Semantic Matchmaking of Resources with Local Closed-World Reasoning

Stephan Grimm[1] and Pascal Hitzler[2]

[1]FZI Research Center for Information Technologies, Germany
stephan.grimm@fzi.de
[2]Institute AIFB, University of Karlsruhe, Germany
hitzler@aifb.uni-karlsruhe.de

Abstract

Ontology languages like OWL allow for semantically rich annotation of resources, such as products advertised at an electronic online marketplace, while the Description Logic (DL) formalism underlying OWL provides reasoning techniques to perform matchmaking on such annotations. We identify peculiarities in the use of DL inferences for matchmaking which are due to the open-world semantics of OWL, and we analyse the use of local closed-world reasoning for its applicability to matchmaking. In particular, we investigate two nonmonotonic extensions to DL, namely autoepistemic DLs and DLs with circumscription, for their suitability of realising local closed-world reasoning in the matchmaking context to overcome these problems. We discuss their different characteristics by means of an elaborate example of an electronic marketplace for PC product catalogues from the eCommerce domain and demonstrate how these formalisms can be used to realise such scenarios.

1 Introduction

With the advent of the Semantic Web [5], resources of all kinds are being annotated with semantically rich meta data for their automated access through the web. For semantic annotation, languages are used that are based on knowledge representation paradigms to encode a "meaningful" characterisation of a resource into a formal description, aiming for automation through machine-understandability. Matchmaking of semantically annotated resources is concerned with the task of verifying compatibility among resources by looking at their semantic descriptions. At an electronic marketplace, for example, supply and demand of advertised products are matched to find compatible offers for requests. Other matchmaking scenarios comprise the discovery of Web Services by semantic descriptions of their functionality or the automated support for recruitment by semantic descriptions of job offers and applicant's profiles.

Matchmaking relies on the availability of a knowledge representation language which provides appropriate means for expressing the relevant information. One of the most prominent choices is the Web Ontology Language OWL [26], which has been recommended by the World Wide Web Consortium (W3C) in

2004. OWL is essentially based on *description logic* (DL) [3], a logic-based formalism with well understood computational and representational properties. It thus allows for logical reasoning, and matchmaking over OWL-DL can be operationalised by techniques of automated deduction. As a fragment of first-order predicate logic, OWL inherits a semantics that adheres to the *open-world assumption* (OWA), under which situations of incomplete information can be handled by the distinction between negative knowledge and the absence of knowledge. At the same time, it also inherits features which make it awkward for representing certain kinds of knowledge. In particular, OWL does not allow for any form of closed-world or default reasoning in such situations.

The quest for alterations or extensions of OWL to make it more suitable for important modelling tasks is currently one of the prominent research issues in ontology language research. While such investigations are frequently pursued on the basis of formal logical argumentation, the natural way of deriving requirements from application needs is often neglected. In this paper, we therefore investigate the usability of OWL for the modelling of matchmaking problems. In particular, we will point out that a pure open-world semantics leads to unintuitive behaviour in certain cases. Going one step further, we will study recent approaches to weakening the open-world assumption in OWL by means of local closed-world features [13], and examine them with respect to the matchmaking requirements we have identified.

This paper thus serves different purposes. It identifies problems due to the open-world assumption in an elaborate discussion of an example taken from the eCommerce domain. It elaborates on the proposal of using a local closed-world semantics to overcome these problems, based on the ideas in [18]. It presents an application of two particular nonmonotonic extensions to DL, namely autoepistemic DL and DL with circumscription, to the problem of matchmaking with respect to their suitability for realising local closed-world reasoning in this context. The detailed example demonstrates how the constructs of these formalisms can be applied to realise matchmaking in a particular scenario. The overall presentation is based on arguments at an intuitive level, however, the interested reader will also find the necessary formal details required to map the intuitive argumentation to the underlying details of the logics used. Readers not primarily interested in the model-theoretic realisation of matchmaking can skip these technicalities without missing the main messages conveyed.

The paper is structured as follows. In Section 2 we will briefly introduce OWL and approaches to local closed-world reasoning with the DL underlying OWL, namely autoepistemic description logics and description logics with circumscription. In Section 3 we will show how resources for matchmaking are modelled in OWL, and in particular we will introduce a large running example which will help us to illustrate our discussion. In Section 4 we show how DL inferencing is used for matching resource descriptions, and we will identify deficiencies of modelling based on the open world assumption. The forms of local closed-world reasoning introduced in Section 2 will then in Section 5 be analysed as to their ability to rectify the identified problems. Section 6 discusses related work, and in Section 7 we will summarise and point to concrete research issues which need to be addressed in the future.

The authors acknowledge support by the German Federal Ministry of Education and Research (BMBF) under the SmartWeb project (01 IMD01 B), and by the Deutsche Forschungsgemeinschaft (DFG) under the ReaSem project.

2 Description Logics and their Nonmonotonic Extensions

The Web Ontology Language OWL [26] is being recommended as a web standard by the World Wide Web Consortium (W3C) since 2004. Since then, OWL has found a multitude of uses, not only on the web, but for knowledge representation in general. In essence OWL, or more precisely its most important variant OWL-DL, is based on *description logic*, or DL for short. DLs have been developed out of the effort to lay strict formal foundations to semantic networks and frame systems, and they combine a rigourous semantics based on first-order predicate logic with an intuitive way of structuring and encoding conceptual knowledge.

In the following we give an introduction to the description logic underlying OWL-DL, and also to two nonmonotonic extensions dealing with local closed-world reasoning, namely autoepistemic DL and circumscriptive DL. We will keep this introduction on an intuitive level to make the paper accessible to readers not familiar with the technical details of logics. On the other hand, we additionally present those formalities which the interested reader requires to connect the intuition in arguments to the underlying model-theoretic semantics. Full details of OWL-DL can be found in [3, 26]. For presentation of OWL-DL statements and ontologies we will use *DL-syntax*, as it is most convenient and the easiest to read for the purposes of this paper.

2.1 Description Logics

The basic elements used to represent knowledge in the description logic formalism are *concepts*, *roles*, and *individuals*. Intuitively, concepts denote classes of things, such as *Computer* or *OperatingSystem*. Roles denote relationships between things, such as *hasComponent* or *runsOS*. Individuals denote instances, such as *DeepBlue* or *WindowsXP*. There are two types of roles, namely abstract roles like *hasComponent*, which relate individuals to individuals, and concrete roles like *capacity*, which assign an element of a concrete datatype $\mathcal{D}$ – in this case a number – to an individual.

