Requirements for Logical Modules

Frank Loebe

Research Group Ontologies in Medicine (Onto-Med)
Institute of Medical Informatics, Statistics and Epidemiology (IMISE)
University of Leipzig, Germany

frank.loebe@imise.uni-leipzig.de
http://www.onto-med.de

Abstract. Modularization for ontologies and thus for logical theories is receiving increasing interest, but a clear presentation of requirements for solutions is still missing. This paper presents a collection of such requirements. Some of these are derived as desiderata in the context of applying modularization to the axiomatization of a top-level ontology, whereas others are determined by analyzing the notion of module in software engineering. Given these requirements, their relationship to current proposals for modularizing ontologies is briefly discussed, with a focus on our application. In a wider context, the paper represents an initial step towards a notion of module for logical languages, which is applicable to the creation and maintenance of large axiomatizations.

1 Introduction

Expedited by the Semantic Web in general, and the Web Ontology Language (OWL) [31] in particular, the use of logical formalisms has become widespread over recent years, even if this may be hidden for many users.[1] The new dimensions of usage of these formalisms, e.g., as concerns the size of ontologies/theories, creates a number of novel problems which have to be tackled also in the formal framework. Mapping and merging ontologies was recognized as a problem of that kind some years ago, and it has seen some progress already [19], while the problem of modularization is still in its beginning, with a limited number of publications addressing this problem explicitly. One might even say that a clear definition of the problem is missing, apart from the rough idea of interrelating large ontologies with a number of smaller components, in order to keep the overall system "better manageable".

Concerning modularization in the context of OWL, the lack of a sophisticated notion of module has already been criticized with its advent, because of the simplicity of means which it provides for combining OWL ontologies. The `owl:imports` statement is the only available element of OWL to achieve a

[1] In all cases of referring to OWL, we actually refer to OWL-DL which corresponds closely to the description logic $\mathcal{SHOIN}(\mathcal{D})$ (see [9, p. 27–32] for a clear presentation of minor differences between these two, which are irrelevant for this paper).

syntax-level splitting of ontologies. The semantics of this statement corresponds to a simple union of the axioms of the importing ontology and the imported ones, which is insufficient for supporting modularized ontologies.

The specific background for our interest in logical modularization is given by the development of the top-level ontology *General Formal Ontology* (GFO) [18], which is to be presented by means of axiomatic theories in several logical languages. For reasons of expressiveness, our primary axiomatization is based on first order logic, but we also provide an OWL version. Independently of which particular logical formalism is chosen, such axiomatizations will comprise at least hundreds of non-uniform axioms, i.e., axioms which do not follow one or a few common schemes, like simple subsumption statements. We will argue that the task of developing and maintaining these axiomatizations can be compared to engineering software artifacts. Currently, we are not aware of any formalism which has been equipped with a notion of module which supports dealing with such theories in an appropriate manner. This paper shall clarify our understanding of "appropriate" by introducing and discussing requirements for modularization.

Among other purposes, this can support the evaluation of existing approaches. Indeed, some extensions for OWL and other logical languages have been proposed which offer ways for modularizing theories (for example, [12, 7, 3]). However, in most cases these have been derived from approaches developed with different motivations. Their characteristics should therefore be examined against modularization requirements, which we do briefly for three approaches.

The overall structure of the paper is as follows. Firstly, we collect our desiderata for modules, which arise directly from working with logical formalisms, in section 2, preceded by a short introduction to the application setting. In section 3, further requirements are obtained by considering the notion of module in software engineering. Section 4 presents a brief analysis of the properties of current proposals in relation to our requirements, before the paper concludes with the fifth section.

A note on the formal setting is due. Much of what follows is independent of a particular logic, like first order logic (FOL) or any specific description logic (DL). We refer to a logic $\mathcal{L}$ as a triple $\mathcal{L} = (L, M, \models)$ of a language L, understood as a set of sentences[2], a set of model structures M, and a satisfiability relation $\models \subseteq M \times L$ between model structures and sentences. By standard definitions, the latter generalizes to model and consequence relations between models, model classes, and theories, all of which are likewise denoted by $\models$. $Cn(T)$ denotes the deductive closure of T. Grammatical definitions of a language L usually distinguish sentence constructions and a vocabulary/signature V. We sometimes speak of the same language even for different vocabularies, and $L(V)$ denotes all sentences which can be built from V in a given grammatical framework. Any subset $T \subseteq L(V)$ is called a theory. The vocabulary of T is denoted by $V(T)$, the corresponding language is $L(T) =_{df} L(V(T))$.

