International Journal of Semantic Computing

PEER-TO-PEER REASONING FOR INTERLINKED ONTOLOGIES

ANNE SCHLICHT, HEINER STUCKENSCHMIDT

Computer Science Institute, University of Mannheim, B6 26,
68159 Mannheim, Germany
{*anne,heiner*}*@informatik.uni-mannheim.de*
http://ki.informatik.uni-mannheim.de

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

The Semantic Web is commonly perceived as a web of partially-interlinked machine readable data. This data is inherently distributed and resembles the structure of the web in terms of resources being provided by different parties at different physical locations. A number of infrastructures for storing and querying distributed semantic web data, primarily encoded in RDF have been developed. While there are first attempts for integrating RDF Schema reasoning into distributed query processing, almost all the work on description logic reasoning as a basis for implementing inference in the Web Ontology Language OWL still assumes a centralized approach where the complete terminology has to be present on a single system and all inference steps are carried out on this system. We have designed and implemented a distributed reasoning method that preserves soundness and completeness of reasoning under the original OWL import semantics and has beneficial properties regarding parallel computation and overhead caused by communication effort and additional derivations. The method is based on sound and complete resolution methods for the description logic $\mathcal{ALC}$ that we modify to work in a distributed setting.

Keywords: Distributed Reasoning; Ontologies

1. Introduction

The Semantic Web is commonly perceived as a web of partially-interlinked machine readable data. This data is inherently distributed and resembles the structure of the web in terms of resources being provided by different parties at different physical locations. A central goal of semantic web technology is to make this distributed network of data accessible in a way that abstracts from the physical distribution by providing tools and technologies for accessing and processing distributed data as if it was stored on the local machine. A number of infrastructures for storing and querying distributed semantic web data, primarily encoded in RDF have been developed [30]. While these infrastructures cover important aspects of distributed data handling such as indexing [31, 16], query routing [33] and optimization [20], the use of logical reasoning, another important cornerstone of semantic web technologies in a distributed setting, is a problem that is ignored by most approaches. While there are first attempts for integrating RDF Schema reasoning into distributed query processing [19], almost all the work on description logic reasoning as a basis for implementing infer-

Ontology A

$A{:}Set \sqsubseteq \exists A{:}part.A{:}Set$
$A{:}Tuple \sqsubseteq \forall A{:}part.\neg A{:}Set$

Ontology B

$B{:}Tuple \sqsubseteq B{:}Set$
$B{:}Pair \sqsubseteq B{:}Tuple$
$B{:}Pair(a)$

Mapping

$A{:}Tuple \doteq B{:}Tuple$
$A{:}Set \doteq B{:}Set$

Fig. 1. Two ontologies contain axioms that describe the concepts Set, Tuple and Pair, an ontology matcher has created a mapping between the ontologies.

ence in the Web Ontology Language OWL [5] still assumes a centralized approach where the complete terminology has to be present on a single system and all inference steps are carried out on this system. This approach has a number of severe drawbacks. First of all, the complete, possibly very large data sets have to be transferred to the central reasoning system creating a lot of network traffic. Further, transferring the complete ontologies to a single reasoner also makes this a major bottleneck in the system. This can go as far as reaching the limit of processable data of the reasoning system.

A number of approaches for distributed reasoning about interlinked ontologies have been proposed that do not require the ontologies to be sent to a central reasoner [10, 13, 22]. All these approaches rely on strong restrictions on the types of links between ontologies or the way concepts defined in another ontology may be used and refined and thus introduce special kinds of links between data sets stored in different locations. In particular, none of these approaches supports the standard definition of logical import from the OWL specification, limiting their usefulness on real data sets. We illustrate these problems using a small example.

Example 1. We assume the two ontologies depicted in Fig. 1 that are connected by a set of equivalence axioms between concept names from the two ontologies. Note that the mapping axioms can also be represented in terms of OWL imports. In this case, the mapping axioms would be part of any of the two ontologies and the other ontology would be imported by the one containing the mapping axioms.

As we can easily see, the combined ontology is inconsistent due to a mismatch in the definitions of the concept 'tuple' which in one case is defined as not containing any set. In the other case a tuple is defined as a special case of a set which in turn is defined to always contain at least one set (the empty set). The constant a which is defined as an instance of Pair which in turn is a tuple makes the ontology inconsistent.

Our goal is to have a distributed reasoning method that performs local reasoning on the two ontologies and that is still able to detect the inconsistency of the combined ontology. Looking at the previous proposals for distributed reasoning mentioned above, we notice

that none of them meets these requirements. The framework of ϵ-connections[13] does not apply in this scenario as it does not allow the specification of equivalence relationships between interlinked ontologies. Using the framework of conservative extensions does not provide any advantages in terms of local reasoning. In particular, the combined ontology is neither a conservative extension of ontology A nor of ontology B as in both cases, the additional information in the other parts can be used to derive new information concerning the signature of ontology A or ontology B, respectively, for instance the unsatisfiability of the concepts $A{:}Tuple$ and $B{:}Tuple$. Encoding the mappings in distributed description logics[10], finally, does allow us to infer a subsumption relation between the concepts $A{:}Tuple$ and $A{:}Set$, but it does not make the combined ontology inconsistent thus loosing some of the implications supported by the original import semantics.

Our aim is to develop a method for reasoning about description logic ontologies that overcomes the disadvantages of existing methods by

- preserving soundness and completeness of reasoning under the original OWL import semantics
- avoiding restrictions on the use of definitions from remote ontologies in local definitions or on the way knowledge is distributed a priori
- avoiding (large parts of) the entire local ontologies to be sent to a central reasoner but supporting local processing of the linked ontologies and limited communication via message passing
- beneficial properties regarding overhead in terms of additional derivations, communication effort needed and savings due to parallel execution of inference steps

We have designed and implemented a distributed reasoning method that satisfies these requirements. The method is based on sound and complete resolution methods for the description logic $\mathcal{ALC}$ that we modify to work in a distributed setting.

The paper is structured as follows: In the remainder of this section we address principles of distributed reasoning before giving an overview of related work in Section 2. The subsequent section gives a brief introduction to resolution reasoning and describes the distribution principle our approach is based on. Section 4 presents the details of our distributed reasoning method, in particular we show that the resolution methods proposed in [32, 23] for description logics can be distributed yielding a sound and complete distributed calculus for $\mathcal{ALC}$ ontologies. In Section 5 we investigate the properties of the method with respect to number of derivations, communication effort and degree of parallelization and show that these parameters are promising. Different options for extending the implementation in future work are discussed in Section 6. Section 7 concludes the work.

1.1. *Distributed Reasoning*

There are various options for distributing the process of logical reasoning. Many of these options have been investigated in the field of automated theorem proving for first-order logics [9, 8]. In the following we discuss these options and their pros and cons with respect

to the requirements and goals defined in the introduction. In particular, we have to make two choices:

(1) We have to choose a reasoning method that is sound and complete for description logics and permits distribution.
(2) We have to choose a distribution principle that supports local reasoning and minimizes reasoning and communication costs.

Reasoning Concerning the reasoning method [8] distinguishes between ordering-based, subgoal reduction, and instance-based strategies. Instance-based strategies are direct implementations of the Herbrand method that generate ground instances of the theory and use propositional methods for testing satisfiability. Subgoal reduction strategies build a single proof at a time by choosing and resolving subgoals and leave the logical model unchanged. Typical examples of subgoal reduction strategies are logic programming methods and analytic tableaux. Ordering-based methods, finally are based on an informed modification of a clause representation by deriving new clauses and deleting redundant ones until the empty clause is derived or no new conclusions can be drawn. This way, ordering-based methods implicitly build many proof attempts in parallel as it is not clear a priori, which derivations finally contribute to the derivation of the empty clause. Typical examples of ordering-based methods are resolution and basic superposition.

Analytic tableaux are the dominant method for implementing sound and complete inference systems for description logics [14]. It has been shown, however, that sound and complete resolution methods for expressive description logics can be defined [32, 23]. We exclude other existing methods such as a reduction of DL reasoning to logic programming from our investigation because these approaches are not sound and complete for the languages we are interested in. Because tableaux-based as well as resolution-based methods meet our requirements with respect to language coverage and completeness, the decisive factor is their suitability for distributed reasoning.

Distribution Principles The survey [8] discusses different strategies for parallelizing logical inference. In particular, the authors distinguish between parallelism at the term-, clause and the search level. The idea of parallelism at the term and the clause level is to speed up basic reasoning functions such as matching, unification or single resolution steps by executing them in parallel using a shared memory in order to improve performance. Obviously, this approach is not suitable for our purposes as it does not envision a distribution of the ontology axioms but just aims at parallel execution of basic reasoning methods. Parallelism at the search level means the parallel execution of the overall derivation process and can be further distinguished into multi-search and distributed search approaches. While in the first case, multiple search processes, often with different heuristics or different starting points are run in parallel, distributed search approaches assign parts of the search space to individual reasoners and coordinate the overall search process by exchanging intermediate results. While the first approach requires the complete ontology to be available at all reasoners, the distributed search paradigm naturally fits the distributed storage of parts of the ontology and therefore represents a paradigm that fits the goals of our research as it allows to assign

the part of the search space relevant for a specific ontology to a local reasoner instance that interacts with other local reasoners if necessary.

