies, highly-expressive languages should be used to create strongly axiomatized ontologies that approximate as well as possible to the ideal ontology of the domain. The focus on these languages should be on representation adequacy, since the resulting specifications are intended to be used by humans [5]. An example of an ontology representation language that is suitable for reference ontologies is OntoUML, a UML class diagram profile that incorporates important foundational distinctions made by the Unified Foundational Ontology (UFO) [14].

As Figure 3 shows, this phase begins with a conceptual modeling activity. During conceptual modeling, initially the ontology engineer should identify the main concepts and relations in the domain. Concepts should be properly organized in taxonomies. Since SABiO suggests the use of OntoUML, the ontology engineer should classify each concept according to the types defined in OntoUML (kind, subkind, phase, role, category, rolemixin etc.). Moreover, OntoUML constraints for relating these types should be respected. Ontological patterns and modeling rules inherent to OntoUML, such as the ones for modeling subkinds, phases, roles [15] and rolemixins [14], should be applied in order to achieve consistent conceptual models.

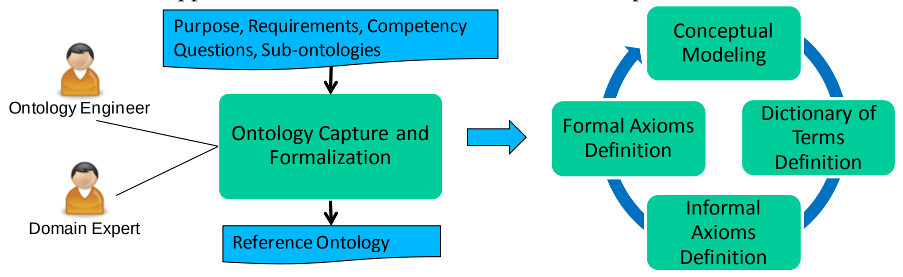
Purpose, Requirements, Competency Questions, Sub-ontologies
Ontology Engineer
Domain Expert
Ontology Capture and Formalization
Reference Ontology
Conceptual Modeling
Formal Axioms Definition
Dictionary of Terms Definition
Informal Axioms Definition

Fig. 3. The Ontology Capture and Formalization Phase.

Ontology capture is strongly supported by the knowledge acquisition process. Knowledge can be elicited from domain experts, as well as from sources of consolidated knowledge, such as books, international standards, and reference models. Concepts, relations and properties reused from non-ontological resources should be ontologically analyzed in the light of the selected foundational ontology; fragments of reused ontologies that are not grounded in the same foundational ontology too.

Concepts and relations are the basis of an ontology, but they can be not enough to capture the domain conceptualization. Constraints must also be taken into account. Thus, axioms specifying constraints and inferences should be specified. Initially, we do not need to write down formal axioms, rather the axioms should be written in natural language, simply reflecting inferences and constraints on the universe of discourse.

Ontology capture should be guided by the competency questions. The ontology elements (concepts, relations, properties and axioms) in the ontology must be necessary and sufficient to answer the competency questions. If the ontology elements are not enough for this purpose, then additional concepts, relations, properties or axioms must be added. In this sense, the ontology capture is an iterative process, strongly linked with the evaluation process.

During formalization, the informal axioms should be written in a formal language. In this language, in contrast to the natural language, we have signs that are unambiguous and formulations that are exact and, therefore, clarity and correctness can be more easily evaluated. On the other hand, formal axioms are not able to substitute their descriptions in natural language; rather, they are to be used to support these descriptions. In fact, each representation plays a specific role.

It is worthwhile to point out that in SABiO, formalization regards solely to writing formal axioms from the informal ones. Thus, formalization should not be confused with ontology implementation, when the entire ontology is codified in a machine-readable language. In order to allow describing complex constraints and inference rules, highly-expressive languages, such as such as first order logics and OCL, should be used.

Finally, many times, during the ontology capture and formalization phase, new competency questions for the ontology arise. Thus, iteration with the previous phase (Purpose Identification and Requirements Elicitation) is very common.

Ontology Design

Once a reference ontology is produced, many times we want to get an operational version to be used by computer applications. In order to achieve this operational version, we need to design and implement it in a particular machine-readable ontology language (e.g. OWL). In the design phase, the conceptual specification of the reference ontology should be transformed into a design specification by taking into account a number of issues ranging from architectural issues and technological non-functional requirements, to target a particular implementation environment. The same reference ontology can be used to produce a number of different designs [5].