[2] We are not interested in formulas with free variables in the case of a language with variables.

2 Requirements From an Application Perspective

2.1 General Setting: Distinguishing Basic and Complex Modules

The current axiomatization of the General Formal Ontology (GFO) comprises about three hundred sentences in FOL, circa one hundred of which are explicit definitions. These are roughly organized into axiomatic fragments with respect to several topics in GFO, like Time, Space, Part-of Relations, etc. This organization is useful for analyzing the resulting theories with respect to their logical properties, most prominently consistency and completeness. However, statements about the system of these fragments, seen as a single theory, are hard to make in the case of FOL, because the vocabularies of these fragments overlap.

In the current situation, one may view these fragments as modules, which should be combined in a sensible manner. Note that in general, given some notion of module, two kinds of usage become available, namely *composition* and *decomposition*. Composition refers to studying the relation between a module and a system built from several modules, whereas decomposition starts with a theory or system and tries to determine an "appropriate" collection of modules. The latter is often called "partitioning" in recent approaches [3, 12]. Currently, our own focus is compositional rather than decompositional.

Another view on GFO is as a ready-made component for the development of domain ontologies, so that GFO as a whole appears as a module of domain ontologies. Under this perspective, GFO should behave like a logical theory in order to fit formalizations of domain ontologies.

According to these views one can distinguish two kinds of modules (or even logics). Modules in the first sense are called *basic modules*, and their core component is some theory $T \subseteq L$, usually provided as a finite axiomatization in a language L. Given a number of basic modules $(T_i)_{i \in I}$ (where I is an arbitrary index set), these can be combined into a *system* S. Such a system may serve itself as a *complex module* for a larger system. If the applied combination operation departs from set-theoretical union, this yields potentially different logics for basic and complex modules. Let us assume that all basic modules T_i adhere to a common logic $\mathcal{L} = (L, M, \models^{bas})$, where $L = L\left(\bigcup_{i \in I} V(T_i)\right)$. Even then the composition operation for creating S out of the T_i may form a new kind of entailment and hence a different logic for S, $\mathcal{L}' = (L', M', \models^{comp})$, with respect to which S represents a theory $S \subseteq L' \supseteq L$.

2.2 Requirements

With this general setting in mind, we formulate and discuss the following requirements for logical modules. At this stage, we do not impose any pre-established constraints on ourselves with respect to feasibility, computability, or the question of whether a single, formal notion of module can satisfy all of them. In this sense, the following is to be seen as a collection of desiderata, rather than a set of requirements which must be met necessarily.

Inclusion/ Local Correctness and Completeness It appears natural that a system S composed of modules should provide the information contained *explicitly* in its modules. If S is composed of basic modules $(T_i)_{i \in I}$, this requires $S \models^{comp} \phi$ for every $\phi \in T_i$, for all $i \in I$. A strengthening of this requirement would also include *implicit* information, replacing $\phi \in T_i$ above by $T_i \models^{bas} \phi$. This is called *local correctness* in [12], which is traced back to [14]. The "inverse" notion is *local completeness* of a module T_i with respect to S. It requires for every ψ with $S \models^{comp} \psi$ and $\psi \in L(T_i)$ that it follows already from T_i, $T_i \models^{bas} \psi$.

Compositionality If certain logical properties are proved for modules, there should be general means to derive these properties for the overall system. For example, the consistency of a set of modules may immediately result in the consistency of their combination, i.e., due to the combination operation defined for modules. Compositionality would be very valuable for undecidable logics such as FOL, because one could then concentrate on attempts to prove logical properties for much smaller theories. Decidable logics allow for computing consistency in principle, however, compositionality would be helpful in practice, because at present, large ontologies can still take reasoners far beyond their capabilities.

Classically Closed Composition A system S composed of modules, viewed as a deductive system, should be deductively closed with regard to classical logic. The rationale behind this is to feed information from S into classical systems, i.e., such which are based on (fragments of) classical logic. This includes OWL and many description logics, as well as rule languages which are currently being adapted to the Semantic Web.

For instance, S should satisfy the deduction theorem (in case of a classical logic being used for its basic modules; or an equivalent for other kinds of logics). If it does not, assume that $S \models^{comp} \phi$ and $S \models^{comp} \phi \rightarrow \psi$, but $S \not\models^{comp} \psi$. A classical system querying S and receiving ϕ and $\phi \rightarrow \psi$ could then derive ψ, which is in conflict with S. The requirement means to avoid such situations for arbitrary classical deductions.