The choice of the distributed search paradigm has consequences for the choice of the reasoning method. In particular, it has been shown that distributed search can be used in combination with ordering-based methods [11, 7] to support parallel execution of logical reasoning. We build on top of these results by proposing distributed reasoning methods based on the principles of resolution. Our proposal extends beyond the state of the art in distributed theorem proving as it addresses specific decidable subsets of first-order logics that have not yet been investigated in the context of distributed theorem proving. Further, existing strategies for assigning inference steps to reasoners such as the ancestor-graph criterion [7] cannot avoid redundancy. We show that an assignment of clauses to reasoners based on the physical location of the corresponding axioms avoids redundancy for the special case of ordered resolution which can be used as a basis for a sound and complete distributed reasoning method for the logic $\mathcal{ALC}$.

2. Related Work

Approaches that implement distributed reasoning on description logic ontologies are relatively rare. Distribution of RDFS ontologies is more popular because adapting the rule based inference mechanism of RDFS ontologies to a distributed setting is rather straight forward compared to distributing the tableau methods that are usually used for description logic. Our approach is closer related to some distributed methods for first order reasoning because they use resolution reasoning.

2.1. *Distributed RDF Reasoning*

Marvin [24] is a platform for parallel and distributed processing of RDF data on a network of loosely-coupled peers. The reasoning method is illustrated on computation of deductive closure. Although there are no contradictions and proofs in RDF(S), the process of computing the deductive closure is comparable to saturation of a set of clauses. Hence, the method implemented by Marvin could be classified as distributed search strategy.

2.2. *Modular DL Reasoning*

Current approaches to distributed reasoning on description logic mostly rely on tableaux methods. Distribution by solving the two choices of nondeterminitic tableaux rules in parallel is difficult as it hampers the application of optimization and blocking strategies. Instead most distributed tableaux approaches try to identify all possible conflicts, i.e. all axioms that might follow from another module and would cause a contradiction and send these as queries to the other modules. So far, this is only done for links with rather restricted expressiveness between the modules.

The most prominent actually distributed T-Box reasoning implementation for ontologies is *Distributed Description Logic* (DDL [10]), a distributed search strategy. It supports only a special type of links (*bridge rules*) between ontologies. The local domains have to be disjoint, i.e. there is no real subsumption between elements of different modules.

Like DDL, *$\mathcal{E}$-connections* [13] treat local domains as disjoint and do not support subsumption relations between modules. $\mathcal{E}$-connections can be used to link ontologies of different expressivity and the resulting network can be translated to common description logics. The tableau reasoner pellet has been extended to support $\mathcal{E}$-connections but reasoning is performed by a single pellet reasoner and not distributed.

[22] propose an approach to modular ontologies based on *conservative extensions*. It is not a distributed reasoning method but a strategy for creating ontology networks with self-contained ontologies such that reasoning on the ontologies separately is complete. Roughly speaking, an ontology A is an conservative extension of another ontology B, if there is no axiom implied by A and not implied by B that uses only symbols from the local signature of B. Hence, for reasoning in the combined ontology about symbols of B, the ontology A can be safely ignored, it has no effect on the semantics of B. The idea of the approach is, that reasoning is very simple, when the ontology network is a conservative extension of each of the ontologies. While the actual reasoning is relatively easy in this approach, the complex task is to create the ontologies in compliance to the strong conservative extension requirements. A method for extracting this type of self-contained ontology modules from a larger ontology is proposed in [12].

[3] proposes a distributed tableau algorithm for *Package-based Description Logics* (PDL) [4]. Proofs for soundness and completeness of the algorithm are given but the method is not implemented and not evaluated. In particular, there is no evidence given that distributed computation of the algorithm is actually faster than computing it on one machine.

KAONp2p [15] is an infrastructure for query answering over distributed ontologies based on the KAON2 system. The approach is focussed on ontology management and knowledge selection for reasoning about assertions (Aboxes). The relevant parts of the terminologies are copied to the peer that answers the query.

2.3. *Distributed Resolution Methods*

Resolution is the most popular method for FOL provers. Approaches to distributed first order reasoning are motivated by efficiency considerations, performance is improved by using multiple processors in parallel. *Roo*[21], for example, is a parallelization of the widely (re)used first order reasoner Otter. While common resolution provers pick one so-called *given* clause and resolve it with all possible partner clauses, Roo picks multiple given clauses in parallel and solves arising conflicts of the parallel processes. In difference to the distributed methods mentioned so far, Roo uses a shared memory and can be classified as paralellization at the clause level according to [8].

Partition-Based Reasoning [2] is a distributed search strategy that requires local reasoning to be complete for consequence finding. The strong completeness requirement for local reasoning inevitably causes derivation of more clauses than necessary for refutation. Nevertheless the distribution method was shown to speed up some resolution strategies in a parallel setting with shared memory. Note that the resolution calculi applied for our distributed reasoning method are not complete for consequence finding, for efficiency considerations they are designed to generate a relatively small number of clauses.

[1] propose a distributed reasoning method for propositional theories. The authors automatically created sets of clausal theories connected by shared variables and investigated the correlations between characteristics of the created theory networks and queries (e.g. number and length of link clauses) and characteristics of the query processing (e.g. depth of a query). The query runtimes where investigated for different complexity of theory networks and queries but not compared to centralized computation.

3. Distributing Logical Resolution

Before describing our distributed resolution method for ontologies, we first briefly review standard resolution reasoning and present the basic idea for distributing resolution according to the distributed search paradigm.

3.1. *Resolution Theorem Proving*

Resolution is a very popular reasoning method for first order logic (FOL) provers. As description logics are a strict subset of first order logic, resolution can be applied to ontologies as well [34]. For this purpose the DL ontology is transformed into a set of first order clauses as defined in section 4.1. Figure 2 illustrates this translation on the example from Figure 1. This translation can be done on a per axiom basis independently of other parts of the ontology. It can be shown that the ontology is consistent if and only if the set of clauses is consistent. The set of clauses is consistent iff exhaustive application of the standard resolution rules does not derive an empty clause. Standard resolution consists of two rules, the main resolution rule is accompanied by the factoring rule.

Definition 1. (Standard Resolution) For clauses C and D and literals A and $\neg B$, standard resolution is defined by the rules

$$\text{Resolution } \frac{C \vee A \quad D \vee \neg B}{C\sigma \vee D\sigma}$$

$$\text{Factoring } \frac{C \vee A \vee B}{C\sigma \vee A\sigma}$$

where the substitution σ is the most general unifier of A and B.

In most cases, the resolution rule alone derives an empty clause if the set of input clauses is inconsistent. But, to derive the empty clause e.g. from the two clauses $P(x) \vee P(y)$ and $\neg P(a) \vee \neg P(b)$ factoring is necessary. It is also possible to combine both rules into one inference rule.

Resolution with factoring is complete for deciding satisfiability of a first order theory, but for efficient resolution reasoning reduction rules have to be added that delete redundant clauses. Reduction rules are similar to inference rules, the difference is that premises are deleted when applying the rule. The most important reduction rule is subsumption elimination, it deletes clauses that are subsumed by other clauses, e.g. clauses $P(x) \vee Q(x) \vee R(x)$ and $P(x) \vee Q(x)$ are replaced by $P(x) \vee Q(x)$ (i.e. the subsumed clause $P(x) \vee Q(x) \vee R(x)$ is deleted).

Definition 2. (Subsumption Reduction) For clauses C and D subsumption reduction is defined by the rule

$$\text{Subsumption Reduction } \frac{C \quad D}{C}$$

where the literals of $C\sigma$ are a subset of the literals of D for some substitution σ.

3.2. *Distributed Resolution*

Algorithm 1 controls the application of resolution rules on a set of input clauses. For recording which clauses have already been resolved with each other, the clause set is split into two sets, the *usable (Us)* set of clauses that have to be resolved and the *worked off (Wo)* set. Clauses in the *Wo* set are saturated, i.e. every clause that could be derived from *Wo* is either already contained in *Wo* or *Us* or redundant. Until the whole set of clause is saturated or an empty clause is found, the algorithm repeatedly picks a clause (the *given* clause) from the *Us* set and moves it to the *Wo* set (lines 4 to 6). Then resolution is applied to the given clause and each clause from the *Wo* set that can be resolved with *given* (e.g. contains a literal that can be unified with a negated literal of the given clause) to obtain new clauses. Reduction is performed in line 8, the "reduce"-function includes deletion of redundant clauses in all clause sets. After reduction the remaining new clauses are added to the *Us* set. The reduction is an essential part of a resolution prover and the most time consuming component [36]. Without reduction, the number of clauses in the *Us* set increases much faster than clauses are moved to *Wo*, hence *Us* is never empty and it is impossible to saturate even a small set of clauses with reasonable time and storage consumption. The structure of reduction rules is the same as for inference rules, but the premises are deleted on application. Algorithm 1 is a simplified version of the resolution prover specified in [36].

Algorithm 1 Resolution Prover

ISSATISFIABLE(KB)[t]

1: $Wo \leftarrow \emptyset$
2: $Us \leftarrow KB$
3: **while** $Us \neq \emptyset$ **and** $\square \notin Us$ **do**
4: $\quad Given \leftarrow$ CHOOSE(Us)
5: $\quad Us \leftarrow Us \setminus \{Given\}$
6: $\quad Wo \leftarrow Wo \cup \{Given\}$
7: $\quad New \leftarrow$ RESOLVE(Given,Wo)
8: $\quad (Given, Us, New) \leftarrow$ REDUCE($Given, Us, New$)
9: $\quad Us \leftarrow Us \cup New$
10: **end while**
11: **if** $\square \in Us$ **then return** false
12: **else return** true
13: **end if**

This basic algorithm can be modified to support distributed reasoning. In particular, the inferences can be distributed across different reasoners by separating the set of input clauses and running provers on separate parts of the set:

- Every reasoner separately saturates the clause set assigned to it.
- Newly derived clauses are propagated to other reasoners if necessary.