Starting from a set of concept names (called *atomic* or *named concepts*), a set of role names, and a set of individual names, complex concept expressions can be formed using *concept constructors* in a nested way. Let us give a few brief examples to illustrate some of the DL concept constructors. $\mathit{Computer} \sqcap \mathit{MobileDevice}$ stands for all mobile computers. $\exists\, \mathit{hasComponent}.\mathit{DVDDrive}$ denotes all things which have a DVD drive as a component. $\mathit{Computer} \sqcap \forall\, \mathit{runsOS}.\neg \mathit{WindowsOS}$ stands for all computers that do not run a Windows operating system. $\mathit{Laptop} \sqcap \leq 3\, \mathit{hasComponent} \sqcap \exists\, \mathit{memory}.\geq_{512}$ denotes all laptops which have at least 3 components and 512 MB main memory or more. $\mathit{Component} \sqcap \exists\, \mathit{supports}^{-}.\{\mathit{WindowsXP}\}$ stands for all components that are supported by WindowsXP.

Formally, the description logic underlying OWL-DL is named $\mathcal{SHOIN}(\mathbf{D})$ and allows for construction of complex concepts according to the following grammar, where A is an atomic concept, p is a primitive abstract role, r is a possibly inverse abstract role, s is a concrete role, d is a concrete domain predicate, a_i

are individuals, c_i are elements of a datatype, and n is a non-negative integer.

$$\begin{array}{rcl} C & \rightarrow & A \mid \bot \mid \top \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \exists r.C \mid \forall r.C \mid \geq n\, r \mid \leq n\, r \\ & & \mid \{a_1, \ldots, a_n\} \mid \exists s.D \mid \forall s.D \mid \geq n\, s \mid \leq n\, s \\ D & \rightarrow & d \mid \{c_1, \ldots, c_n\} \\ r & \rightarrow & p \mid p^- \end{array}$$

An OWL-DL ontology consists of statements that represent the *axioms* of a corresponding DL knowledge base *KB* composed of a *TBox* and an *ABox*. The TBox describes terminological knowledge, and TBox axioms comprise concept inclusions of the form $C \sqsubseteq D$ which state subsumption between two concepts C and D. For example, the axiom $\mathit{WindowsPC} \sqsubseteq \mathit{Computer} \sqcap \exists\, \mathit{runsOS}.\mathit{WindowsOS}$ states that any Windows PC is a computer that runs some Windows operating system. Another kind of TBox axiom is a concept equivalence of the form $C \equiv D$, which is a shortcut for the two inclusions $C \sqsubseteq D$ and $D \sqsubseteq C$. For example, the axiom $\mathit{Laptop} \sqcup \mathit{PocketPC} \equiv \mathit{Computer} \sqcap \mathit{MobileDevice}$ states that laptops together with pocket PCs form just the family of mobile computers. Furthermore, inclusion can also be stated for roles, as in the axiom $\mathit{hasGfx} \sqsubseteq \mathit{hasComponent}$, which says that having a graphics card implies having a component.

The ABox, on the other hand, describes assertional knowledge, and ABox axioms comprise concept assertions of the form $C(a)$, which state membership of an individual a to a concept C, and role assertions of the form $r(a, b)$, which relate two individuals a and b via the role r. For example, the axiom $\mathit{Laptop}(\mathit{MyComputer})$ states that my computer is a laptop, while the axiom $\mathit{runsOS}(\mathit{MyComputer}, \mathit{WindowsXP})$ says that my computer runs WindowsXP.

Reasoning with OWL-DL ontologies rests on the model-theoretic semantics of description logics, which is given by means of *interpretations*. Formally, an interpretation $\mathcal{I}$ is a mapping from the concept, role and individual names into sets $\Delta^{\mathcal{I}}$, called *domains of interpretation*. Individuals, for example, directly map to objects in the interpretation domain, while concepts are interpreted as subsets and roles are interpreted as binary relations, which are called their *extensions*, respectively. Table 1 specifies this semantics technically in form of conditions for the various constructors, which an interpretation $\mathcal{I}$ must satisfy. (The semantics of role axioms and concrete role constructs has been omitted due to brevity – we refer to [3, 26] for a full definition.) Intuitively, an interpretation specifies

$$\begin{array}{rcl} \top^{\mathcal{I}} = \Delta^{\mathcal{I}} \quad , \quad \bot^{\mathcal{I}} & = & \emptyset \\ A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \quad , \quad r^{\mathcal{I}} & \subseteq & \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \\ (C \sqcap D)^{\mathcal{I}} & = & C^{\mathcal{I}} \cap D^{\mathcal{I}} \\ (C \sqcup D)^{\mathcal{I}} & = & C^{\mathcal{I}} \cup D^{\mathcal{I}} \\ (\neg C)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\ (\forall\, r.C)^{\mathcal{I}} & = & \{a \in \Delta^{\mathcal{I}} \mid \forall b.(a, b) \in r^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\} \\ (\exists\, r.C)^{\mathcal{I}} & = & \{a \in \Delta^{\mathcal{I}} \mid \exists b.(a, b) \in r^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \\ (\geq n\, r)^{\mathcal{I}} & = & \{a \in \Delta^{\mathcal{I}} \mid \#\{b \in \Delta^{\mathcal{I}} \mid (a, b) \in r^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \geq n\} \\ (\leq n\, r)^{\mathcal{I}} & = & \{a \in \Delta^{\mathcal{I}} \mid \#\{b \in \Delta^{\mathcal{I}} \mid (a, b) \in r^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \leq n\} \\ \{a_1, \ldots, a_n\}^{\mathcal{I}} & = & \{a_1^{\mathcal{I}}, \ldots, a_n^{\mathcal{I}}\} \\ (r^-)^{\mathcal{I}} & = & \{(b, a) \mid (a, b) \in r^{\mathcal{I}}\} \end{array}$$

Table 1: Model-theoretic semantics for part of $\mathcal{SHOIN}(\mathbf{D})$.

a particular arrangement of objects in the interpretation domain in terms of membership in concept and role extensions. For example, in one interpretation an individual as *MyComputer* can be in the extension of the concept *Laptop*, in another interpretation it can be in the extension of the concept *DesktopPC*, and in yet another one it can be in the extensions of both. If the arrangement of objects in an interpretation $\mathcal{I}$ is in accordance with the axioms in a knowledge base *KB* then $\mathcal{I}$ is called a *model* of *KB*. Formally, $\mathcal{I}$ is a model of *KB* if it satisfies the conditions $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, $C^{\mathcal{I}} = D^{\mathcal{I}}$, $r_1^{\mathcal{I}} \subseteq r_2^{\mathcal{I}}$, $a^{\mathcal{I}} \in C^{\mathcal{I}}$, $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$ for all concept inclusions, concept equivalences, role inclusions, concept assertions and role assertions in *KB*, respectively. We will use the symbol $\mathcal{M}(KB)$ to denote the set of all models of *KB*.