Directionality Composition should allow for a directed information flow among modules, such that a module may use another without having an impact on it. This requirement is motivated by the second use case from above, where a domain ontology is based on a top-level ontology. Of course, one would not want effects from the domain ontology to be transferred to the top-level ontology (in the worst case, causing inconsistency where there was none). Formally, if T_1 and T_2 are components of S and composition assures a directed information flow from T_1 to T_2 within S, then T_2 may take sentences from T_1 into account, but not vice versa. A very similar requirement is stated in [16].

Comprehensibility A module should remain "comprehensible", with the purpose of supporting maintainability. Two options in order to achieve this for basic modules are (a) to have a rather small vocabulary and small set of axioms of arbitrary form, or (b) to have a possibly large, structured vocabulary equipped with only simple axioms, which furthermore follow one or a few common schemes.

For complex modules, comprehensibility should derive from the way they are composed of basic modules.

Stability A system composed of modules should remain stable if a single module evolves, or loosely related modules are added. Especially, the system structure should remain stable, i.e., the addition of a module does not change the previously established structure among other modules. Furthermore, side effects of evolution should be reduced to a minimum. This criterion has also been proposed in [25].

These requirements arise naturally in our application setting. We are aware of the fact that most of them have been mentioned in different circumstances, as indicated according to our knowledge. Distributed First Order Logic (DFOL) [16] is motivated with a list of properties which has the strongest overlap with ours we are aware of (see also section 4.3).

Concerning the interrelations among these requirements, one can already notice that there are some which are likely to prevent attempts to satisfy all of them. For example, directionality is a requirement which interferes with local correctness and completeness as well as classically closed composition. For example, let S answer queries in $L(T_1)$ only by means of its module T_1, but queries in $L(T_2) \supset L(T_1)$ by means of $T_1 \cup T_2$. This can easily create a situation where S loses classically closed composition, e.g., by no longer satisfying the deduction theorem. Local correctness and completeness are problematic even without other requirements. Both conditions are very restrictive, and the question arises whether a non-trivial composition operation can be found, which is different from set-theoretical union but satisfies local correctness and completeness.

Based on these requirements, one could now start to develop a notion of module. However, so far this is rather an ad hoc collection of requirements. In order to put the overall approach onto firmer ground, we will next consider the notion of module in software engineering.

3 Modules in Software Engineering

The reason for considering software engineering as an appropriate source relies on the assumption that the development of large axiomatizations in some aspects resembles software engineering, where we have come to this view based on our current experiences. Moreover, modules and modularization have proved as concepts of much success in software engineering. Note that a broad reading of module applies here, because several specialized fields have created their own basic notions which contain the general idea of modularization, e.g., "object" in object-oriented approaches, or "component" in component-based software development.

3.1 Basic Notions: Interface and Service

Starting with some general definitions, Siedersleben defines a module as "a closed software unit with a well-defined interface", which forms "the smallest unit of

software design" [28, p. 97, own translation]. He refers to [24] in stating that data abstraction is the primary means to split systems into modules, where the distinction between *interface* and an implementational *body* of a module forms the central idea. An interface defines the *services* (or tasks) of a module.

Given this general characterization, *interface* and *service* form two central elements of modules in software engineering, which should be migrated to a logical understanding. The immediate services a logical module could provide are reasoning services. For example, [4, primarily ch. 1] mentions some reasoning types for description logics, like concept inclusion, concept satisfiability, or consistency checking. However, in this generality, it seems hard to fix an analogue for the notion of interface. In the sequel we therefore focus on answering entailment queries, i.e., queries of the form $T \models \phi$. This is the most relevant kind of query for our application purpose, and several other query types can be reduced to it (in many logics).

For the entailment service, a number of options for defining an interface of a theory $T \subseteq L$ appear. Two general approaches would be to view an interface either as a query language $Q \subseteq L$, or, considering the set of answers, again as a theory. The query language approach allows for restrictions of both, the language grammar as well as the vocabulary. A special case of this is to just restrict the language by means of the vocabulary, i.e., to use a subset $V' \subseteq V(T)$ of the given vocabulary as interface specification. The theory-based understanding of interface is even more general. It can integrate the query language approach by defining $T' =_{df} Cn(T) \cap Q$, but it may also allow for transformations of the formulas of T. Currently, for reasons of comprehensibility and for simplicity of specification, we advocate the query language view of what a logical interface should be. Independent of this choice, a theory may provide several such interfaces, like in software engineering.