The propagation of clauses is added into line 9 of Algorithm 1: instead of directly adding the new clauses to the local *Us* set, some of the new clauses may be added to the *Us* sets of other reasoners and new clauses received from other reasoners may be added to the local *Us* set.

Definition 3. (Allocation) An *allocation* for a set $\mathcal{C}$ of clauses and a set of modules M is a relation $a \in (\mathcal{C} \times M)$ such that

$$\forall c \in \mathcal{C} \colon \exists m \in M \colon a(c, m)$$

The set of modules a clause c is allocated to by the allocation a is defined by

$$a(c) := \{m \in M \mid a(c, m)\}$$

If the allocation relation is functional we may omit the parenthesis and write $a(c) = m$.

In addition to the propagation of clauses we have to add a second modification to the algorithm to turn it into a distributed resolution algorithm. In contrast to the centralized case, a reasoner that has saturated the local clause set may have to continue reasoning once a new clause is received from another reasoner. The whole system of connected reasoners stops if the empty clause is derived by one of the reasoners or all are saturated.

After this intuitive description of a distributed resolution algorithm, we define distributed resolution formally:

Definition 4. (Distributed Resolution Calculus) A *distributed resolution calculus* $R(a)$ is a resolution calculus that depends on an allocation relation $a \colon \mathcal{C} \to M$ such that each rule *r* of $R(a)$ is restricted to premises $P \subset \mathcal{C}$ with

$$\exists m \in M \colon \forall c \in P \colon a(c, m)$$

Hence, the rules of a distributed resolution calculus are restricted to premises allocated to the same module. A distributed calculus can be obtained from any resolution calculus by defining an allocation relation and adding the allocation restriction to each rule of the calculus.

Definition 5. (Allocation Restriction) Given a rule *r* of a resolution calculus and an allocation function *a* for a set of clauses $\mathcal{C}$ with range *M* the *allocation restriction* is

$$\exists m \in M \colon \forall c \in P \colon a(c, m)$$

where *P* is the set of premises of rule *r*.

Obviously, termination of the underlying calculus is preserved by distribution if it does not depend on reduction rules. In the worst case, each inference of the original calculus

is performed once in every module of the distributed calculus. The results presented in Section 5 also indicate that local reduction (i.e. deleting clauses that are redundant with respect to the reasoner they are processed by) is sufficient in practice.

Preserving completeness without allocating each clause to every reasoner is more difficult, we have to make sure the allocation restriction never excludes inferences that are possible in the original calculus. Referring to Algorithm 1 we have to ensure a given clause is resolved with every matching clause in any of the distributed *Wo* sets. Hence, for standard resolution, a given clause C has to be propagated to any reasoner whose *Wo* set contains a clause with a literal that matches (i.e. is unifiable and of opposite polarity) any of the literals in C. This would lead to a substantial communication overhead and potentially redundant inference steps.

To avoid redundancy, we aim at allocating every clause to only a single reasoner. A functional allocation guarantees that the same resolution step is never carried out twice, because equivalent clauses are always assigned to the same unique reasoner which takes care of avoiding local redundancy. In the next section we will show how this can be achieved without sacrificing completeness.

4. Distributed Resolution for $\mathcal{ALC}$

As we have seen above, the ability to define a sound and complete distributed reasoning method relies on two requirements: (1) the existence of a sound and complete resolution calculus and (2) the ability to find a corresponding allocation function that satisfies the allocation restriction. In this section, we show that for the case of ontologies defined in $\mathcal{ALC}$ both of these requirements can be satisfied leading to a sound and complete distributed resolution method. We do not address reduction rules in this section because reduction is not necessary to guarantee the theoretical properties of the proposed calculus. However, for efficient reasoning reduction is essential and hence the practical effects distribution has on reduction are discussed in the experimental section.

4.1. *Preliminaries*

Before presenting the calculus our distributed method is based on, we define the description logic we use and the translation of description logic axioms to first order clauses.

The Description Logic $\mathcal{ALC}$ An $\mathcal{ALC}$ ontology is a set $\mathcal{O}$ of axioms α build from concepts C according to the following syntax given in Backus-Naur-Form in Table 1.

The *signature* of an ontology $Sig(\mathcal{O})$ is the disjoint union of concept names, property names and individual names. The semantics of $\mathcal{ALC}$ is defined by a mapping of the concepts and axioms to first order logic. For example, the $\mathcal{ALC}$ axiom $Set \sqsubseteq \exists part.Set$ translates to $\forall x\colon Set(x) \rightarrow \exists y\colon part(x,y) \wedge Set(y)$.

The translation of a description logic ontology to FOL is the conjunction of the translated axioms. Note that concept names correspond to unary predicates, property names correspond to binary predicates. The normalization presented in the next subsection greatly simplifies the translation to first order clauses, therefore we give the simplified mapping after

$$\begin{aligned}
\alpha &::= C \sqsubseteq C | C \equiv C | C(x) | R(x,x) \\
C &::= \top | \ \bot \ | A | \neg C | C \sqcap C | C \sqcup C | \exists R.C | \forall R.C \\
A &::= concept_name \\
R &::= property_name \\
x &::= individual_name
\end{aligned}$$

Table 1. $\mathcal{ALC}$ Syntax

describing normalization.

Normalization The resolution calculus we apply requires first order clauses as input, hence the first order formulas obtained from an ontology are translated to clauses. To guarantee termination of the applied resolution calculus, the ontology has to be normalized prior to clausification. This ensures that only certain types of axioms and corresponding clauses occur in the reasoning procedure. For simplicity, we assume the ontology contains only subsumption axioms $A \sqsubseteq C$ where A is not a complex concept and no equivalence axioms. Complex subsumptions $C \sqsubseteq D$ are equivalent to $\top \sqsubseteq \neg C \sqcup D$ and equivalences $C \equiv D$ can be replaced by two subsumptions $C \sqsubseteq D$ and $D \sqsubseteq C$. The definitorial form normalization we use replaces complex concepts C in the right hand side of an axiom by a new concept name A and adds the axiom $A \sqsubseteq C$ to the ontology. Thus, it splits up nested axioms into simple ones by introducing new concepts. E.g. the axiom $A \sqsubseteq B \sqcup \exists r.C$ is replaced by the axioms $A \sqsubseteq B \sqcup Q$ and $Q \sqsubseteq \exists r.C$.

Definition 6. (Definitorial Form) For simple subsumptions $A \sqsubseteq D$ with atomic concept A the Definitorial Form is defined by

$$\text{Def}(A \sqsubseteq D) := \begin{cases} \{A \sqsubseteq D\} & \text{if all subterms of } D \text{ are literal concepts} \\ \{Q \sqsubseteq D|_p\} \cup \text{Def}(A \sqsubseteq D[Q]_p) & \text{if } D|_p \text{ is not a literal concept} \end{cases}$$

where a literal concept is either a concept name or a negated concept name and Q is a new concept name.

Clausification After normalization, the ontology contains only simple axioms that are

translated to first order clauses as follows:

$A \sqsubseteq B$	$\neg A(x) \vee B(x)$
$A \sqsubseteq B \sqcap C$	$\neg A(x) \vee B(x)$
	$\neg A(x) \vee C(x)$
$A \sqsubseteq B \sqcup C$	$\neg A(x) \vee B(x) \vee C(x)$
$A \sqsubseteq \exists r.B$	$\neg A(x) \vee r(x, f(x))$
	$\neg A(x) \vee B(f(x))$
$A \sqsubseteq \forall r.B$	$\neg A(x) \vee \neg r(x, y) \vee B(y)$

E.g. the axiom $A \sqsubseteq B$ corresponds to the first order formula $\forall x\colon A(x) \rightarrow B(x)$ which is equivalent to the clause $\neg A(x) \vee B(x)$. As usual, all variables are implicitly $\forall$-quantified, existencial quantifiers are translated using skolem functions. The clauses resulting from the ontologies of Figure 1 are depicted in Figure 2.

4.2. *A Complete Resolution Method for $\mathcal{ALC}$*

It has been shown that standard resolution methods can be adapted to provide sound and complete reasoning for $\mathcal{ALC}$ [32, 23]. These methods that provide the basis for our work rely on ordered resolution, which depends on two parameters: An *order* of literals and a *selection function* that maps every clause to a subset of it's negative literals. These parameters are used to restrict the applicability of inference rules, thus reducing the number of derived clauses and avoiding redundant inferences. The ordered resolution calculus consists of the following two inference schemata ordered resolution and factoring. The letters C and D refer to clauses, A and B are literals.

Definition 7. (Ordered Resolution)

$$\text{Ordered resolution} \quad \frac{C \vee A \quad D \vee \neg B}{C\sigma \vee D\sigma}$$

where

(1) σ is the most general unifier of A and B
(2) either $\neg B$ is selected in $D \vee \neg B$ or else nothing is selected in $D \vee \neg B$ and $B\sigma$ is maximal w.r.t. $D\sigma$
(3) $A\sigma$ is strictly maximal with respect to $C\sigma$
(4) nothing is selected in $C\sigma \vee A\sigma$

$$\text{Positive factoring} \quad \frac{C \vee A \vee B}{C\sigma \vee A\sigma}$$

where

(1) σ is the most general unifier of A and B
(2) $A\sigma$ is maximal with respect to $C\sigma \vee B\sigma$
(3) nothing is selected in $C\sigma \vee A\sigma \vee B\sigma$

This calculus is well known to be sound and complete for first order clauses. However, for guaranteeing termination on $\mathcal{ALC}$ a special parameterization is necessary. In particular, [32] showed that using ordered resolution, decidability on $\mathcal{ALC}$ can be preserved by transforming the ontology into *definitorial form* (see Definition 6) before clausification and applying ordered resolution with appropriate selection and ordering: If standard clausification is applied to an $\mathcal{ALC}$ ontology in definitorial form, satisfiability of the resulting clauses can be decided by the following calculus:

Definition 8. ($\mathcal{ALC}$ Resolution) *$\mathcal{ALC}$ resolution* $R_{\mathcal{ALC}}$ is the calculus

- consisting of the inference rules ordered resolution and factoring,
- with selection of exactly the negative binary literals, and
- literal ordering $\succ$ with $R(x, f(x)) \succ \neg C(x)$ and $D(f(x)) \succ \neg C(x)$, for all function symbols f, and predicates R, C, and D.