Even after filtering out those interpretations that do not satisfy all axioms, a knowledge base *KB* has in general a multitude of models in which things are interpreted differently. For example, if we did not say in our ontology whether a particular graphics adapter is supported by a Windows operating system or not, there are models in which it is and such in which it is not. In such a case, we say that we have incomplete knowledge about the support of graphics adapters by Windows operating systems, which is captured by the situation of multiple models. Contrarily, if we explicitly list all graphics adapters with support by Windows operating systems and also state that these are the only ones, we say that we have complete knowledge about Windows support for graphics adapters, which is reflected by *KB* having only models in which the supported graphics adapters is exactly our explicit list. This style of semantics is also referred to as "open-world" semantics, since we do not assume to have full knowledge about the domain of discourse by just the axioms given – unless explicitly encoded.

Reasoning with OWL-DL ontologies is now based on the following standard DL reasoning tasks, defined for a DL knowledge base *KB*.

- *Knowledge base satisfiability*: *KB* is *satisfiable* if it has a model.
- *Concept satisfiability*: a concept C is *satisfiable* with respect to *KB* if there exists a model $\mathcal{I} \in \mathcal{M}(KB)$ in which the extension $C^{\mathcal{I}}$ of C is non-empty.
- *Instance checking*: an individual a is an instance of a concept C with respect to *KB*, i.e. $KB \models C(a)$, if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ holds for all models $\mathcal{I} \in \mathcal{M}(KB)$.
- *Subsumption*: a concept C is subsumed by a concept D with respect to *KB*, i.e. $KB \models C \sqsubseteq D$, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for all models $\mathcal{I} \in \mathcal{M}(KB)$.

Intuitively, knowledge base satisfiability checks an ontology for consistency, concept satisfiability checks whether a concept can have instances, instance checking tests an individual to be an instance of a concept, and subsumption tells us whether a concept is in general more specific than another one.

2.2 Autoepistemic Description Logics

We now introduce the first nonmonotonic extension of description logics which we will study in this paper. It is based on the idea of autoepistemic logic [22], which allows for introspection of knowledge bases, i.e. to express knowledge about what the knowledge base *knows*. We follow [10], where the basic DL $\mathcal{ALC}$ was extended by an autoepistemic knowledge operator $\mathbf{K}$, yielding the autoepistemic description logic $\mathcal{ALCK}$. In [28], epistemic operators have also

been incorporated into more expressive DLs that capture features of OWL-DL. The **K**-operator can be applied as an additional constructor to both concepts and roles, and can intuitively be paraphrased as "known to be". These autoepistemic extensions allow for local closed-world reasoning [13] and a logical reconstruction of non-monotonic features of frame-based knowledge representation systems, such as concept and role closure, defaults, integrity constraints and procedural rules [28].

To understand the intuition behind the **K**-operator, consider the knowledge base $\mathcal{KB} = \{\mathit{Application}(\mathit{XOffice}), \mathit{runsUnder}(\mathit{XOffice}, \mathit{RedHat})\}$, and the concept $D = \mathit{Application} \sqcap \exists\, \mathit{runsUnder}.\neg \mathit{WindowsOS}$, which can be paraphrased as "applications which run under an operating system other than Windows". Since $\mathcal{KB}$ does not say whether *RedHat* is a Windows operating system or not, *XOffice* is not in the extension of D in general. On the contrary, consider the autoepistemic concept $D' = \mathit{Application} \sqcap \exists\, \mathbf{K}\mathit{runsUnder}.\neg\mathbf{K}\,\mathit{WindowsOS}$, which can be intuitively paraphrased as "applications which are *known* to run under an operating system *not known* to be Windows". Based on the facts in $\mathcal{KB}$, we cannot derive that *RedHat* is a Windows operating system. Therefore, *RedHat* is *not known* to be a Windows operating system, and thus, *XOffice* is in the extension of D' in all models of $\mathcal{KB}$.

The formal semantics of autoepistemic DLs is defined similarly to that of classical DL introduced before, with the difference that interpretations are replaced by *epistemic interpretations* $(\mathcal{I}, \mathcal{W})$, which are pairs of an interpretation $\mathcal{I}$ and a set of interpretations $\mathcal{W}$ seen as "possible worlds". Ordinary concepts and roles are interpreted just as in the classical case shown in Table 1, while epistemic concepts and epistemic roles are interpreted by intersecting the extensions of their non-epistemic counterparts over all possible worlds, as follows.

$$(\mathbf{K}C)^{\mathcal{I},\mathcal{W}} = \bigcap_{\mathcal{J}\in\mathcal{W}} C^{\mathcal{J},\mathcal{W}} \quad , \qquad (\mathbf{K}r)^{\mathcal{I},\mathcal{W}} = \bigcap_{\mathcal{J}\in\mathcal{W}} r^{\mathcal{J},\mathcal{W}}$$

An epistemic concept $\mathbf{K}C$ is interpreted as the set of all individuals which belong to the concept C in all interpretations in $\mathcal{W}$. In this way, applying **K** to a concept C produces the set of objects which are members of C in all possible worlds, i.e. which are definitely *known to be* members of C. Similarly, an epistemic role $\mathbf{K}r$ is interpreted as the pairs of individuals that belong to the role r in all possible worlds.

The only reasoning task we will consider in autoepistemic DLs is the satisfiability of an epistemic concept C with respect to a non-epistemic knowledge base $\mathcal{KB}$, i.e. one with no occurrence of epistemic operators in it. In this special case, the set $\mathcal{W}$ coincides with the set $\mathcal{M}(\mathcal{KB})$ of all classical models of $\mathcal{KB}$ [11]. Thus, the extension of an epistemic concept (or role) in an epistemic interpretation contains only those (pairs of) individuals which necessarily must be in there from the given knowledge, considering all models of $\mathcal{KB}$. Hence, the respective reasoning task of concept satisfiability is defined as follows:

> C is satisfiable with respect to $\mathcal{KB}$ if and only if there is an interpretation $\mathcal{I} \in \mathcal{M}(\mathcal{KB})$ such that $C^{(\mathcal{I},\mathcal{M}(\mathcal{KB}))} \neq \emptyset$.

2.3 Circumscriptive Description Logics

Besides autoepistemic logic, circumscription – due to McCarthy [21] – is another prominent historic approach to local closed-world reasoning. In essence, it rests on the idea of closing knowledge by enforcing certain extensional minimality conditions on interpretations. We follow the recent work [7] which incorporates these ideas into description logics. The actual description logic used in this case was $\mathcal{ALCQIO}$, which is basically $\mathcal{SHOIN}(\mathbf{D})$ without the use of datatypes and role hierarchies, but extended with a slight modification of the $\geq$- and $\leq$-constructors.