The notion of a *body* degenerates to some extent in the declarative setting, where a dynamics like program execution is missing or at least not more informative than the semantics. In the query language view, the overall theory appears as the body, which at least allows for information hiding (see below).

3.2 Key Features of Software Engineering Modules

There are some key features of modules in software engineering (cf., for instance, [5, 13, 1, 26]) which can now be considered with respect to their logical counterparts. (*Italics* mark software engineering formulations.)

Structured into explicit interface and body *Modules define explicit interfaces over which they communicate and the interface is the only way to communicate. Apart from that the module comprises a body which is accessed only internally by a module.*

This feature has already been covered by the above discussion of interpretations of "interface". In the query language approach to interfaces, communication between modules means that one module queries another in a query language provided by the latter. This is very similar to the message passing metaphor between theories employed in [3], see section 4.2.

Information hiding, encapsulation and visibility *Internal information and information handling is hidden for other modules. Therefore, a module provides encapsulated functionality and / or data.*

It depends on the relation between the language $L(T)$ of a module T and the query language Q, serving as interface, to what extent information hiding and visibility are realized. A problem with this feature is that it is not clear, in a declarative setting, why some aspects of a theory should be generally hidden. That is, why should certain vocabulary elements appear in *no* interface for T? With respect to a single interface, it is useful to think of information hiding and encapsulation. For example, given a top-level ontology T_{TLO}, one may be interested to use just the theory of time built into it, so let $T_{TLO} \models T_{Time}$. For this purpose, T_{TLO} may provide $L(T_{Time})$ as a time interface. With our stability criterion in mind, changes to the top-level ontology must be transferred to some domain ontology which uses T_{Time} only then if they affect the service accessible via the time interface.

Let us now consider the case of multiple interfaces. In general, a theory T with a number of logical interfaces defined by means of query languages $Q_1, \ldots, Q_n$ yields a family of theories, namely $(Cn(T) \cap Q_i)_{1 \leq i \leq n}$. Note that these theories cannot by default be seen as modules of T themselves, because two such theories may be related in arbitrary ways with respect to entailment, in dependence of the relations among the Q_i. In software engineering, this freedom is constrained for modules by means of the next criterion.

Independence *The tasks a module has in the overall architecture are (to an appropriate degree) independent from those of other modules.*

With respect to the entailment service, which means to answer queries in a specific language, we interpret this in such a way that different modules answer questions with respect to different "topics", which is naturally reflected by the use of different languages. Starting with the grammar level, it is not obvious to us how differences among the grammar would capture this criterion well. In contrast, different vocabularies can reasonably be seen to refer to different topics. A clear but strong notion of independence would require different vocabularies to be disjoint. However, less restrictive interpretations will be needed to achieve communication among modules.

It is important to note two axes of vocabulary differences, regarding the ontological relations of the intended meaning of the vocabulary elements. Horizontally (the term borrows from a hierarchical image of taxonomies), two vocabularies may be concerned with distinct concepts in the sense that those concepts do not share any instances. Trees, walks, and numbers are distinct in this reading. In contrast, the vertical axis relates to the generality of notions. For instance, vocabularies of top-level and domain ontologies are usually disjoint, but top-level concepts classify the same entities referred to by domain entities. For predicate logics and description logics, these types of differences interact with the corresponding model structures, if these languages are used in a way which allows for an ontological interpretation of relations between entities in model structures. For example, predication may reflect instantiation. OWL, in particular, is

exposed to this problem because of its very nature as an ontology language. Accordingly, modularization approaches based on or with specific effects on model structures should be examined for implications on the contents expressed ontologically (see also section 4.1). In general, a solution for modularization which handles both types of differences appears to be preferable over another dealing with only one of them.

Due to interfaces as query languages, we have provided a language-oriented interpretation of independence for modules. Another approach could consider reasoning inside modules to be independent, rather than the languages used. However, complete independence would then mean to not exchange any formulas. Otherwise, the communication of formulas from one module to another creates interdependences of reasoning. Hence, we claim that a better analogue for this is *interaction* among modules rather than *independence*. Of course, interaction is likewise observed in software engineering, and constrained for modules by the subsequent characteristic.