The ordering requirement is satisfied by every lexicographic path ordering (LPO, Definition 9) based on a total precedence $>$ on function, predicate and logical symbols with $f > P > \neg$ for every function symbol f and predicate symbol P.

Definition 9. (Lexicographic Path Ordering) A lexicographic path ordering (LPO) is a term ordering induced by a well-founded strict precedence $>$ over function, predicate and logical symbols, defined by:
$s = f(s_1, \ldots, s_m) \succ g(t_1, \ldots, t_n) = t$ if and only if
t is a proper subterm of s or at least one of the following holds

(i) $f > g$ and $s \succ t_i$ for all i with $1 \leq i \leq n$
(ii) $f = g$ and for some j we have $(s_1, \ldots, s_{j-1}) = (t_1, \ldots, t_{j-1})$, $s_j \succ t_j$ and $s \succ t_k$ for all k with $j < k \leq n$
(iii) $s_j \succeq t$ for some j with $1 \leq j \leq m$

LPOs have the subterm property, i.e. $t \succ t'$ for all terms t' that are subterms of term t. Furthermore, if $>$ is total, the LPO induced by $>$ is total on ground terms. For the simple types of literals occurring in clauses translated from $\mathcal{ALC}$ axioms (Table 2 described below), a LPO ordering with precedence as required by Definition 8 breaks down to a simple ordering definition. Literals that contain different variables are incomparable, for literals that share the same variables the ordering is determined by three rules:

- Literals containing a function symbol precede literals that do not contain a function symbol.
- Literals containing a function symbol are ordered according to the precedence of the function symbols.
- Literals not containing a function symbol are ordered according to the precedence of the predicate symbols.

We briefly review the proof for termination of $R_{\mathcal{ALC}}$ on $\mathcal{ALC}$ ontologies because it is relevant for distribution of the calculus.

Theorem 1. ($\mathbf{R}_{\mathcal{ALC}}$ Termination [32]**)** For an $\mathcal{ALC}$ knowledge base KB, saturating the clauses obtained from the definitorial form of KB by $R_{\mathcal{ALC}}$ decides satisfiability of KB and runs in time exponential wrt. the size of KB.

The theorem can be proved by showing that the set of $\mathcal{ALC}$ clauses (i.e. clauses obtained from translation of an $\mathcal{ALC}$ ontology) depicted in Table 2 is closed under $R_{\mathcal{ALC}}$.

	$\mathcal{ALC}$ clause type	resolvable literal
1	$R(x, f(x)) \vee \mathbf{P}(x)$	$R(x, f(x))$
2a	$\mathbf{P}(x)$	$(\neg)P(x)$
2b	$\mathbf{P_1}(\mathbf{f}(x)) \vee \mathbf{P_2}(x)$	$(\neg)P(f(x))$
3	$\neg R(x, y) \vee \mathbf{P_1}(x) \vee \mathbf{P_2}(y)$	$\neg R(x, y)$
4	$\mathbf{P}(\mathbf{a})$	$(\neg)P(a)$
5	$(\neg)R(a, b)$	$(\neg)R(a, b)$

Table 2. Clause types resulting from the translation of an $\mathcal{ALC}$ ontology to first order clauses.

$\mathbf{P}(t)$, where t is a term, denotes a possibly empty disjunction of the form $(\neg)P_1(t) \vee \cdots \vee (\neg)P_n(t)$. $\mathbf{P}(\mathbf{f}(x))$ denotes a disjunction of the form $\mathbf{P_1}(f_1(x)) \vee \cdots \vee \mathbf{P_m}(f_m(x))$. Note that this definition allows each $\mathbf{P_i}(f_i(x))$ to contain positive and negative literals. We adapted the notation of the $\mathcal{ALC}$ clauses from [23] and subdivided type 2 clauses because the resolvable literal (discussed below) is of type $P(f(x))$ if one exists and otherwise $P(x)$.

When an $\mathcal{ALC}$ ontology in definitorial form is translated to first order clauses as described in Section 4.1, every clause is of one of the types listed in Table 2. Further, applying ordered resolution to clauses of the listed types results in a listed clause type. Since functions are not nested, the number of literals that can be built using a finite number of symbols is finite. Hence, if duplicated literals are deleted, the saturation terminates because otherwise it would create a infinite number of clauses from a finite number of literals.

In this section, we described a sound and complete resolution calculus for $\mathcal{ALC}$. What remains to be shown is that we can define a corresponding allocation function that preserves completeness of this calculus.

4.3. *Distributing $\mathcal{ALC}$ Resolution*

The idea for defining an allocation function is to restrict the inference options such that there is a unique literal for every clause that may be resolved in a subsequent inference step and use this unique literal as a basis for deciding the allocation of the clause. This guarantees that clauses that can be the premises of a resolution step are always sent to the same reasoner thus enabling every inference that occurs in the centralized resolution reasoner. This idea is stated precisely in the following definitions and results.

Definition 10. (Resolvable Literal) A literal L of a clause $C \vee L$ is a *resolvable literal* of $C \vee L$ wrt. $R_{\mathcal{ALC}}$ iff there is a clause $D \vee L'$ such that $R_{\mathcal{ALC}}$ can be applied to the premises $C \vee L$ and $D \vee L'$ deriving the clause $(C \vee D)\sigma$ where σ is an appropriate substitution.

The essential precondition for our distribution strategy is that for a given clause, the literal that will be resolved in the next resolution step does not depend on the available other premise candidates, it can be determined without considering other clauses.

Theorem 2. (Unique resolvable literal) Every $\mathcal{ALC}$ clause contains exactly one resolvable literal.

Theorem 2 holds because a resolvable literal has to be either selected or maximal and part of a clause with no selected literal. $\mathcal{ALC}$ clauses contain at most one selected literal and the ordering is total on ground clauses and clauses containing only one variable. Thus, only clauses that contain more than one variable may have multiple maximal literals, but then they are of type 3 and contain one selected literal.

Note that for $\mathcal{ALC}$ clauses, using a LPO ordering yields a simple strategy for determining the resolvable literal: If the clause contains a selected literal, this is the resolvable literal. Else, if the clause contains a function symbol, the literal containing the highest function symbol according to the precedence of symbols is the resolvable literal. If the clause contains neither selected literal nor function, the literal containing the highest predicate symbol is the resolvable literal.

Based on the uniqueness of the resolvable literal, we can now define a suitable allocation function for $\mathcal{ALC}$.

Definition 11. (Allocation for $\mathcal{ALC}$) The allocation of a clause c for the distributed calculus $R_{\mathcal{ALC}}$ is defined by

$$a_{\mathcal{ALC}}(c) := alloc(topSymbol(resolvableLiteral(c)))$$

where

- $resolvableLiteral(c)$ is the unique resolvable literal of clause c.
- $topSymbol(L) := P$ for every literal $L = (\neg)P(t)$ or $L = (\neg)P(t_1, t_2)$ with unary or binary predicate symbol P and terms t, t_1, t_2.
- $alloc\colon Sig(\mathcal{O}) \rightarrow M$ is an allocation of the signature symbols of the input ontology $\mathcal{O}$, including concepts introduced by the definitorial form transformation.

Note that for interlinked ontologies, $\mathcal{O}$ is the union of all ontologies and $alloc(X)$ can be defined by the namespace of the concept or property name X. For a single ontology, every partitioning of the ontology terms induces an allocation of symbols via randomly allocating parts to modules. In previous work we showed that allocation for a monolithical ontology can be computed using graph partitioning methods [26, 25]. But, the amount of communication between the peers is larger and the reasoners spend more time waiting for new input when the knowledge base is a partitioned ontology because the ontology parts are generally more densely connected than ontologies in an ontology network. For determining where (and if) a derived clause is sent, we first pick the unique resolvable