In contrast to autoepistemic DL, there is no additional language construct like the **K**-operator in circumscriptive DL. Instead, an external *circumscription pattern* is used which gives specifics on the extensional minimisation to be performed. As a simplification of [7], for our purposes a circumscription pattern is a triple $\mathsf{CP} = (M, F, V)$ where M, F, and V are mutually disjoint sets of concept and role names called the *minimised*, the *fixed* and the *varying* predicates, respectively. (For convenience, we denote concept and role names as predicates.) Reasoning is then performed on a *circumscribed knowledge base* $\mathrm{circ}_{\mathsf{CP}}(\mathit{KB})$, which is a classical knowledge base KB together with a circumscription pattern CP.

Intuitively, minimisation of the predicates in M restricts their extensions to contain only those (pairs of) individuals, for which there is evidence in the knowledge base to be contained in the extension. During minimisation, the extensions of predicates in F are fixed while those of predicates in V vary freely. Fixation of predicates is a way to restrict minimisation, preventing certain predicates from being affected, while varying predicates do not get a special treatment. Sometimes we will omit the set V in circumscription patterns, and in this case V is understood as containing all predicates not mentioned in either M or F.

To give an example of reasoning with circumscribed knowledge bases, consider the following knowledge base and circumscription pattern.

$$\begin{aligned} \mathit{KB} &= \{\ \mathit{Laptop} \sqsubseteq \mathit{Computer}, \mathit{Computer} \sqsubseteq \mathit{Hardware}, \\ &\qquad \mathit{Application} \sqcap \exists\, \mathit{runsUnder}\,.\,\mathit{LinuxOS}(\mathit{XOffice})\ \} \\ \mathsf{CP} &= (M = \{\mathit{Hardware}, \mathit{Laptop}, \mathit{Application}, \mathit{LinuxOS}\}, F = \{\mathit{Computer}\}). \end{aligned}$$

The concept *Laptop* is unsatisfiable with respect to the circumscribed knowledge base $\mathrm{circ}_{\mathsf{CP}}(\mathit{KB})$. The reason is that there is no evidence for the existence of any laptop in KB, and hence the extension of the minimised concept *Laptop* is restricted to the empty set. On the other hand, the concept *Computer* is satisfiable, as it is fixed, although it is subsumed by the minimised *Hardware*. Also the concept *Hardware* is satisfiable, since its extension contains that of the fixed *Computer*. Furthermore, the concept *Application* is satisfiable although it is minimised. The reason is that there is an individual, *XOffice*, explicitly asserted to this concept, which is evidence for the existence of an application in KB. Even the concept *LinuxOS* is satisfiable, for which there is no such explicit assertion. The reason is that there is evidence for the existence of an unknown object that XOffice runs under and that is a Linux operating system.

Technically, minimisation in circumscriptive DLs is realised by introducing a preference relation $<_{\mathsf{CP}}$ which allows to compare interpretations in terms of their extensions for minimised predicates. The preferred models of a knowledge base KB are those minimal in the extensions of minimised predicates. Formally,

$\mathcal{J} <_{\mathsf{CP}} \mathcal{I}$ holds if the following conditions are satisfied: (i) $\Delta^{\mathcal{J}} = \Delta^{\mathcal{I}}$, (ii) $a^{\mathcal{J}} = a^{\mathcal{I}}$ for all individuals a, (iii) $p^{\mathcal{J}} = p^{\mathcal{I}}$ for all $p \in F$, (iv) $p^{\mathcal{J}} \subseteq p^{\mathcal{I}}$ for all $p \in M$, (v) there is some $p \in M$ such that $p^{\mathcal{J}} \subset p^{\mathcal{I}}$. The models of a circumscribed knowledge base $\text{circ}_{\mathsf{CP}}(\mathit{KB})$ are those minimal with respect to $<_{\mathsf{CP}}$, such that for reasoning with $\text{circ}_{\mathsf{CP}}(\mathit{KB})$ only the preferred models of KB are taken into account. Conditions (i) and (ii) say that only interpretations are compared which share the same domain of interpretation and also the same assignment of individuals. Condition (iii) requires comparable interpretations to also be equal with respect to the extensions of fixed predicates, such that minimisation is done for each such extension separately. Conditions (iv) and (v) assure that $\mathcal{J}$ is actually "smaller" than $\mathcal{I}$ in the extension of some minimised predicate, while it is not "bigger" in the extension of any other minimised predicate.

3 Modelling Resources in Description Logics

Semantic Web languages are designed for the annotation of resources of all kinds, allowing to describe them in terms of the vocabulary defined in a domain specific ontology. In the context of matchmaking, both requested and advertised resources are checked for compatibility by matching their semantic annotation. The notion of resource is very generic and can range from ticket offers and hotel bookings by travel agencies over skill profiles at job markets to the functional descriptions of Web Services. In this paper we consider product descriptions in an eCommerce setting as an example for resources, in particular within the domain of PC product catalogues. However, the techniques we use for description and matchmaking are domain independent and generally apply to any kind of resource.

Structured descriptions of resources typically characterise a resource by specifying values for its properties or restrictions on them. For example, a description of a PC that is offered in an electronic advertisement could specify a desktop PC with 1024 MB of main memory having a graphics adapter that supports dual screen, while the notions of PC, main memory, etc. are predefined in a referred to domain ontology. The values of resource properties can themselves be complex resources with subproperties. So, the graphics adapter, for example, can itself be specified in terms of its components and attributes. In this sense, a resource description specifies an arbitrarily nested data object by restricting its parameters in a graph structure. Resource descriptions taken from other domains could specify a job applicant of age 25 with a master degree in Artificial Intelligence and with programming skills in object oriented languages, or the booking of a hotel room in the city of Madrid that is air-conditioned and within 1 km distance to the railway station.

Resource Classes as DL Concepts

Ontology languages used for semantic annotation typically distinguish between *instances*, which represent concrete objects in the domain, and *concepts*, which represent classes of objects grouping together instances that have certain properties in common. A particular PC "Compaq Presario V6133EU" with an 80GB hard drive and "NVIDIA GeForce Go 6150" graphics adapter is typically modelled as an instance, while the class of all Laptops with integrated wireless

network abilities is typically modelled as a concept.

When supply and demand for PC products meet at an electronic market place, both requesters and providers of PCs describe their resources in terms of concepts rather than instances, abstracting from the concrete objects in the PC domain. On the provider side, a description in terms of instances would overload the supplier's product catalogue with listing all the various combinations of advertised PC variants explicitly. For example, a supplier that offers PCs with different kinds of hard disks varying in capacity and multiple options for an onboard graphics adapter that run either Windows or Linux operating systems and come with any application pre-installed, would have to include an entry for any combination of parameters that they support. Instead, they want their product catalogue to have a manageable amount of entries, each representing a class of PCs with restrictions on certain properties that leave some options. On the requester side, a description in terms of instances would require the demander to explicitly specify all the properties of the desired product, even if they are indifferent about some of them. Instead they express their demand in form of a concept, requesting, for example, a Laptop with integrated wireless network adapter and with at least 512 MB main memory, no matter which particular network adapter or which kind of graphics card or other unspecified feature is finally delivered.