Cohesion vs. coupling *Modules should behave differently regarding cohesion and coupling. Cohesion refers to interconnections within a module, whereas coupling names interconnection among modules. Cohesion should be maximized in modules, coupling minimized.*

Sentences in a logical theory may have an impact concerning entailments on each other, without any structural restrictions. This can be understood as cohesion. Combining theories by set-theoretical union yields a theory whose sentences are in the same way closely interconnected. Thus, this form of composition does not adopt the distinction between cohesion and coupling. Compared to our requirements above, directionality can be seen as a means to achieve a notion of coupling.

Substitutability without side-effects *Ideally, a module can be exchanged by a module of equal character (i.e. which supports the same functionality) without unexpected side-effects concerning coupled modules and the overall system. In practice, usually some restrictions apply to the ideal case.*

Given the query-language view of interface, "equal character" would literally mean that a substitute module T' provides the same answers to queries in Q as the original module T. Accordingly, this criterion can be met perfectly even by standard theories, yet for a seemingly uninteresting special case. Logically equivalent axiomatizations, i.e., given $Cn(T') \cap Q = Cn(T) \cap Q$, would form distinct modules which are perfectly interchangeable without any side-effects. However, due to the fact that composition may yield a different form of entailment, this depends on the inclusion requirements. For example, the above equation may fail for a system comprising T, and later T', if only the inclusion of explicit information is provided.

Another relevant case regards multiple interfaces: one may refer "equal character" to a single interface. Returning to the time-example, T_{Time} may remain constant while T_{TLO} is changed in some other respects. Again, this connects to our stability requirement.

There are further features, like *readability*, *reuse status*, or *separate editability*, which do not contribute as much to an understanding of "module" as those we have discussed so far. What can be observed is that there is hardly any overlap between our ad hoc requirements and those transferred from software engineering. By no means should this be interpreted negatively for the latter. Instead, implicit assumptions could be explicated by this approach. With a more profound set of requirements available, we can now start to evaluate the adequacy of related work for our purposes.

4 Discussion of Existing Proposals

Modularization of logical theories is a rather young research interest, but it is related to a number of fields which seem to provide good starting points. The most direct approaches to modularization we are aware of and which moreover fit the context of logic-based ontologies are that of Bernardo Cuenca Grau and colleagues [9, 12, 11, 10], which we will call the "semantic encapsulation" approach (based on [9]); "partition-based reasoning" of Eyal Amir et al. [3, 22, 2], and the "distributed logic" approach developed by Luciano Serafini and colleagues [7, 16, 6]. For each of these, we provide a short contextual placement and sketch their main ideas, followed by a selective evaluation against the presented requirements. The latter is driven by the question of which approach may suit a structured axiomatization of GFO. Finally, further related work is briefly mentioned.

4.1 Semantic Encapsulation

This approach was initially based on ε-connections [20], which is a method of combining certain kinds of logics, and thus belongs to the field of combining, fibring, and fusing logics. For ε-connections, there is the general result that the combination of decidable logics yields a possibly more expressive, but still decidable formalism.

The semantic encapsulation approach started as an application of ε-connections to combining ontologies based on description logics [11], which is motivated by the idea of an integrated use of independently developed OWL ontologies on the web. Recently, it has been extended in a way which departs from ε-connections [12].

Nevertheless, it still shares a central feature with the ε-connections method, for a certain class of $\mathcal{SHOIQ}$ theories: [12] introduces a partitioning algorithm for such a theory T in which the resulting set of partitions "reflects" a group of specific models of T. These models have a partitioned domain themselves, and each single partition of T is evaluated in a partition of the domain of the model. It is this property which may justify the name "semantic encapsulation". Note that each single partition has the property that, using OWL terminology, two concepts which entail another always belong to the same partition. Modules in the sense of [12] can be computed based on unions of such partitions. Thus they "inherit" this property.

This property has an effect which we find problematic, all the more against our background of developing a top-level ontology: given a domain ontology which can be modularized according to [12], the introduction of a top-level ontology on top of that domain ontology would cause all modules to collapse into one. This has practically been observed in tests with the GALEN ontology, which is equipped with its own top-level [9]. In terms of the presented requirements, modules in [12] do not satisfy stability and independence. The latter is due to the fact that modules are defined with respect to concepts in the initial ontology, and modules of two concepts can overlap arbitrarily. Due to this problem and the fact that directionality is not satisfied either, semantic encapsulation is not applicable to the second of our use cases, where a top-level ontology should serve as a component of domain ontologies.