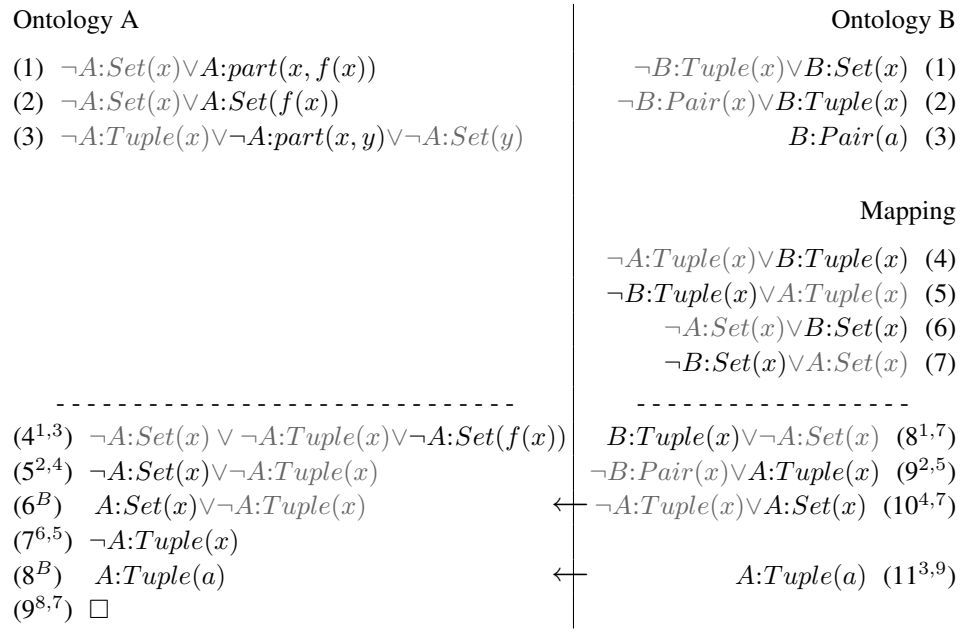

Fig. 2. Distributed ordered resolution example on the ontologies from Figure 1. The axioms are translated to clauses specified by lists of literals, all variables are universally quantified implicitly. The existential quantification is translated to a skolem function. Inferred clauses are depicted below the dashed line, the superscript numbers refer to the origin of a clause, i.e. the premises it is derived from or the ontology that send it. Arrows indicate propagation of a clause to the other ontology. The precedence of symbols is $B{:}Set > B{:}Tuple > B{:}Pair > A{:}Set > A{:}Tuple > A{:}part$, all literals with top symbol $A{:}part$ are selected. To help tracking the inferences resolvable literals are set in black, other literals in gray. Hence, a derived clause consists of the gray literals of its premises.

literal of the clause, then the top symbol of this literal and finally the reasoner this symbol is allocated to. If a symbol s is allocated to a module m we say that module m is *responsible* for s. Figure 2 illustrates distributed resolution on the example ontologies from Figure 1. The completeness of the distributed method for deciding $\mathcal{ALC}$ satisfiability follows from the similar property of $R_{\mathcal{ALC}}$.

Theorem 3. (Completeness of distributed $\mathcal{ALC}$ resolution) The distributed resolution calculus $R_{\mathcal{ALC}}(a_{\mathcal{ALC}})$ is a complete resolution calculus for deciding $\mathcal{ALC}$ satisfiability.

Theorem 3 holds because we can show that the allocation restriction is satisfied for every application of a rule $r \in R_{\mathcal{ALC}}$. I.e. every pair of $\mathcal{ALC}$ clauses that can be resolved as premises in an ordered resolution inference are allocated to the same module because the literals that are unified in the inference have the same top predicate and are the unique resolvable literals of their respective clauses. Thus, distributing ordered resolution according to the allocation given in Definition 11 is a sound, complete and terminating distributed method for deciding $\mathcal{ALC}$ satisfiability. Hence, we can allocate each derived clauses to

a reasoner based on the namespace of a specific concept or property name. The theorem guarantees that the resulting distributed inference process performs sound and complete reasoning over the interlinked ontology.

5. Experiments

We conducted a number of experiments for testing the performance of the distributed resolution method in a realistic setting that are reported in this section. The main goal of these experiments is to analyze the effectiveness of the method on realistic data. We identify three parameters that influence the effectiveness of the method and are therefore the main target of our investigation besides the actual upload and runtime.

Number of Derived Clauses Resolution methods often suffer from the extremely high number of clauses that are generated in the derivation process. This is the case because resolution is not per se goal-driven but relies on the generation of new clauses from the existing knowledge base until the empty clause is derived building a number of proof attempts in parallel. In a centralized setting, the number of clauses is reduced by a number of strategies for identifying and eliminating redundant clauses. It is well known that this elimination cannot completely be done in a distributed setting. Therefore the number of clauses produced by our method in comparison to a centralized setting is an important parameter. Our claim is that the number of derived clauses is not significantly higher than in the centralized setting.

Amount of Communication Communication overhead created by the necessary exchange of clauses between reasoners is a major source of ineffectiveness as communication costs are relatively high compared to the costs for local derivations. This is especially relevant if the reasoners are distributed in a wide network which is a realistic setting in the context of the semantic web. In general, a distributed method can only be effective if the amount of communication needed is small in comparison to the local reasoning steps. We therefore investigate the number of clauses that are exchanged between the different reasoners in comparison to the overall number of clauses derived. We claim that the number of clauses to be exchanged is not a significant part of the derived clauses or in other words that most of the derived clauses stay at the same reasoner where they were derived.

Degree of Parallelization Besides the reduction of the amount of data that has to be handled by a single reasoner the main argument in favor of distributed reasoning is the potential to perform multiple reasoning steps in parallel. This, however, requires a communication scheme that guarantees the availability of necessary information at all reasoners. In case of a suboptimal communication scheme, reasoners will be idle for some or most of the time waiting for input from other reasoners. This will significantly reduce the performance of the overall system. We therefore investigate the average idle time of reasoners. Our claim is that all reasoners are busy most of the time.

5.1. *Implementation*

Our distributed resolution implementation is based on the first order prover SPASS[a] [37], of which Algorithm 1 gives an overview. A number of different resolution strategies including ordered resolution are supported, precedence and selection can be specified in the input file. We implemented the definitorial form normalization separately and use Spass for clausification of the normalized ontology. We store the clauses in separate files and include precedence and selection to every input file. Apart from compliance to the requirements of $R_{\mathcal{ALC}}$ (Definition 8) the precedence was random, the selection function selects all negative binary predicates. The applied reduction rules include forward and backward subsumption reduction[b]. A prerelease of our implementation is available online[c]. For turning SPASS into a distributed reasoner (i.e. adding the "Distributed" option) the main extension is support for sending and receiving clauses. Sending is the easy part, the provided printing methods were modified to print to a string instead of a file and the string is then send to another module via TCP (Transmission Control Protocol) connection. For receiving new clauses the input parser was modified to read clause lists from a string that are subsequently added to the local clause store. A set of received clauses is treated like a set derived from a given clause, i.e. it is forward and backward reduced with respect to the local worked off clause list before adding the non redundant received clauses to the usable list. It is important to make sure that forward and backward reduction are performed prior to sending and after receiving a clause for deleting as many redundant clauses as possible.

Since the number of modules is small in our setting we connect all reasoners at startup, for larger number of modules we will only establish the necessary connections. To avoid the local reasoning being blocked on sending a clause while the destination module is busy and cannot receive, the clause communication is performed in separate processes. I.e. the send routine that is called after reduction for clauses allocated to other modules, does not actually send the clause but writes it to a send buffer. The actual communication process sends the content of this buffer to its destination when the destination module is idle. The priority of the reasoning and communication processes is adjusted such that while not saturated locally, a clause is only send to another reasoner if this destination reasoner signals that it is idle. New clauses are only received when the local clause set is completely saturated.

Startup and shutdown of the system is initialized by a central control process. In a fully decentralized P2P system this job is performed by the peer that receives a query. The control process starts the separate machines on their respective input clauses files. Apart from passing clauses between each other the reasoners send status messages whenever they are locally saturated, when they continue reasoning on newly received clauses and when they derive an empty clause. When one reasoner finds a proof or all reasoners are saturated

[a] `http://www.spass-prover.org`

[b] The complete configuration for Spass is: Distributed=1 PGiven=0 PDer=0 PProblem=0 Auto=0 Splits=0 Ordering=1 Sorts=0 Select=3 FullRed=1 IORe=1 IOFc=1 IEmS=0 ISoR=0 IOHy=0 RFSub=1 RBSub=1 RInput=0 RSSi=0 RObv=1 RCon=1 RTaut=1 RUnC=1 RSST=0 RBMRR=1 RFMRR=1

[c] `http://ki.informatik.uni-mannheim.de/dire.html`

for an interval longer than the maximal time necessary for clause propagation the query is answered and the reasoners are shut down.

5.2. *Datasets*

We tested our implementation on two datasets: A set of two large ontologies connected by a very large number of links and a set of five more sparsely connected ontologies.

The Anatomy Dataset The ontologies of the anatomy track are the NCI Ontology describing the human anatomy, published by the National Cancer Institute (NCI), and the Adult Mouse Anatomical Dictionary, which has been developed as part of the Mouse Gene Expression Database project. Both resources are part of the Open Biomedical Ontologies (OBO). The complex and laborious task of generating the reference alignment has been conducted by a combination of computational methods and extensive manual evaluation. In addition, the ontologies were extended and harmonized to increase the number of correspondences between both ontologies. The subset of the NCI Ontology used in the experiments contains 3304 concepts describing the human anatomy while the mouse anatomy ontology contains 2744 concepts of the anatomy of the mouse. Despite the fact that one ontology describes humans and the other mice, there are a significant number of correspondences between concepts in the two ontologies. The manually created reference mapping contains 1544 correspondences between concepts in the two ontologies. An elaborate description of creating the reference alignment can be found in [6, 17]. To allow answering queries, we use a coherent version of the reference alignment that was provided by the organizers of the Ontology Alignment Evaluation Initiative (OAEI). The data set contains no individuals, it covers the scenario where a small number of relatively large ontologies of rather low expressiveness are connected through a mapping.

The OntoFarm Dataset The OntoFarm data set[d] consists of a set of ontologies in the domain of conference organization that have been collected by researchers of the Knowledge Engineering Group at the University of Economics Prague [35]. The ontologies cover the structure of a conference, involved actors, as well as issues connected with the submission and review process. Compared to the anatomy dataset, these ontologies are sparsely connected and rather small. From the 15 ontologies in this collection we used five ontologies (cmt, confOf, ekaw, iasted, sigkdd) in our experiments, because for these ontologies manually created and verified mappings were available. Table 3 summarizes the properties of these five ontologies.