Description Logic is particularly suited to represent resources by restricting their properties using complex concept expressions that denote abstract classes of objects. In a way similar to [17, 18], we show how we map the structured description of abstract resource classes to the DL modelling constructs introduced in Section 2 by an example within the eCommerce PC product catalogue scenario.

Figure 1 shows a resource description in form of a DL concept expression, either issued by a provider who supplies a PC, or by a requester who demands a PC from an electronically available product catalogue. The concept expression R represents the class of PCs with at least 512 MB of main memory whose graphics adapter have a DVI output.

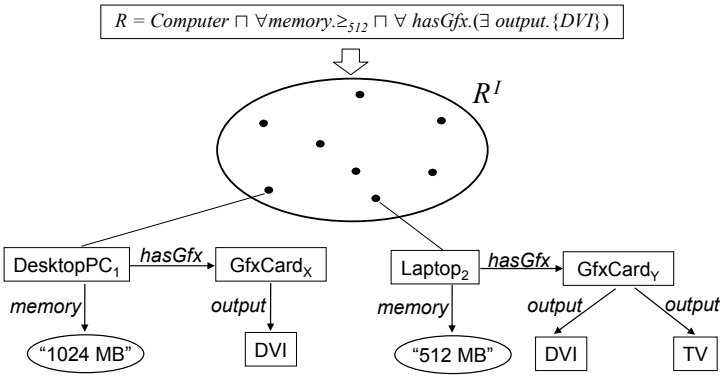


Figure 1: Resource description as DL concept expression

The concepts, individuals and roles that occur in the complex expression of R, such as *Computer*, *DVI* or *hasGfx*, originate from a domain ontology about PCs, which both requesters and providers refer to in their descriptions. Given a model-theoretic interpretation $\mathcal{I}$, the extension $R^{\mathcal{I}}$ of the resource concept represents all the concrete resources that are captured by the description on the instance level. Figure 1 exemplifies two such concrete resource instances: a desktop PC with 1024 MB main memory and DVI enabled graphics adapter, and a laptop with 512 MB main memory and a DVI graphics adapter that also supports TV output. The set $R^{\mathcal{I}}$ can contain many other such instances that fit in the requirements of the concept expression in R.

To ensure that a particular resource instance is captured by a description, the concept expression in R has to be chosen such that there exists an interpretation $\mathcal{I}$ which is a model of the concept expressed through R with respect to the background domain ontology and in which the particular instance is contained in the extension $R^{\mathcal{I}}$. More strictly, the concept expression in R can even require certain resource instances to be included in all its models. To exclude a particular resource instance from being captured by the description, the expression in R has to be chosen such that there is no model of the concept expressed through R in which this resource instance is contained in $R^{\mathcal{I}}$.

The variance in the properties of a resource description maps to the existence of multiple interpretations that are models of the resource concept and is connected to the open-world assumption made in the DL formalism, as described in [17]. On the one hand, the open-world semantics allows us to describe resources in form of compact descriptions expressing constraints on possible instances, which avoids listing all intended resources explicitly and allows for incomplete descriptions. On the other hand, it requires us to carefully include additional restrictions whenever some particular instances are to be ruled out. For example, if we don't specify in a description of a PC whether the graphics adapter should support TV output or not, both the alternatives are captured, since in some models it does while in others it does not. If we prefer one alternative over the other, we have to adjust our resource description such that it either explicitly requires or explicitly disclaims the support of TV output.

Example Scenario

Here, we introduce a scenario within the domain of PC product catalogues in eCommerce, which we use as a running example to demonstrate matchmaking throughout the paper.

Consider an electronic marketplace where demand and supply for PCs meet in form of electronically advertised resource descriptions. Providers advertise their product catalogues with offers for PC configurations, each represented by a supply resource description S in form of a DL concept. Requesters issue a demand D, also represented by a resource description in form of a DL concept, which specifies their desired PC configuration. All the configurations advertised or requested may have various options in their parameters. In their descriptions, all parties refer to an agreed upon domain ontology O_{PC} for computers that defines basic notions such as "Hardware", "Operating System" or "Application" and their interrelation. They also refer to vendor specific information about concrete components or software, such as "WindowsXP" or particular graphics adapters, defined in an ontology O_{ven}. Furthermore, each party adds its own

customary ontology, e.g. O_A for provider A, which is based on the formerly described, more general ontologies. Altogether, these ontologies form a knowledge base *KB* that is the basis to perform matchmaking of the supplies and demands within the electronic marketplace.

Example 1 (Example on PC Product Catalogue)

$$
\begin{aligned}
O_{PC} = \{\ & \mathit{Computer} \sqsubseteq \mathit{Hardware},\ \mathit{Component} \sqsubseteq \mathit{Hardware},\ \mathit{Computer} \sqsubseteq \neg \mathit{Component},\\
& \mathit{Computer} \sqsubseteq \forall\, \mathit{hasComponent}\,.\,\mathit{Component} \sqcap\ \leq 1 \mathit{hasGfx} \sqcap\ = 1\, \mathit{runsOS},\\
& \mathit{Laptop} \sqsubseteq \mathit{Computer},\ \mathit{DesktopPC} \sqsubseteq \mathit{Computer},\ \mathit{Laptop} \sqsubseteq \neg \mathit{DesktopPC},\\
& \mathit{GfxCard} \sqsubseteq \mathit{Component} \sqcap\ \neg \mathit{StorageDevice} \sqcap\ \forall\, \mathit{hasOutput}\,.\,\{\mathit{Analog}, \mathit{DVI}, \mathit{TV}\},\\
& \mathit{StorageDevice} \sqsubseteq \mathit{Component},\ \mathit{hasGfx} \sqsubseteq \mathit{hasComponent},\ \mathit{PocketPC} \sqsubseteq \mathit{Computer},\\
& \mathit{DualScreenGfxCard} \sqsubseteq \mathit{GfxCard} \sqcap\ \geq 2\, \mathit{hasOutput},\ \mathit{hasStorage} \sqsubseteq \mathit{hasComponent},\\
& \mathit{RAIDStorage} \sqsubseteq \mathit{StorageDevice} \sqcap\ \forall\, \mathit{capacity}\,.\,\geq_{40},\\
& \mathit{Application} \sqsubseteq \mathit{Software},\ \mathit{OpSys} \sqsubseteq \mathit{Software},\ \mathit{Application} \sqsubseteq \neg \mathit{OpSys},\\
& \mathit{WindowsOS} \sqsubseteq \mathit{OpSys},\ \mathit{LinuxOS} \sqsubseteq \mathit{OpSys},\ \mathit{WindowsOS} \sqsubseteq \neg \mathit{LinuxOS},\\
& \mathit{OpSys} \sqsubseteq \forall\, \mathit{supports}\,.\,\mathit{Component},\ \mathit{Computer} \sqsubseteq \forall\, \mathit{runsOS}\,.\,\mathit{OpSys},\\
& \mathit{EmbeddedOS} \sqsubseteq \mathit{OpSys} \sqcap\ \neg \exists\, \mathit{supports}\,.\,\mathit{RAIDStorage}\quad \}
\end{aligned}
$$