Nevertheless, [12] may be useful for the analysis of monolithic fragments of a top-level ontology, in order to guide modularization. Here, it is advantageous that semantic encapsulation satisfies local correctness and completeness, because this requirement is desirable with respect to the internals of a top-level ontology.

4.2 Partition-based reasoning

The goals of this approach are to support reasoning over multiple theories with overlapping content and vocabularies, as well as the improvement of the efficiency of reasoning by means of partitions. [3] is the latest introduction to the theoretical framework, which is based on earlier publications [22, 2]. The results apply to propositional logic as well as FOL.

Neglecting the algorithmic details of [3], the main idea of this approach is to use a message passing metaphor from the object-oriented paradigm in software engineering for reasoning over a partitioning of some theory. More precisely, given a theory T and a partitioning $(T_i)_{1 \leq i \leq n}$ of T, message passing algorithms are specified which employ "standard" reasoning within each T_i, but use message passing between certain T_i and T_k, i.e., if $L(T_i) \cap L(T_k) \neq \emptyset$, they can exchange formulas in $L(T_i) \cap L(T_k)$. The specified algorithms are proved to be sound and complete with respect to classical reasoning over T, where these results are heavily based on Craig interpolation [8, Lemma 1].

Moreover, a decomposition algorithm is presented in [3] which tries to minimize three parameters, ranked by importance. With respect to a single partition T_i, firstly, this is the number of symbols which are exchanged with other partitions and, secondly, the number of unshared symbols within T_i. With respect to the overall partitioning, the number of partitions should be kept to a minimum, which is the third and least enforced parameter.

Taking partitions as modules and set-theoretical union as the composition operation, this approach satisfies inclusion of explicit information and classically closed composition. The intersection languages for message passing correspond properly to a specific type of interface specification. Moreover, the decomposition approach minimizes coupling, module size, and the number of modules, which is adequate also according to our setting. However, having union as composition is rather weak and prevents directionality. Therefore, utility for our

purposes is primarily seen in the decomposition algorithm, similarly to semantic encapsulation.

4.3 Distributed Logics

This branch comprises works of Luciano Serafini et al. on developing *Distributed First Order Logic* (DFOL) [16], *Distributed Description Logics* (DDL) [6] and a contextualized form of OWL, called C-OWL [7]. Moreover, these approaches have been developed based on work in the contextual reasoning community, more precisely based on Local Model Semantics (LMS) and Multi-Context Systems (MCS), cf. [15, 27].

The major idea of LMS/MCS, which conveys to the more recent approaches, is to have a collection of theories (possibly in different logical systems) each of which is first of all interpreted *locally*, i.e., with respect to the semantics of its associated logic. On top of this, *bridge rules* specify interconnections among these theories, i.e., given T_1 and T_2, a bridge rule is a pair (ϕ, ψ) with $\phi \in L(T_1)$, $\psi \in L(T_2)$, which states that $T_1 \models \phi$ allows one to assume ψ in T_2.

The distributed logic approach is compositional rather than decompositional, and it is equipped with a non-trivial composition operation, because bridge rules provide directionality. Therefore, it appears to match some of our needs better than the previous approaches, which are oriented towards decomposition. This impression is further augmented by considering the list of properties required for DFOL in [16]. However, some properties are problematic, e.g. "hypothetical reasoning across subsystems" is prevented, which contradicts our requirement of classically closed composition. Further, local correctness and completeness is not guaranteed, which would be more relevant for certain cases than directionality. On a more technical level, the approach of keeping everything local and independent (by disjoint languages for modules) causes some inconvenience (at least), due to the fact that the representation of overlapping vocabularies implies the addition of many bridge rules.

4.4 Further Related Work

Certainly the most prominent approach aiming at reasoning and structuring theories, which has not been discussed here, is the solution provided for structuring the CYC knowledge base [21] in terms of microtheories. The theoretical solution has primarily been developed by Rahmanathan Guha [17], also in the field of contextual reasoning. It remains a future task to analyze this in detail.

Another line of research provides less logic-oriented approaches for decomposing ontologies, among them [29, 30]. Moreover, Alan Rector also tackles the problem, yet at a different, more methodological end. [25] presents methodological guidance for constructing ontologies based on a set of disjoint taxonomies (understood as trees of concepts), which may be related by axioms afterwards. It seems to be open whether or how this affects reasoning with ontologies constructed in this way.