In order to be able to apply our distributed reasoning method to this set of ontologies, we converted datatype properties to object properties and eliminated logical operators outside the $\mathcal{ALC}$ fragment, i.e. property hierarchies, functional, transitive and inverse properties. The resulting ontologies together with manually created and verified mappings are used as the basis for a second set of experiments that cover the scenario of a set of multiple small

[d]The ontologies are available at `http://nb.vse.cz/~svabo/oaei2006/`.

Name	Number of Classes	Number of Properties	Logic
Cmt	36	59	$\mathcal{ALCIF}(D)$
ConfTool	38	36	$\mathcal{SIF}(D)$
Ekaw	77	33	$\mathcal{SHIN}(D)$
Iasted	140	41	$\mathcal{ALCIF}(D)$
Sigkdd	49	28	$\mathcal{ELI}(D)$

Table 3. Ontologies from the OntoFarm dataset used in the experiments

ontologies.

5.3. *Experiments on the Anatomy Dataset*

We carried out experiments on the Anatomy Dataset in three different settings. In a first setting we translated the entire knowledge base consisting of both ontologies and mappings encoded as equivalence statements between predicates into clauses, loaded these clauses into the first-order reasoner SPASS letting the reasoner chose its own settings on the dataset. We refer to this setting as 'autoconfig' and use it as a baseline. We compared this basic setting with our own implementation of the distributed resolution method introduced above. More specifically, we use two different settings. The first setting simulates a non distributed setting by using only a single reasoning peer that uses ordered resolution for reasoning about the entire knowledge base. We refer to this setting as 'global'. Finally, we used a setting that corresponds to our vision of distributed reasoning. Here each of the two ontologies is placed on a reasoning peer. The equivalence axioms representing the mappings between the ontologies are added to both ontologies. Distributed ordered resolution is used to test the overall knowledge base for consistency. We refer to this setting as 'distributed'. In the following we present and discuss the results of our experiments with respect to the number of derived clauses, the amount of communication needed and the degree of parallelization as well as the time needed for loading and and reasoning about the ontologies.

Load- and Runtime We compared the time needed for loading the clause sets into the reasoner as well as the times needed for executing a consistency check on the knowledge base. Concerning the latter it has to be noted that the combined ontology is consistent. This means that the knowledge base has to be completely saturated and there is no early termination due to the derivation of the empty clause. The results of comparing the upload and runtimes in the three different settings is summarized in Table 4.

As expected, loading the clause set into the reasoner takes most time in the autoconfig setting. This is natural as the reasoner first has to analyze the data to determine the best parameter configuration. As the parameters are manually determined in the global setting, the upload time is significantly lower. We can also observe that the upload time is again reduced in the distributed setting. This can be explained by the fact that on upload time a number of index structures are initialized. The effort for creating these indices is super-

	autoconfig	global	distributed
Upload Time	436,8	20,2	6,7
Runtime	515,8	202,4	55,1

Table 4. Upload and runtime on the Anatomy dataset in seconds

linear. As a result, the time needed to upload the ontology is again significantly reduced in the distributed setting despite the fact that the overall number of clauses that have to be loaded is higher as the clauses representing mappings between the ontologies are uploaded twice. Thus a first advantage of our distributed reasoning method is in reducing the upload time for large ontologies.

We can observe a similar situation with respect to runtime. In the autoconfig setting, the SPASS reasoner needs nine minutes for checking the consistency of the overall knowledge base. In the global setting, this is already reduced to about two minutes. This reduction can be explained by the choice of the specific resolution method that is known to perform well on $\mathcal{ALC}$ ontologies. The important observation is that in the distributed setting, the runtime is again significantly reduced to less than a minute. This speedup cannot only be explained be by the parallelization of the reasoning as this could have only reduced the runtime to about 100 seconds. The fact that less than a minute is needed indicates that the distributed setting also benefits from the reduction of the local search spaces that makes local derivations more effective. This means that a second major benefit of our method is a speedup of local reasoning due to the reduction of the local search spaces.

Derived Clauses and Propagations When trying to analyze the behavior of our distributed reasoning method and explain the significant reduction of the runtime, we first look at the number of derived clauses and the number of clauses that are propagated between the different reasoner peers. Table 5 summarizes the results with respect to this aspect.

	autoconfig	global	distributed
derivations	15324	15324	15324
propagations	-	-	8567
	(0%)	(0%)	(56%)

Table 5. Derivation and propagation of clauses on the Anatomy dataset in number of derived clauses

The first thing we notice is the fact that in all three settings, the same number of clauses were derived to compute the closure of the knowledge base. For the distributed setting, this is good news as it means that for this setting it was sufficient to perform redundancy checks and reductions at the local peers without missing potential for optimization which potentially is a major problem for distributed theorem proving methods. We can further see

that a significant amount of communication took place between the two reasoning peers in the distributed setting. This means that the good results concerning runtime cannot be explained simply by the lack of interactions between the ontologies. Further, this shows that the effort needed for exchanging clauses between the reasoners is overcompensated by the benefits of having smaller local search spaces.

Degree of Parallelization Further, we look at the degree of parallelism by looking at the average time the reasoning peers where busy compared to the overall runtime of the system. Obviously, in the autoconfig and the global setting where we only have a single reasoner this reasoner is active all the time and therefore is busy for 100%. The interesting parameter is the average time a reasoning peer is busy in the distributed setting. The respective information is provided in Table 6.

	autoconfig	global	distributed
Runtime	536,2	112,4	55,1
Avg. Busy Time	536,2	112,4	45,0
	(100%)	(100%)	(82%)

Table 6. Degree of parallelization on the Anatomy dataset in average busy time of the reasoning peers.

As we can see from the Table, quite a good degree of parallelism was achieved on the anatomy dataset. During the overall runtime of 55 seconds each of the two reasoners was busy for 45 seconds on average. This is a rate of more than 80% that indicates that indeed a significant part of the computation was executed in parallel leading to a further reduction of the overall runtime.

5.4. *Experiments on the OntoFarm Dataset*

A second set of experiments was carried out on the five ontologies from the OntoFarm Dataset mentioned above. The goal of this second experiment was to analyze the behavior of the method on a set of multiple small ontologies mutually connected through mappings, which is another typical situation faced on the semantic web. As a setting for this experiment, we considered the ontologies in alphabetical order starting with the CMT ontology, adding one ontology after another including mappings to all previously considered ontologies. Statistics of the different resulting configurations are given in Table 7.

We executed our reasoning method on the resulting ontologies and compared the localized with a distributed configuration. In this experiment, we did not consider the autoconfig setting any more as the first experiment clearly showed that configuring the reasoner to ordered resolution performs significantly better than the generic setting. Further, we did not consider upload time as a criterion as the ontologies involved are rather small, ranging from

	Number of Clauses					
		mapping to				
	ontology	cmt	confof	ekaw	iasted	Overall
cmt	228					228
confof	199	24				451
ekaw	256	19	40			766
iasted	587	8	18	20		1399
sigkdd	153	20	13	22	30	1637

Table 7. Size of the datasets used in the experiments in terms of number of clauses

two- to six hundred clauses which is not enough to produce meaningful numbers. Besides these differences we again measured the parameters identified in the introduction of this section, namely runtime, number of derived clauses, number of propagated clauses and average busy-time of the reasoning peers. In addition to the first experiment, we also look at how these parameters change with an increasing number of ontologies and reasoning peers which is an important issue with respect to scalability. Furthermore, we recorded runtime and busy-time not only for consistency tests but also for positive subsumption queries. Negative queries (i.e. queries answered with "No") do not vary much in runtime, the results can be expected to be similar to the consistency test [28]. A subsumption query tests if a given axiom $A_1 \sqsubseteq A_2$ is derivable from the ontology network by adding the assertional clauses $A_1(testinstance)$ and $\neg A_2(testinstance)$ to the appropriate clause sets. If the axiom follows from the ontology network, the query is positive and the query clauses produce a contradiction. For creating all positive queries, it is necessary to classify (i.e. compute the concept hierarchy of) the ontology first. Otherwise all pairs of concepts have to be tested which would take about two days. Since we did not yet implement classification for our distributed resolution method, we used the common tableau reasoner Pellet[e] [29] for classification. Classifying the whole ontology network merged with the mappings failed due to memory limitations, hence we only generated all positive subsumption queries of the two ontology networks consisting of *cmt, confof, ekaw* and *iasted, sigkdd* respectively. For each of the five ontology networks we used for the consistency test, we also tested the applicable positive subsumption queries obtained via classification and computed the average runtime and busy-time.

Runtime The first and most important aspect is the overall runtime of the system with respect to checking consistency of the overall knowledge base. As in the anatomy case, the ontologies considered are consistent which means that the knowledge base has to be completely saturated for the consistency test. The results of comparing the overall runtime of the system in a global setting where all inferences are carried out by a single reasoning peer and the distributed setting where each ontology is assigned to a dedicated reasoning

[e] http://clarkparsia.com/pellet/

Ontologies	saturation		positive query	
	global	distributed	global	distributed
cmt	0,24	0,24		
cmt+confof	0,32	0,24	0,04	0,05
cmt+confof+ekaw	1,09	0,52	0,06	0,09
cmt+confof+ekaw+iasted	11,22	2,54	0,20	0,12
cmt+confof+ekaw+iasted+sigkdd	13,50	3,59	0,24	0,15

Table 8. Runtime on the OntoFarm dataset in seconds

peer are compared in Table 8.