$$
\begin{aligned}
O_{ven} = \{\ & \mathit{DualScreenGfxCard}(\mathit{2ScreenCard}),\ \mathit{hasOutput}(\mathit{2ScreenCard}, \mathit{DVI}),\\
& \mathit{GfxCard}(\mathit{ViewTV}),\ \mathit{hasOutput}(\mathit{ViewTV}, \mathit{TV}),\ \mathit{GfxCard}(\mathit{LuxCard}),\\
& \leq 1\, \mathit{hasOutput}(\mathit{LuxCard}),\ \mathit{hasOutput}(\mathit{LuxCard}, \mathit{Analog}),\\
& \mathit{supports}(\mathit{WindowsXP}, \mathit{2ScreenCard}),\ \mathit{supports}(\mathit{WindowsCE}, \mathit{2ScreenCard}),\\
& \mathit{supports}(\mathit{WindowsXP}, \mathit{ViewTV}),\ \mathit{supports}(\mathit{RedHat}, \mathit{LuxCard}),\ \mathit{LinuxOS}(\mathit{RedHat})\\
& \mathit{WindowsOS}(\mathit{WindowsXP}),\ \mathit{WindowsOS} \sqcap \mathit{EmbeddedOS}(\mathit{WindowsCE}) \qquad \}
\end{aligned}
$$

$$
\begin{aligned}
O_A = \{\ & \mathit{DVIDualScreenGfxCard} \equiv \mathit{DualScreenGfxCard} \sqcap\ \exists\, \mathit{hasOutput}\,.\,\{\mathit{DVI}\},\\
& \mathit{MiniTower} \sqsubseteq \mathit{DesktopPC} \sqcap\ \forall\, \mathit{hasStorage}\,.\,(\forall\, \mathit{capacity}\,.\,\leq_{20}),\\
& \mathit{GfxCard}(\mathit{NoNameGfxCard})\quad \}
\end{aligned}
$$

$$
O_B = \{\ \mathit{WorkStation} \equiv \exists\, \mathit{hasComponent}\,.\,(\mathit{StorageDevice} \sqcap\ \forall\, \mathit{capacity}\,.\,\geq_{40})\quad \}
$$

$$
\begin{aligned}
D_1 &= \mathit{Computer} \sqcap\ \exists\, \mathit{hasGfx}\,.\,\mathit{DualScreenGfxCard}\\
&\qquad \sqcap\ \forall\, \mathit{hasComponent}\,.\,(\exists\, \mathit{supports}^{-}\,.\,\mathit{WindowsOS})\\
D_2 &= \mathit{DesktopPC} \sqcap\ \exists\, \mathit{hasStorage}\,.\,\mathit{RAIDStorage}\\
&\qquad \sqcap\ \exists\, \mathit{runsOS}\,.\,(\exists\, \mathit{supports}\,.\,\mathit{DualScreenGfxCard}\\
&\qquad\qquad \sqcap\ \exists\, \mathit{supports}\,.\,\mathit{RAIDStorage})
\end{aligned}
$$

$$
\begin{aligned}
S_{A_1} &= \mathit{MiniTower} \sqcap\ \exists\, \mathit{hasGfx}\,.\,\mathit{DVIDualScreenGfxCard}\\
S_{A_2} &= \mathit{PocketPC} \sqcap\ \exists\, \mathit{hasGfx}\,.\,\{\mathit{NoNameGfxCard}\} \sqcap\ \forall\, \mathit{runsOS}\,.\,\{\mathit{WindowsCE}\}\\
S_{A_3} &= \mathit{DesktopPC} \sqcap\ \exists\, \mathit{hasGfx}\,.\,(\exists\, \mathit{hasOutput}\,.\,\{\mathit{TV}\}) \sqcap\ \forall\, \mathit{runsOS}\,.\,\{\mathit{WindowsXP}\}\\
S_{B_1} &= \mathit{Laptop} \sqcap\ \forall\, \mathit{hasComponent}\,.\,(\exists\, \mathit{supports}^{-}\,.\,\mathit{LinuxOS})\\
S_{B_2} &= \mathit{Workstation} \sqcap\ \exists\, \mathit{hasGfx}\,.\,(\leq 1\, \mathit{hasOutput}) \sqcap\ \forall\, \mathit{runsOS}\,.\,\mathit{LinuxOS}
\end{aligned}
$$

Domain Knowledge. The ontology O_{PC} distinguishes computers and their components on the hardware side. Computers are split into laptops and desktop PCs, while representatives for hardware components in the example are graphics adapters and storage devices with properties such as output or capacity. On the software side, O_{PC} speaks about operating systems that support hardware components and distinguishes Linux from Windows systems. By means of more complex axioms, O_{PC} imposes restrictions on the introduced taxonomy, stating,

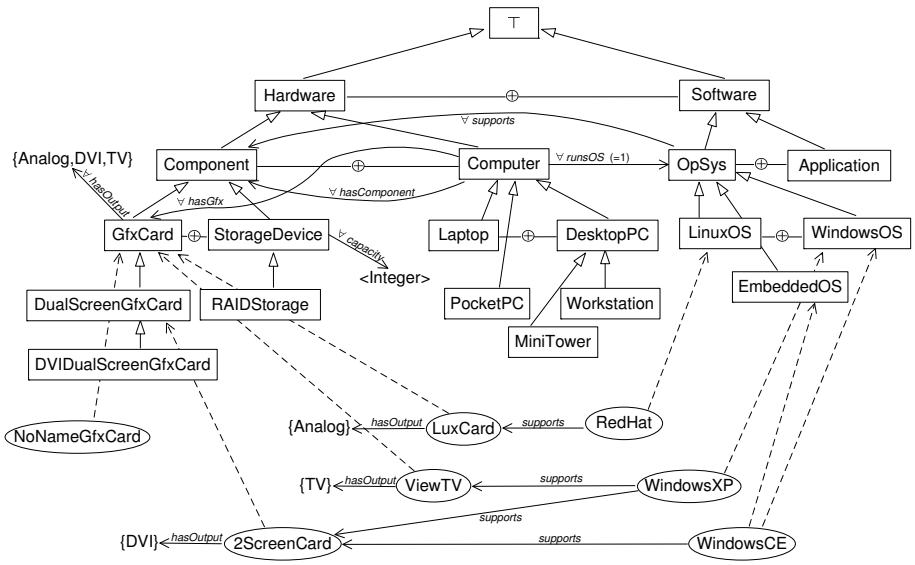


Figure 2: Example Ontology

for example, that embedded operating systems do not support RAID storage devices or that only cases are considered in which a computer runs exactly one operating system. While O_{PC} describes general knowledge about PCs, O_{ven} contains the vendor specific information about concrete hardware and software. For example, it provides information about a concrete graphics adapter named "2ScreenCard" which supports dual screen and which has a digital DVI output. It also reflects the explicit support of operating systems that vendors claim for their hardware components. For example, "2ScreenCard" is stated to have support by WindowsXP and WindowsCE. Furthermore, the providers A and B bring in their own ontologies for the definition of new concepts which they use in their resource descriptions. The ontology O_A, for example, introduces a concept for dual screen graphics adapters that support DVI output, which is then used in the supply S_{A_1}. Figure 2 visualises some of the domain knowledge captured in these ontologies.