Figure 4 shows the activities that comprise the ontology design phase. First, the ontology engineer must work together with the ontology designer to complement the list of technological non-functional requirements for the operational ontology, and to define the environment on which it will be implemented. Differently from reference ontologies, operational ontologies are not focused on representation adequacy, but are designed with the focus on guaranteeing desirable computational properties. The design phase, thus, aims at bridging the gap between the conceptual modeling of reference ontologies and the coding of them in terms of an operational ontology language [5]. Issues that should be addressed in the design phase include: determining how to deal with the differences in expressivity of the languages that are used in each of these phases, and how to produce lightweight specifications that maximize specific technological non-functional requirements, such as reasoning performance.

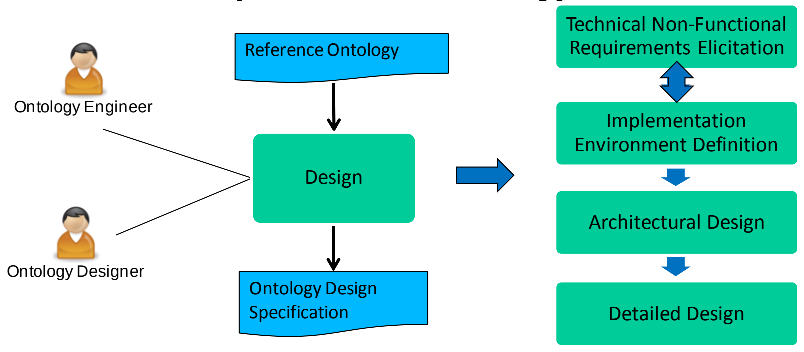
Ontology Engineer
Ontology Designer
Reference Ontology
Design
Ontology Design Specification
Technical Non-Functional Requirements Elicitation
Implementation Environment Definition
Architectural Design
Detailed Design

Fig. 4. The Ontology Capture and Formalization Phase.

Once defined the implementation environment, the ontology designer must revisit the ontology modularization defined in the beginning of the project. Now, she has to take the technological non-functional requirements and the characteristics of the implementation environment into account to define the ontology architecture.

Finally, during detailed design, the ontology designer has to address the problems related to the lower expressivity of the operational languages when compared to the models expressed in OntoUML and the formal axioms (outputs of the ontology capture and formalization phase). Generally, heavyweight ontologies must give rise to lightweight ones. If the implementation environment includes OWL and SWRL, the transformation from OntoUML to these languages proposed in [16] can be applied. In this case, we recommend using the Ontology Lightweight Editor (OLED) [16].

Ontology Implementation

The implementation phase regards implementing the ontology in the chosen operational language.

Ontology Testing

In SABiO, ontology test refers to dynamic verification and validation of the behavior of the operational ontology on a finite set of test cases, against the expected behavior regarding the competency questions. In this sense, SABiO's testing phase is competency questions-driven, and considers mainly black-box testing, although white-box testing can also be applied. A test case comprises an implementation of a competency question as a query in the chosen implementation environment (the specification to be tested), plus instantiation data from the fragment of the ontology being tested (input), and the expected result based on the considered instantiation (expected output). As Figure 5 shows, ontology testing in SABiO comprises three main activities.

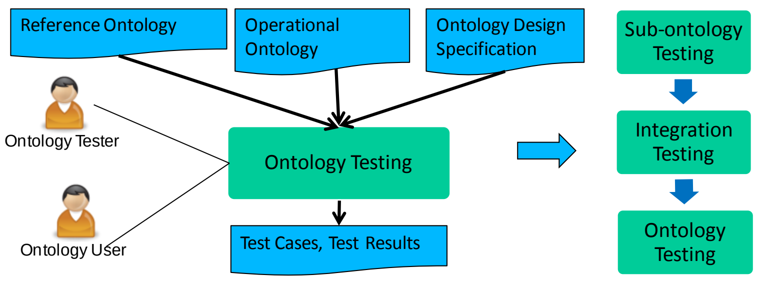

Fig. 5. The Ontology Test Phase.