As we can see, the results of the experiments confirm the observations of the anatomy experiment in the sense that we observe a significant reduction in the overall runtime when distributing the inference process of the consistency test. While this reduction is rather minor for the small datasets it becomes more significant the more ontologies and reasoning peers are involved: while for three ontologies the distributed setting is about twice as fast as the global one in the case of four ontologies there is a speedup almost of factor five compared to the global setting. This speedup can again be explained by the reduction of the local search spaces reduces the time for local derivations and over-compensates the additional communication effort which in this case is much higher than for the anatomy case as we will see in the following. For positive queries, as expected, the absolute runtime and also the differences between global and distributed setting is much smaller. Here, the distributed computation is only faster if more than three reasoning peers are involved. For comparing our method with common tableau reasoning we ran Pellet on the merged ontology consisting of all ontology modules and mappings. The consistency check took about 3 seconds, the answer to a random positive query was returned after 1.7 seconds.

Number of Derived Clauses Looking at the number of derived clauses, we see an effect of the more expressive ontologies. While in the anatomy case distributed evaluation did not have an impact on the overall number of derived clauses this is not longer the case on the OntoFarm dataset. The results of the experiments with respect to this issue are summarized in Table 9.

Ontologies	global	distributed
cmt	2031	2031
cmt+confof	3726	3752
cmt+confof+ekaw	9586	9680
cmt+confof+ekaw+iasted	35809	53935
cmt+confof+ekaw+iasted+sigkdd	41023	74028

Table 9. Number of derived clauses on the OntoFarm dataset

We see that for a small number of ontologies (up to three) the overall number of derived clauses only increases by a small amount, but for the last two settings with four and five ontologies, a significant number of additional derivations can be observed. It is likely that this behavior will always be observed in systems with many ontologies. Currently, reduction mechanisms are applied to new clauses before they are sent and after they are received by a reasoning peer. As long as interactions are primarily between pairs of ontologies this strategy seems to be efficient. When more ontologies come into play, interactions will occur between multiple ontologies not all of which can be detected in the course of a bilateral communication. While the resulting increase of derivations were over-compensated by the savings due to smaller local search spaces in our experiments, a more effective strategy for detecting redundant clauses thus reducing the number of derived clauses is a major issue for future work.

Amount of Communication Concerning the amount of communication between reasoning peers the situation is similar to the observations made with respect to the overall number of derived clause. On contrast to the latter, we already observed the need for significant communication between reasoning peers in the anatomy experiments. Compared to the amount of communication in those experiments, the fraction of clauses that has to be send around is smaller. The concrete numbers are given in the Table below.

Ontologies	global	distributed	
cmt	0	0	(0%)
cmt+confof	0	288	(8%)
cmt+confof+ekaw	0	1466	(15%)
cmt+confof+ekaw+iasted	0	14624	(27%)
cmt+confof+ekaw+iasted+sigkdd	0	25217	(34%)

Table 10. Number of propagated clauses on the OntoFarm dataset

As Table 10 shows, the amount of communication needed for saturating the knowledge base increases withe the number of ontologies involved. We see a significant increase from three to four ontologies which can be explained by the steep increase of the number of clauses derived overall. Besides this effect also the fraction of clauses that have to be sent increases with the number of ontologies. Compared to the anatomy case where more than half of the clauses were sent in the distributed setting, this fraction is smaller for the OntoFarm dataset. This can be explained by the fact that there are more links between entities in the anatomy ontologies and the derivation of new knowledge almost always consisted of combining existing knowledge of the two ontologies. As the ontologies in the ontofarm dataset are more sparsely interlinked, a good share of new knowledge can already be derived inside the individual ontologies lowering the relative amount of knowledge exchanged between the ontologies. The use of more effective reduction strategies motivated above will also reduce the amount of clauses to be send. Beyond this, there is little chance

of improving the numbers given as the completeness of our reasoning method relies on the unambiguous assignment of inference steps to reasoning peers and previous experiments [26] showed that an assignment of inferences to reasoners based on partitioning techniques is not as efficient as the results achieved by using the natural distribution of knowledge which is the basis of the results in this paper.

Degree of Parallelization When more ontologies are concerned, the issue of parallelizability becomes critical as inference steps in one reasoning peer can rely on derivations taking place in other reasoners with no guarantee that the required results are available on time for the other reasoners to continue. The results of investigating this issue are shown in the Table below.

Ontologies	saturation		positive query	
cmt	0,24	(100%)		
cmt+confof	0,14	(59%)	0,05	(99%)
cmt+confof+ekaw	0,28	(54%)	0,09	(99%)
cmt+confof+ekaw+iasted	1,70	(67%)	0,12	(99%)
cmt+confof+ekaw+iasted+sigkdd	2,00	(56%)	0,14	(92%)

Table 11. Average busy-time of the reasoning peers in the distributed setting

The results of our experiments show that the degree of parallelization achieved on the consistency tests of the OntoFarm dataset does not reach the result on the anatomy dataset where a busy-time of more than 80% was observed. In these experiments, we observed average busy times of around 60% for all settings. This is less than for the anatomy dataset but still an acceptable result, especially because the average busy times do not seem to decrease with the number of ontologies. This means that the overall system becomes more effective with a growing number of peers as an increasing number of peers contributes to a solution of the problem which together with a stable average busy time leads to a linear increase of computational power available for solving the problem. Furthermore, the degree of paralellization is close to 100% for positive queries. This is not surprising, as the proof for the query is found long before the consistency test would be answered, at a time where most reasoning peers did not finish saturation of their initial clause set. This result is indeed important as it supports our claim that the method presented scales to larger problems.

5.5. *Summary of Results*

In summary, our experiments clearly show that the distributed reasoning method proposed in this paper consistently outperforms a globalized computation both in terms of upload- and runtime. In particular, we measured a runtime speedup by a factor of up to four for five reasoning peers. A closer look at the inference process revealed that the communication necessary in a distributed setting is significant – sometimes more than half of the derived

clauses were send to other reasoners – but the communication costs are over-compensated by the reduction of the local search spaces leading to faster local inference and a linear increase of the computational power available for solving the problem by parallel execution of the inference process.

6. Future Work

Based on the results of the experiments, we have identified the effective reduction of redundant clauses as a major target for further optimization as in the presence of multiple reasoning peers the overall number of derivations increases due to an incomplete elimination of redundant clauses. In addition, the implementation used for the experimental evaluation will be extended in serveral ways. First, the supported expressibility can be extended to $\mathcal{ALCHI}$ without losing completeness of the reasoning procedure. Furthermore, performance of our approach can be improved by optimizing the allocation of symbols and clauses dynamically. Finally, investigating variation in the precedence of symbols that determines the ordering of literals may reduce runtime and communication effort.

6.1. *Extension to More Expressive Description Logics*

The theoretical investigations and practical evaluation presented in this work applies distributed resolution to $\mathcal{ALC}$ ontologies. In this section we discuss the application of our method to more expressive ontologies.

Transitivity Transitivity axioms (e.g. *Trans(partOf)*) are used to declare that certain properties are transitive. These axioms can be eliminated by a well known transformation, reducing the expressivity of an to $\mathcal{ALC}$. Hence, with some preprocessing we can decide satisfiability of $\mathcal{ALC}$ ontologies with transitivity axioms, i.e. description logic $\mathcal{S}$. The transformation is polynomial in the size of the input, but the adaption to the distributed setting is not trivial. In contrast to the transformation of description logic axioms to first order clauses mentioned so far, the translation of transitivity depends on the whole ontology and not only on the transitivity axioms. Hence, the linked ontologies cannot be transformed independently.

Nominals Nominals are concepts with a single instance, e.g. the concept $\{$*Erdös*$\}$ with instance *Erdös* is used in the concept description $\exists$ *coauthorOf*$\{$*Erdös*$\}$. Nominals are replaced by common concepts for many applications: Each nominal $\{nom\}$ is replaced by a new concept *Nom* and the axiom *Nom(nom)* is added to the ontology. The restriction that *Nom* may not contain another instance is not expressible in description logic without nominals.

Datatypes Datatypes $(\mathcal{D})$ can be eliminated without changing the semantics by moving the datatypes into the abstract domain. In practice, sorts (datatype and abstract) are handled different from the other predicates to speed up reasoning. Built-in datatype predicates can

be added to support e.g. the *greater* relation between integers.

Cardinality Restrictions Cardinality restrictions are used to declare certain properties as functional or define e.g. a rich person is a person that owns at least three houses ($RichPerson \sqsubseteq Person \sqcap \exists_{\geq 3} owns.House$). If an ontology contains qualified ($\mathcal{Q}$) or unqualified ($\mathcal{N}$) cardinality restrictions or functional properties ($\mathcal{F}$), the corresponding set of first order clauses contains equality literals. To deal with these equalities, a more involved calculus than ordered resolution is necessary to decide satisfiability. A calculus that decides satisfiability for clauses obtained from $\mathcal{ALCHIQ}$ ontologies is the basic superposition calculus presented in [18]. But, the superposition rules can be applied to premises where the resolvable literals have different top symbols. Hence, extension of our approach in this direction is not straightforward. Distributed resolution for cardinalities topic is addressed in more detail in [27].