5 Conclusion

Modularization is about to attract focused research interests, in particular with respect to formal logical approaches. However, requirements, i.e., clearly defined properties which a solution should satisfy are rarely provided. In this paper, we introduce and discuss a collection of requirements for modules/ modularization in logical settings, on a fairly language-independent level. On the one hand, these requirements are motivated directly from an application perspective, which involves the manual construction and maintenance of large axiomatizations. On the other hand, we hold that these tasks bear analogy to software engineering, and therefore we study a logical interpretation of requirements for modules in that field. The overall set of requirements is best understood as a catalog of properties – nevertheless, not meant to be complete – which can be used to analyze existing proposals, as briefly exemplified for three approaches, or to guide the design of new work on modularization.

There are three major conclusions which we draw from the discussion of the requirements. Firstly, the theory of modules in software engineering has turned out to be useful for our purposes. By means of this analysis we have identified a number of requirements which are applicable and desirable in the logical case, and which have not had much overlap with those requirements specified on an ad hoc basis. So these form a useful extension of our initial list.

Secondly, in the course of the discussion it has become clear that some of the requirements will hardly be implemented by a single approach to modularization. In particular, the requirements of local correctness and completeness, classically closed composition, independence, and directionality form an area of conflict. Note that this applies independently from the actual logical framework. Accordingly, we feel that a unique solution to modularization will remain insufficient, but specialized methods should be applied in dependence on the intended application.

Thirdly, it seems that the requirement of compositionality has not been taken into account thus far. Admittedly, of those properties presented, this one is certainly the one which is closest to be termed "wishful thinking", especially if local correctness and completeness should be maintained. However, compositionality would be very beneficial, and there may be ways to achieve it if certain other requirements must be abandoned anyway, as part of some trade-off.

The presented requirements analysis is just an initial step for further work on, ultimately, implementing the top-level ontology GFO in a modular way. A direct continuation of this paper would be a more formal analysis of the interrelations of those requirements that can be captured formally, as well as relating them to properties of the logical system used. Moreover, the paper is a starting point for work on modularization which provides compositionality. In a longer term, one may consider extensions to different logics used for the modules (similar to the general LMS/ MCS approach), hence moving to hybrid logics. In this direction, the lack of a natural "reference logic", with which the resulting systems could be compared, forms another interesting topic, which we further consider to relate back to ontology, here as a source of justification.

References

1. Stephen T. Albin. *The Art of Software Architecture: Design Methods and Techniques*. Wiley and Sons, Indianapolis, 2003.
2. Eyal Amir and Sheila McIlraith. Partition-based logical reasoning. In Anthony G. Cohn, Fausto Giunchiglia, and Bart Selman, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Seventh International Conference (KR-2000), Breckenridge, Colorado, USA, April 11-15, 2000*, pages 389–400, San Francisco, 2000. Morgan Kaufmann.
3. Eyal Amir and Sheila McIlraith. Partition-based logical reasoning for first-order and propositional theories. *Artificial Intelligence*, 162(1-2):49–88, 2005.
4. Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, Cambridge (UK), January 2003.
5. Helmut Balzert. *Lehrbuch der Software-Technik: Volume 1 – Software-Entwicklung*. Lehrbücher der Informatik. Spektrum Akademischer Verlag, Berlin, 2. edition, 2000. [*in German*].
6. Alex Borgida and Luciano Serafini. Distributed Description Logics: Assimilating information from peer sources. *Journal on Data Semantics*, 1:153–184, 2003. Published as Vol. 2800 of Lecture Notes in Computer Science, Berlin, Springer.
7. Paolo Bouquet, Fausto Giunchiglia, Frank van Harmelen, Luciano Serafini, and Heiner Stuckenschmidt. C-OWL: Contextualizing ontologies. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *The Semantic Web – ISWC2003: Proceedings of the Second International Semantic Web Conference, Sanibel Island, Florida, USA, Oct 20-23*, volume 2870 of *Lecture Notes in Computer Science*, pages 164–179, Berlin, 2003. Springer.
8. William Craig. Three uses of the Herbrand-Gentzen theorem in relating Model Theory and Proof Theory. *Journal of Symbolic Logic*, 22(3):269–285, September 1957.
9. Bernardo Cuenca Grau. *Combination and Integration of Semantic Web Ontologies*. PhD thesis, Department of Informatics, Valencia University, Valencia (Spain), 2006.
10. Bernardo Cuenca Grau, Bijan Parsia, and Evren Sirin. Working with multiple ontologies on the Semantic Web. In McIlraith et al. [23], pages 620–634.
11. Bernardo Cuenca Grau, Bijan Parsia, and Evren Sirin. Combining OWL ontologies using ε-Connections. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 4(1):40–59, 2006.
12. Bernardo Cuenca Grau, Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Modularity and web ontologies. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Tenth International Conference (KR-06), Lake District of the UK, Jun 2-5*, pages 198–209, Menlo Park (California), 2006. AAAI Press.
13. Peter Forbrig and Immo O. Kerner. *Lehr- und Übungsbuch Softwareentwicklung*. Carl Hanser, Munich, 2004. [*in German*].
14. James Garson. Modularity and relevant logic. *Notre Dame Journal of Formal Logic*, 30(2):207–223, 1989.
15. Chiara Ghidini and Fausto Giunchiglia. Local models semantics, or contextual reasoning = locality + compatibility. *Artificial Intelligence*, 127(2):221–259, 2001.
16. Chiara Ghidini and Luciano Serafini. Distributed first order logic. In Dov Gabbay and Maarten de Rijke, editors, *Frontiers of Combining Systems 2: Papers presented*