Initially, test cases are run in the context of a sub-ontology. As sub-ontologies are integrated, ontology integration testing is performed. In this activity, the same test cases are re-run, but now considering the sub-ontologies already integrated. Finally, ontology testing is performed. In this activity, the test cases are run again in the context of the full ontology. During ontology testing, new test cases can be defined for testing technological non-functional requirements. Thus, ontology testing may include, among others, recovery and stress testing for web ontologies, and performance testing for checking inference performance. Validation testing can also be performed by using the operational ontology in actual software applications, according to the intended uses originally proposed to the ontology. Validation testing should be performed by ontology users.

4 SABiO's Support Processes

As shown in Figure 1, SABiO considers five main support processes: knowledge acquisition, documentation, configuration management, evaluation and reuse. These processes span the whole development process (or considerable parts of it).

Knowledge Acquisition Process

Knowledge acquisition occurs mainly in the initial phases of the ontology development process. Conventional methods and techniques for knowledge acquisition and for requirements elicitation applies, mainly those devoted to collaborative knowledge acquisition, such as brainstorming.

Domain experts are the main source for knowledge acquisition. Without the involvement of them, the ontology project can be impaired. Other important sources of knowledge are consolidated bibliographic material, such as classical books, international standards, glossaries, lexicons, classification schemes, and reference models. The methodological guidelines for reusing non-ontological resources described in [12] can be applied for selecting the most suitable resources to be used. The use of the knowledge elicited from these sources necessarily involves reengineering. In this context, ontological analysis based on a foundational ontology, as previously discussed, is an important technique to be applied. For an example of such case, see [17].

Reuse Process

Along the development process, there are many opportunities for reusing conceptualizations already established for the domain in hands. Typically there are for main sources for reuse: existing domain ontologies, core ontologies, foundational ontologies, and ontology patterns.

Existing ontologies for the domain can be reused, totally or partially. Especially in the case of partially reusing existing domain ontology, techniques of ontology merge, mapping and eventually reengineering apply [12]. Reuse of core ontologies (ontologies that provide a precise definition of structural knowledge in a specific field that spans across different application domains in this field [18]) are made mainly by means of specialization. Concepts and relations of the core ontology are extended by means of subtype relations in order to capture the more specific conceptualization regarding the domain in hands. New concepts, relations, properties and axioms can then be introduced in the domain ontology. Foundational ontologies can also be reused. In this case, reuse can be done by means of specializations (as in the case of core ontologies), but also by analogy. In reuse by analogy, foundational concepts and relations are not explicitly extended in the domain ontology, but implicitly used for deriving the structure of a portion of the domain ontology. In this sense, reuse by analogy is strongly related to the structuring of the concepts and relations in a domain ontology. A third way of reusing foundational ontologies is by using its foundations to analyze fragments of the domain ontology (ontological analysis). The use of OntoUML as modeling language for developing reference domain ontologies, and OntoClean [19] are examples of ways of performing ontological analysis.

Reuse can also be achieved by means of ontology patterns (OPs). In [20], Falbo et al. propose an OP classification that is closely related to patterns in Software Engineering, including four main types of OPs: conceptual OPs, architectural OPs, design OPs, and programming OPs (idioms). Ontology Conceptual Patterns are fragments of either foundational ontologies (Foundational OPs) or domain reference ontologies (Domain-related OPs). Foundational OPs are reusable fragments of foundational ontologies, while Domain-related OPs are reusable fragments extracted from reference domain ontologies. These patterns are to be used during the ontology conceptual modeling activity. Ontology Architectural Patterns are patterns that describe how to arrange an ontology (generally a large one) in terms of sub-ontologies or ontology modules, as well as patterns that deal with the modular architecture of an ontology network. These patterns can be used both during the purpose identification and requirements elicitation phase (ontology modularization activity), and in the ontology design phase (ontology architectural design activity). Since modularity is recognized as an important quality characteristic of good ontologies, we advocate for their use since the first stages of ontology development, for splitting the ontology into smaller parts, allowing tackling the problems one at a time. When applied in the architectural design activity, the purpose is to reorganize the ontology modules for addressing technological aspects, in special by taking non-functional requirements into account. Ontology Design Patterns (ODPs) are to be used during the ontology detailed design activity. There are two main types of ODPs [20]: logical and reasoning ODPs. Reasoning ODPs addresses specific design problems related to improving reasoning with ontologies (and qualities related to reasoning, such as computational tractability, decidability and reasoning performance). Logical ODPs, in turn, regards problems related to the expressivity of the formalism to be used in ontology implementation. They help to solve design problems that appear when the primitives of the implementation language do not directly support certain logical constructs. Logical ODPs are extremely important for ontology design, since most languages for coding operational ontologies are not focused on representation adequacy. We should highlight that many patterns that address reasoning and logical problems are, in fact, Ontology Idioms (or Ontology Programming Patterns), since they describe how to solve problems related to reasoning or to the expressivity of a specific language (e.g., OWL). Ontology idioms can be reused both during ontology detail design and ontology implementation activities [20]. Finally, during ontology test, test cases can be reused.