6.2. *Flexible Allocation*

In this work, we assumed the allocation of symbols is defined by the namespaces of the ontology terms. In previous work we showed that allocation for a monolithical ontology can be computed using graph partitioning methods [26, 25]. In both settings, the allocation is determined prior to the reasoning process but it is possible to extend the method to more flexible allocation of symbols. A flexible allocation has two advantages for reasoning performance: First, the computation load can be distributed among the peers more evenly. If one peer is very busy while another remains idle, symbols allocated to the busy peer can be reallocated to the idle peer. Second, communication can be reduced by dynamically optimizing the allocation appropriately. In theory, the allocation of a symbol can be changed at any point in the reasoning process. The reallocation procedure is similar to an update in a routing table. Even if the update is asynchronous and peers may temporary use different allocations, clauses are never deleted accidentally because peers that receive clauses they are not responsible for will just forward them according to their local allocation. To avoid propagating clauses back and forth, the peer New that becomes responsible for the reallocated symbol P should be the first to update the local allocation table. Then the peer Old that was responsible for P before changes his table and sends all clauses with resolvable literal top symbol P to New. The other peers are updated subsequently, if Old receives clauses that should be allocated to New they are passed on. It is also possible to introduce new reasoning peers by reallocating symbols or shutdown peers and reallocate their local clause set to another peer.

6.3. *Advanced Precedence*

The precedence of symbols that determines the order of literals and thereby also impacts the allocation of clauses has to be defined such that function symbol precede predicate symbols and negation is the smallest symbol. Apart from these requirements the precedence in our experiments was random. Since the lists of symbols were combined from seperate files, the precedence is the concatenation of the separate function and predicate lists of all ontologies.

This configuration can be seen as a baseline, optimizing the precedence of modules and the local precedences may decrease the number of propagations and the overall runtime.

7. Conclusions

In this paper, we have addressed the problem of distributing reasoning methods for ontologies across different reasoners on the web. Our focus was on distributing resolution reasoning and we showed that resolution can be distributed without losing completeness if resolvable clauses are always processed by the same reasoner. Based on this observation, we presented a sound and complete distributed resolution method for ontologies that builds upon existing work on resolution reasoning for $\mathcal{ALC}$ ontologies.

This work can be seen as a first step towards a highly scalable distributed reasoning architecture for the semantic web and is in line with recent efforts in building such a reasoning infrastructure[f]. Future work will be concerned with the extension of the approach to more expressive ontology languages as well as with the optimization of the reasoning approach by investigating orderings of ontology signatures that minimize the communication overhead.

Acknowledgment This work was partially supported by the German Science Foundation in the Emmy-Noether Program under contract Stu 266/3-1.

References

[1] Philippe Adjiman, Philippe Chatalic, Francois Goasdoué, Marie-Christine Rousset, and Laurent Simon. Distributed reasoning in a peer-to-peer setting: Application to the semantic web. *Journal of Artificial Intelligence Research*, 25:269–314, 2006.

[2] Eyal Amir and Sheila McIlraith. Partition-based logical reasoning for first-order and propositional theories. *Artificial Intelligence*, 162(1-2):49–88, 2005.

[3] Jie Bao, Doina Caragea, and Vasant Honavar. Tableau-based federated reasoning algorithm for modular ontologies. In *Web Intelligence*, 2006.

[4] Jie Bao, George Voutsadakis, Giora Slutzki, and Vasant Honavar. Package-based description logics. In Heiner Stuckenschmidt, Christine Parent, and Stefano Spaccapietra, editors, *Modular Ontologies*. Springer, 2009.

[5] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. Owl web ontology language reference, 2004.

[6] Oliver Bodenreider, Terry F. Hayamizu, Martin Ringwald, Sherri De Coronado, and Songmao Zhang. Of mice and men: Aligning mouse and human anatomies. In *Proceedings of the American Medical Informatics Association (AIMA) Annual Symposium*, 2005.

[7] Maria Paola Bonacina. The clause-diffusion theorem prover peers-mcd (system description). In *Proceedings of the 14th International Conference on Automated Deduction*, volume 1249 of *Lecture Notes In Computer Science*, pages 53 – 56, London, UK, 1997. Springer Verlag.

[8] Maria Paola Bonacina. A taxonomy of parallel strategies for deduction. *Annals of Mathematics and Artificial Intelligence*, 29(1–4):223–257, 2000. Published in February 2001.

[f] the Large Knowledge Collider: `http://www.larkc.eu/`

[9] Maria Paola Bonacina and Jieh Hsiang. Parallelization of deduction strategies: An analytical study. *J. Autom. Reasoning*, 13(1):1–33, 1994.
[10] Alex Borgida and Luciano Serafini. Distributed description logics: Assimilating information from peer sources. *Journal of Data Semantics*, 1:153–184, 2003.
[11] Susan E. Conry, Douglas J. MacIntosh, and Robert A. Meyer. Dares: A distributed automated reasoning system. In *Proc. AAAI-90*, pages 78–85, 1990.
[12] Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. Modular reuse of ontologies: Theory and practice. *J. of Artificial Intelligence Research (JAIR)*, 31:273–318, 2008.
[13] Bernardo Cuenca Grau, Bijan Parsia, and Evren Sirin. Combining owl ontologies using e-connections. *Journal Of Web Semantics*, 4(1), 2005.
[14] F. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in description logics. In G. Brewka, editor, *Principles of Knowledge Representation and Reasoning*, Studies in Logic, Language and Information, pages 193–238. CLSI Publications, 1996.
[15] Peter Haase and Yimin Wang. A decentralized infrastructure for query answering over distributed ontologies. In *Proceedings of The 22nd Annual ACM Symposium on Applied Computing (SAC)*, 2007.
[16] Andreas Harth, Jürgen Umbrich, Aidan Hogan, and Stefan Decker. Yars2: A federated repository for querying graph structured data from the web. In *ISWC/ASWC 2007*, pages 211–224, 2007.
[17] Terry Hayamizu, Mary Mangan, John Corradi, James Kadin, and Martin Ringwald. The adult mouse anatomical dictionary: a tool for annotating and integrating data. *Genome Biology*, 6(3):R29, 2005.
[18] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reasoning in Description Logics by a Reduction to Disjunctive Datalog. *Journal of Automated Reasoning*, 39(3):351–384, 2007.
[19] Zoi Kaoudi, Iris Miliaraki, and Manolis Koubarakis. RDFS reasoning and query answering on top of dhts. In *International Semantic Web Conference*. Springer, 2008.
[20] George Kokkinidis, Lefteris Sidirougos, and Vassilis Christophides. Query processing in RDF/S-based P2P database systems. In Staab and Stuckenschmidt [30].
[21] Ewing L. Lusk, William W. McCune, and John Slaney. Roo: A parallel theorem prover. In *In Proceedings of the 11th CADE*, volume 607 of LNAI, pages 731–734. Springer Verlag, 1992.
[22] C. Lutz, D. Walther, and F. Wolter. Conservative extensions in expressive description logics. In *Twentieth International Joint Conference on Artificial Intelligence IJCAI-07*, 2007.
[23] Boris Motik. *Reasoning in Description Logics using Resolution and Deductive Databases*. PhD thesis, Universität Karlsruhe (TH), Karlsruhe, Germany, January 2006.
[24] Eyal Oren, Spyros Kotoulas, George Anadiotis, Ronny Siebes, Annette ten Teije, and Frank van Harmelen. Marvin: A platform for large-scale analysis of semantic web data. In *Proceedings of the International Web Science Conference*, 2009.
[25] Anne Schlicht and Heiner Stuckenschmidt. A flexible partitioning tool for large ontologies. In *International Conference on Web Intelligence and Intelligent Agent Technology (WI/IAT)*, 2008.
[26] Anne Schlicht and Heiner Stuckenschmidt. Towards distributed ontology reasoning for the web. In *International Conference on Web Intelligence and Intelligent Agent Technology (WI/IAT)*, 2008.
[27] Anne Schlicht and Heiner Stuckenschmidt. Distributed resolution for expressive ontology networks. In *Web reasoning and rule systems : Third International Conference, RR 2009*, 2009.
[28] Anne Schlicht and Heiner Stuckenschmidt. Peer-to-peer reasoning for interlinked ontologies. *International Journal of Semantic Computing, Special Issue on Web Scale Reasoning*, 2010 (to be published).
[29] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet:

A practical OWL DL reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007.

[30] Steffen Staab and Heiner Stuckenschmidt, editors. *Semantic Web and Peer-to-Peer: Decentralized Management and Exchange of Knowledge and Information*. Springer Verlag, 2006.

[31] Heiner Stuckenschmidt, Richard Vdovjak, Geert-Jan Houben, and Jeen Broekstra. Index structures and algorithms for querying distributed rdf repositories. In *WWW 2004*, pages 631–639, 2004.

[32] Tanel Tammet. *Resolution methods for Decision Problems and Finite Model Building*. PhD thesis, Chalmers University of Technology and University of Göteborg, 1992.

[33] Christoph Tempich, Steffen Staab, and Adrian Wranik. Remindin': semantic query routing in peer-to-peer networks based on social metaphors. In *WWW 2004*, pages 640–649, 2004.

[34] D. Tsarkov, A. Riazanov, S. Bechhofer, and I Horrocks. Using vampire to reason with owl. In *International Semantic Web Conference (ISWC)*, 2004.

[35] Ondřej Šváb, Vojtěch Svátek, Petr Berka, Dušan Rak, and Petr Tomášek. Ontofarm: Towards an experimental collection of parallel ontologies. In *International Semantic Web Conference*. Springer, 2005.

[36] Christoph Weidenbach. Combining superposition, sorts and splitting. In Alan Robinson und Andrei Voronkov, editor, *Handbook of Automated Reasoning*, volume II, chapter 27. Elsevier, 2001.

[37] Christoph Weidenbach, Uwe Brahm, Thomas Hillenbrand, Enno Keen, Christian Theobalt, and Dalibor Topić. SPASS version 2.0. In Andrei Voronkov, editor, *Automated deduction - 18th International Conference on Automated Deduction*. Springer, 2002.