Supply and Demand. In the example, there are two providers, named A and B, who advertise supply descriptions at the electronic marketplace as part of their product catalogues.

Provider A offers three different configurations for PCs. In supply S_{A_1} they describe a mini tower PC which has a dual screen enabled graphics adapter with DVI output. In their ontology O_A they introduce mini towers as desktop PCs with storage devices of a capacity smaller than 20 GB. With supply S_{A_2} they describe a pocket PC with some no name graphics adapter that runs WindowsCE. Notice that the no name graphics adapter is not one specified by any vendor in O_{ven} but is introduced by the provider in O_A. Finally, in S_{A_3} they describe a desktop PC that has a graphics adapter with TV output and that

runs Windows XP.

Provider B offers two different configurations for PCs, specialising on Linux machines. In supply S_{B_1} they describe a laptop all of whose components are supported by a Linux operating system. In supply S_{B_2} they describe a workstation with a single output graphics adapter that runs Linux.

In the first demand issued to the electronic marketplace, D_1, a requester describes a computer with dual screen graphics adapter all of whose components are supported by a Windows operating system. Suppose this demand has been issued by a company that installs variants of the Windows operating system on any PC they buy in a customary way. They want to make sure that the installed dual screen graphics adapter is supported by some Windows variant.

Suppose the requester of the second demand, D_2, already has a dual screen graphics adapter which they want to install into the requested PC themselves after delivery. They additionally plan to upgrade the system with a RAID storage device to use it as a server. To make sure that matching offers are compatible with the dual screen graphics adapter and with RAID storage, they ask for a desktop PC that runs an operating system which has support for both dual screen and RAID storage.

4 Matching Resource Descriptions with DL Inferencing

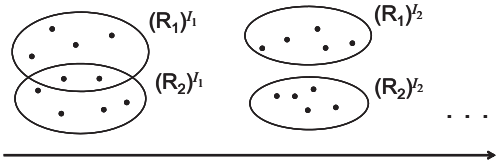
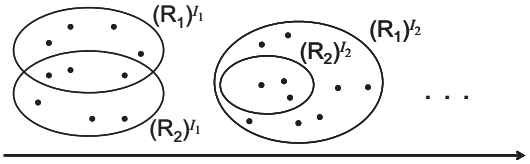
Matchmaking of semantically annotated resources in general aims towards checking for compatibility among resources. The compatibility between two annotated resources is verified by matching their semantic descriptions. In our setting of abstract descriptions that represent resource classes, two resource descriptions match if the sets of concrete resources they capture do overlap. In particular, if the two sets intersect then there are some concrete resources which are captured by both of the descriptions, which justifies their compatibility.

4.1 DL Inferences for Matching

Based on the resource descriptions expressed as DL concepts, matching can be performed by using DL inference services, such as satisfiability or entailment. Different reasoning services can be used to check for different forms of overlap between two sets of intended resources, by looking at the concept expressions and their models with respect to a domain ontology. In our presentation of matching under local closed-world reasoning in Section 5, we focus on intersection as the most basic form of overlap, based on the ideas in [14, 30] and [29]. However, for a better understanding of the intuition behind intersection matching, and for sake of completeness, we also present other forms of matching here. We review various inferences that have been used for matching in the literature, each one having its characteristic intuition by answering a specific question about the resources involved. We denote by $\mathcal{KB}$ a DL knowledge base that contains the axioms of a domain ontology with respect to which matching is evaluated.

Intersection Matching. As first proposed in [14, 29], the idea behind intersection matching is to check whether two resource descriptions, R_1 and R_2, share some intended concrete resource which is captured by both of them. Technically, this is the case if, in an interpretation $\mathcal{I} \in \mathcal{M}(KB)$, the extensions of R_1 and R_2 have a non-empty intersection, i.e. $R_1^{\mathcal{I}} \cap R_2^{\mathcal{I}} \neq \emptyset$. Under the open-world semantics of DL there can be many interpretations $\mathcal{I}$ that are models of KB, and therefore, intersection matching can be carried out in two ways, as depicted in Table 2.

Table 2: DL Inferences for Intersection Matching

Inference:	*Satisfiability of Concept Conjunction*
Input:	domain knowledge KB, resource descriptions R_1, R_2
Formula:	$R_1 \sqcap R_2$ is satisfiable w.r.t. KB
Situation:	
Intuition:	Is there a way to resolve unspecified issues such that R_1 and R_2 capture some common concrete resource?
Inference:	*Entailment of Concept Non-Disjointness*
Input:	domain knowledge KB, resource descriptions R_1, R_2
Formula:	$KB \cup \{R_1 \sqcap R_2 \sqsubseteq \bot\}$ is unsatisfiable
Situation:	
Intuition:	Do R_1 and R_2 specify some common concrete resource, regardless of how unspecified issues are resolved?

The inference *satisfiability of concept conjunction*, illustrated in the upper part of Table 2, can be realised by the standard DL reasoning task of checking whether a concept is satisfiable with respect to a knowledge base. The concept here forms the conjunction of the concept expressions used in the two resource descriptions. The intuition behind this inference is to check whether there is some way in which the given, possibly incomplete knowledge can be interpreted such that the two descriptions share an intended resource, while in other ways of interpretation they might not. Technically, this intuition exactly maps to satisfiability of the conjunction $R_1 \sqcap R_2$, which checks for the existence of an interpretation $\mathcal{I} \in \mathcal{M}(KB)$ in which $R_1^{\mathcal{I}} \cap R_2^{\mathcal{I}}$ is non-empty.