Documentation Process

Results from the ontology development process must be documented. Moreover, results from some support processes, such as evaluation, must also be documented. So, documentation is a process that has to occur in parallel with the others.

In order to ensure uniformity in the ontology projects, it is useful to define the basic set of documents to be produced in all projects. Templates for the main documents must also be defined. SABiO suggests three main documents to be produced as the documentation of an ontology project. As a result of the first two phases of the development process, a Reference Ontology Specification should be produced. The Operational Ontology Design Specification documents the aspects related to the design phase, including: information related to the implementation environment, technical non-functional requirements, ontology architecture, and main design decisions. For documenting the operational ontology (i.e., the source code implementing the ontology), SABiO suggests that the organization defines naming conventions and rules for commenting the ontology code. Finally, for documenting the ontology test phase, a test document shall be produced, including test cases and test results.

Due to the highly collaborative nature of ontology projects, we suggest the use of wikis for documentation, especially for documenting reference ontologies.

Configuration Management Process

The main documents proposed by SABiO, as well as the source code of the operational ontologies must have their configuration managed. Thus, once approved, they must be submitted to the Configuration Management, where they will be controlled at least concerning changes, versions, and delivery.

Evaluation Process

Although shown as an activity of the ontology development process, ontology testing is, in fact, an evaluation activity. Ontology testing consists of the dynamic evaluation (i.e. running code) of the behavior of an operational ontology on a finite set of test cases, against the expected behavior. On the other hand, several other static evaluation activities (those not involving running the code) must be performed during the ontology development process. Those activities are performed by means of technical reviews. In the context of ontology engineering, the purpose of a technical review is to evaluate an intermediary work product of the ontology development process to determine its suitability for its intended use. The results should confirm (or not) that the work product meets the specifications, and adheres to standards.

SABiO's evaluation process comprises two main perspectives:

- Ontology Verification: aims to ensure that the ontology is being built correctly, in the sense that the output artifacts of an activity meet the specifications imposed on them in previous activities.
- Ontology Validation: aims to ensure that the right ontology is being built, that is, the ontology fulfills its specific intended purpose.

Concerning ontology verification, the focus is on two main aspects: (i) Are the ontology quality criteria (competency, clarity, coherence, consistency, minimal ontological commitment, etc.) being met? (ii) Are the established standards (e.g., document templates) and processes being correctly applied? Regarding the quality criteria, one stands out: competency. As aforementioned, SABiO ontology testing phase is driven by the competency questions. However, this is not enough. During the ontology capture and formalization phase, the reference ontology should also be verified whether it meets the requirements. This can be done by means of expert judgment. A table indicating which ontology elements (concepts, relations, and axioms) are able to answer each competency question should be built. The purpose of this table goes further verification. It can also be used as a traceability tool, supporting change management.

Concerning ontology validation, the participation of domain experts and ontology users is essential. Ontology users have to evaluate whether the ontology is adequate for their intended uses. For validating the reference ontology with domain experts, the use of a graphical notation is very important, since generally they are not able to read formal specifications. Besides expert judgment, another relatively easy way to validate a reference ontology is by means of instantiation. The reference ontology should be able to represent real world situations. Thus, an instantiation table should be produced from real world situations.

There are many evaluation techniques proposed in the literature. Several of them can also be applied in a complementary way to the ones mentioned above. See a good list of them in [12, 21].

5 Related Work

There are many ontology methods proposed in the literature. However, as pointed by Corcho et al. [22], none of the existing approaches is fully mature if compared to software engineering methodologies. Although we clearly have advanced in the last years, as shown by the findings of the survey performed by Simperl et al. in 2009 [23], there are still room for improvements, and Software Engineering plays an important role in this scenario.