at FroCoS'98, Amsterdam, 1998, number 7 in Studies in Logic and Computation, pages 121–139. Research Studies Press Ltd., Baldock (UK), 2000.
17. Rahmanathan V. Guha. Contexts: A formalization and some applications. Technical Report STAN-CS-91-1399, Computer Science Department, Stanford University, 1991. [PhD thesis].
18. Heinrich Herre, Barbara Heller, Patryk Burek, Robert Hoehndorf, Frank Loebe, and Hannes Michalek. General Formal Ontology (GFO) – A foundational ontology integrating objects and processes, Part I: Basic Principles. Onto-Med Report 8, Research Group Ontologies in Medicine, Institute of Medical Informatics, Statistics and Epidemiology, University of Leipzig, Leipzig, 2006.
19. Yannis Kalfoglou and Marco Schorlemmer. Ontology mapping: the state of the art. *The Knowledge Engineering Review*, 18(1):1–31, 2003.
20. Oliver Kutz, Carsten Lutz, Frank Wolter, and Michael Zakharyaschev. ε-Connections of abstract description systems. *Artificial Intelligence*, 156(1):1–73, 2004.
21. Douglas B. Lenat and Rahmanathan V. Guha. *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project*. Addison-Wesley, Reading (Massachusetts), 1990.
22. Bill MacCartney, Sheila A. McIlraith, Eyal Amir, and Tomás E. Uribe. Practical partition-based theorem proving for large knowledge bases. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03), Acapulco, Mexico, Aug 9-15, 2003*, pages 89–98, San Francisco, 2003. Morgan Kaufmann.
23. Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors. *The Semantic Web – ISWC 2004: Proceedings of the Third International Semantic Web Conference, Hiroshima, Japan, Nov 7-11*, volume 3298 of *Lecture Notes in Computer Science*. Springer, Berlin, 2004.
24. D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, December 1972.
25. Alan Rector. Modularisation of domain ontologies implemented in description logics and related formalisms including OWL. In John Gennari, Bruce Porter, and Yolanda Gil, editors, *Proceedings of the Second International Conference on Knowledge Capture (K-CAP'03), Sanibel Island, Florida, USA, Oct 23-25*, pages 121–128, New York, 2003. ACM Press.
26. Johannes Sametinger. *Software Engineering with Reusable Components*. Springer, Berlin, 1997.
27. Luciano Serafini and Paolo Bouquet. Comparing formal theories of context in AI. *Artificial Intelligence*, 155(1-2):41–67, May 2004.
28. Johannes Siedersleben. Software-Architektur. In Johannes Siedersleben, editor, *Softwaretechnik: Praxiswissen für Softwareingenieure*, chapter 6, pages 89–114. Carl Hanser Verlag, Munich, 2. edition, 2003. [*in German*].
29. Heiner Stuckenschmidt and Michel Klein. Modularization of ontologies. Deliverable D21, Vrije Universiteit, Amsterdam (for IST Project 2001-33052 Wonderweb), 2003.
30. Heiner Stuckenschmidt and Michel Klein. Structure-based partitioning of large concept hierarchies. In McIlraith et al. [23], pages 289–303.
31. W3C. Web Ontology Language (OWL) Specifications. W3C Recommendations, World Wide Web Consortium (W3C), Cambridge (Massachusetts), 2004. http://www.w3.org/2004/OWL/.