Among the various existing methods, we selected the NeOn Methodology for Ontology Engineering [12] to compare to SABiO, since it is the result of the largest project devoted to Ontology Engineering already performed, involving 14 partners and during 4 years. The NeOn Methodology focuses on the engineering of ontology networks, and does not prescribe a rigid workflow, but instead it suggests a variety of pathways for developing ontologies. Like SABiO, the NeOn Methodology includes several processes strongly linked to the development process, organized in 9 scenarios. Some processes, namely Ontology Development, Evaluation (Verification and Validation), Reuse, Documentation, Configuration Management and Knowledge Acquisition, are common to both methods. The NeOn Methodology considers also other aspects not explicitly consider in SABiO, such as Ontology Localization, Management, Alignment, and Consensus Reaching Process. In this sense, SABiO can be enriched by applying several ideas from the NeOn Methodology. Aspects related to collaborative development, addressed by methods such as DILIGENT [24], are not adequately addressed in SABiO, and thus practices proposed by this (and other more collaborative) methods should be introduced in SABiO.

The striking features of SABiO when compared to the NeOn Methodology (and also other ontology engineering methods) are: (i) The recognition that both reference domain ontologies and operational ontologies are useful in themselves. SABiO supports the development of both types of domain ontologies. This distinction leads to the perception of the importance of a design phase (as largely recognized in Software Engineering) in the ontology development process. Moreover, pattern-oriented reuse in SABiO is also guided by this distinction. Other ontology engineering methods do not prescribe a design phase in the sense used in Software Engineering. (ii) SABiO recognizes the importance of the use of foundational ontologies in the development of domain ontologies, and proposes the use of OntoUML during ontology capture, as well as ontological analysis techniques. These two main features of SABiO have impact in activities of other processes. For instance, Knowledge Acquisition and Reuse Processes are strongly influenced by them. Thus, there are differences in the approaches proposed by SABiO and the NeOn Methodology regarding these processes.

6 Final Considerations

This paper presented the new version of SABiO, an ontology engineering method. This version was firstly used for developing ROoST (Reference Ontology on Software Testing) [25], and an operational ontology derived from it. In this project, in general, SABiO 2.0 worked well, especially the reuse process, by the reuse of domain-related patterns organized as an ontology pattern language. It is important to highlight also the great support provided by the Ontology Lightweight Editor (OLED) [16] for automatically generating an OWL ontology (operational ontology) from the OntoUML models.

We know that we need more feedback in order to improve the current practices of SABiO. Since SABiO is now being applied in the development of other domain ontologies, we intend to collect new feedback from these projects, in order to better evaluate the current version of the method.

Acknowledgments. This research is funded by the Brazilian Research Funding Agency CNPq (Process Number 485368/2013-7).

References

1. Falbo, R.A., Menezes, C.S., Rocha, A.R.C. A Systematic Approach for Building Ontologies. Proceedings of the 6th Ibero-American Conference on Artificial Intelligence, Lisbon, Portugal, Lecture Notes in Computer Science, vol. 1484, 1998.
2. Falbo, R.A., Bertollo, G., A software process ontology as a common vocabulary about software processes. International Journal of Business Process Integration and Management (IJBPIM), v. 4, p. 239-250, 2009.
3. Oliveira, K.M., Zlot, F., Rocha, A.R., Travassos, G.H., Galotta, C., Menezes, C.S., Domain-oriented software development environment, Journal of Systems and Software, v. 72, issue 2, pp. 145-161, July 2004.
4. Falbo, R. A., Experiences in Using a Method for Building Domain Ontologies, International Workshop on Ontology in Action, Proceedings of the 16th International Conference on Software Engineering and Knowledge Engineering, 2004 , pg 474-477.
5. Guizzardi, G., On Ontology, ontologies, Conceptualizations, Modeling Languages and (Meta)Models, In O. Vasilecas, et al. (Org.). *Frontiers in Artificial Intelligence and Applications, Databases and Information Systems IV*. IOS Press, Amsterdam, 2007.
6. Fielding, J.M.; Simon, J.; Ceusters, W.; Smith, B., Ontological Theory for Ontology Engineering. In: Proceedings of 9th International Conference on the Principles of Knowledge Representation and Reasoning, Whistler, Canada, 2004.
7. Fettke, P., Loos, P., Ontological Evaluation of Reference Models Using the Bunge-Wand-Weber Model. Proceedings of the 2003 Americas Conference on Information Systems, Florida, USA, 2003.

8. Shanks, G., Tansley, E., Weber, R., Using Ontology to Validate Conceptual Models. Communications of the ACM, v. 46, issue 10, 2003, pp. 85-89.
9. Smith, B., Against Idiosyncrasy in Ontology Development. In Proceedings of the 4th International Conference on Formal Ontology and Information Systems, Baltimore, 2006.
10. Bringuente, A.C., Falbo, R.A., Guizzardi, G., Using a Foundational Ontology for Reengineering a Software Process Ontology. Journal of Information and Data Management, v. 2, pp. 511-526, 2011.
11. Grüninger, M., Fox, M.S., Methodology for the Design and Evaluation of Ontologies. Workshop on Basic Ontological Issues in Knowledge Sharing, 1995.
12. Suarez-Figueroa, M. C., Gomez-Perez, A., Motta, E., Gangemi, A., Ontology Engineering in a Networked World. Springer, Berlin, 2012.
13. d'Aquin, M., Schlicht, A., Stuckenschmidt, H., Sabou, M., Criteria and Evaluation for Ontology Modularization Techniques, In: Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization, Springer-Verlag, 2009. pp. 67–89.
14. Guizzardi, G., *Ontological Foundations for Structural Conceptual Models*, Universal Press, 2005.
15. Guizzardi, G., Graças, A.P., Guizzardi, R.S.S., Design Patterns and Inductive Modeling Rules to Support the Construction of Ontologically Well-Founded Conceptual Models in OntoUML, 3rd International Workshop on Ontology-Driven Information Systems, 2011.
16. Barcelos, P.P., Santos, V.A., Silva, F.B., Monteiro, M.E., Garcia, A.S., An Automated Transformation from OntoUML to OWL and SWRL. In: Proceedings of the 6th Seminar on Ontology Research in Brazil (ONTOBRAS 2013), Belo Horizonte, Brazil, 2013.
17. Ruy, F.B., Falbo, R. A., Barcellos, M.P., Guizzardi, G., An Ontological Analysis of the ISO/IEC 24744 Metamodel. 8th International Conference on Formal Ontology in Information Systems (FOIS'2014), Rio de Janeiro, Brazil, 2014.
18. Scherp, A., Saathoff, C., Franz, T., Staab, S.: Designing core ontologies. Applied Ontology, vol. 6, n. 3, 2011, pp. 177-221.
19. Guarino, N., Welty, C., Evaluating Ontological Decisions with OntoClean. Communications of the ACM. 45(2), 2002, pp. 61-65.
20. Falbo, R. A., Guizzardi, G., Gangemi, A., Presutti, V., Ontology Patterns: Clarifying Concepts and Terminology. Proceedings of the 4th Workshop on Ontology Patterns (WOP2013), Sydney, Australia, 2013.
21. Obrst, L., Ashpole, B., Ceusters, W., Mani, I., Ray, S., Smith, B., The Evaluation of Ontologies - Toward Improved Semantic Interoperability, In: Semantic Web: Revolutionizing Knowledge Discovery in the Life Sciences, Baker, C.J.O., Cheung, K-H. (eds), Springer, 2007.
22. Corcho, O.; Fernández-López, M.; Gómez-Pérez, A. Methodologies, tools and languages for building ontologies. Where is their meeting point? Data & knowledge engineering 46 (1), 2003, pp. 41-64.
23. Simperl, E., Mochol, M., Burger, T., Achieving Maturity: the State of Practice in Ontology Engineering in 2009, International Journal of Computer Science and Applications, Vol. 7, No. 1, 2010, pp. 45 - 65.
24. Pinto, H. S., Tempich, C., Staab, S., Ontology engineering and evolution in a distributed world using DILIGENT. In: Handbook on Ontologies. 2nd edition, Springer, 2009.
25. Souza, E. F., Falbo, R. A., Vijaykumar, N. L. Using Ontology Patterns for Building a Reference Software Testing Ontology. In: 8th International Workshop on Vocabularies, Ontologies and Rules for the Enterprise and Beyond, Vancouver, Canada, 2013.