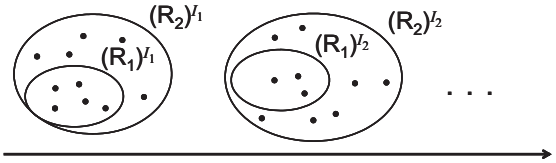
The inference *entailment of non-disjointness* as introduced in [17], illustrated in the lower part of Table 2, does not directly map to a standard DL reasoning task but can be realised by adding a disjointness axiom to KB and checking the resulting knowledge base for satisfiability. The additional axiom states the disjointness between the concept expressions used in the two resource descriptions. The intuition behind this inference is to check whether the two descriptions share an intended resource no matter in which way the given, pos-

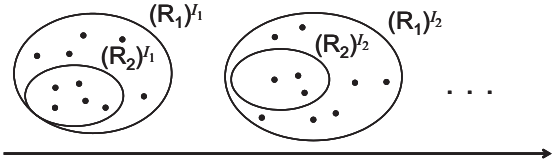
sibly incomplete knowledge is interpreted. Technically, this is the case if in all interpretations $\mathcal{I} \in \mathcal{M}(KB)$ the intersection $R_1^{\mathcal{I}} \cap R_2^{\mathcal{I}}$ is non-empty.

While satisfiability of concept conjunction is a rather weak check, entailment of non-disjointness is quite strict in that it requires the modelers of resource descriptions to exclude any interpretation in which an intended resource is not captured, which is usually achieved by including additional restrictions into the resource description. It is an attempt to cope with some problems in matching due to the open-world semantics, which we discuss in the following, and for which we propose a better solution through support of local-closed world reasoning, based on earlier work in [18]. For this reason, we focus on satisfiability of concept conjunction and mean this inference when we speak of intersection matching in the sequel.

Subsumption Matching. Another form of overlap between sets of intended resources is full containment of one set in the other, which maps to subsumption reasoning in DL. The idea behind subsumption matching is to check whether a description R_1 is a specialisation (or generalisation) of another description R_2. In the literature about matchmaking, the two directions of subsumption have been denoted by "Plugin" and "Subsumes" [25, 20], as illustrated in Table 3.

Table 3: DL Inferences for Subsumption Matching

Inference:	*Entailment of Concept Subsumption* (Plugin)
Input:	domain knowledge KB, resource descriptions R_1, R_2
Formula:	$KB \models R_1 \sqsubseteq R_2$
Situation:	
Intuition:	Do the concrete resources captured by R_2 encompass the concrete resources captured by R_1, regardless of how unspecified issues are resolved?

Inference:	*Entailment of Concept Subsumption* (Subsumes)
Input:	domain knowledge KB, resource descriptions R_1, R_2
Formula:	$KB \models R_2 \sqsubseteq R_1$
Situation:	
Intuition:	Do the concrete resources captured by R_1 encompass the concrete resources captured by R_2, regardless of how unspecified issues are resolved?

Subsumption matching is realised by the standard DL reasoning task of checking for the entailment of an inclusion axiom $R_1 \sqsubseteq R_2$ in case of a Plugin

match, or $R_2 \sqsubseteq R_1$ in case of a Subsumes match. The intuition behind subsumption matching is to check whether any intended concrete resource captured by the description R_1 is also captured by the description R_2, or the other way round, no matter in which way the given, possibly incomplete knowledge is interpreted. Technically, if the inclusion axiom is entailed this means that for all interpretations $\mathcal{I} \in \mathcal{M}(\mathcal{KB})$ the containment $R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$ respectively $R_2^{\mathcal{I}} \subseteq R_1^{\mathcal{I}}$ holds.

A special form of subsumption matching occurs if subsumption holds in both directions, i.e. $\mathcal{KB} \models R_1 \equiv R_2$, which has been called exact match in the literature [29, 14]. In this particular case, the sets of intended resources of two descriptions is identical.

Discussion of Matching Inferences. In the matchmaking literature [25, 24, 20], the different matching inferences have been used to establish a ranking among matched resource descriptions according to the following order of strictness.

$$fail \prec intersect \preceq subsume - plugin \preceq exact$$

Such an order is justified by the observation that some matches are implied by others. For example, an exact match always implies the subsumption matches, while each of the subsumption matches implies an intersection match.

Based on non-standard DL reasoning services, there have also been proposed other inferences for matching, which aim for a more fine-grained ranking of resource descriptions by approximating the logical notions of concept satisfiability and subsumption. Namely in [9], the use of concept contraction and concept abduction has been introduced to rank both intersection matches and non-matches in different ways. Concept contraction is used to identify those parts of the two concepts in resource descriptions that prevent their conjunction from being satisfiable. Concept abduction is used to identify those parts of a concept in a resource description that prevent it from being subsumed by the concept in another resource description.

In our setting, however, we are not concerned with the ranking of resource descriptions but set the focus on achieving the desired matching behaviour with the standard inferences. Based on the intuition behind the abstract resource descriptions introduced in Section 3, we add a different aspect to the discussion of the interrelation between the various matching inferences. Namely, we give an intuitive reading to various of the inferences, identifying particular questions they give answers to in terms of sets of intended resources and incompleteness in resource descriptions. We claim that the relation between the different types of match depends on the actual setting in which matchmaking is applied and that our intuitive reading of the inferences helps to better decide which type of matching to use, giving a more fine-grained account of their interrelation than a general ordering does.

For example, if a requester is indifferent about the options left open in their demanded PC configuration, they will not rate a subsumption based match with an entry in a provider's product catalogue higher than a mere intersection match, since they are already satisfied by the common concrete resources that determine the choices for the options in a particular way, although some other choices they agree on might be not supported on the provider side. On the other hand, if the requester has preferences about the options in their demanded

configuration then they might prefer a plugin match over an intersection match, since the plugin match ensures that any choice of options on the requester side is supported by the advertised supply, which allows for selecting their preferred choice.

In our presentation of matchmaking we focus on eliminating undesired positive matches due to open-world semantics, which inherently occur in the most basic form of matching by intersection. On the other hand, forms of matching that are based on entailment can easily be too strict, missing desired positive matches in various cases, as argued in [18, 17]. Therefore, we only consider satisfiability of concept conjunction as an inference for matching in the following, and we interpret options left open in resource descriptions as equal alternatives without taking into account preferences of requesters or providers. We define a boolean function $match(\mathcal{KB}, R_1, R_2)$, that takes as input two resource descriptions and a knowledge base containing all the relevant domain knowledge referred in the descriptions, as follows.

$$match(\mathcal{KB}, R_1, R_2) = \begin{cases} \mathsf{true}\,; & R_1 \sqcap R_2 \text{ satisfiable w.r.t. } \mathcal{KB} \\ \mathsf{false}\,; & \text{otherwise} \end{cases}$$

4.2 Counterintuitive Matching Behaviour due to OWA

On the one hand, open-world semantics allows for a proper handling of incomplete resource descriptions with variance in their parameters on both the requester and the provider side. On the other hand, it requires modelers to state "negative" information, such as disjointness or class non-membership, explicitly, as they do not hold by default. Based on the example scenario introduced in Section 3, we illustrate some cases in which the open-world semantics leads to an undesired behaviour of the intersection matching inference.

Intersection Matching and the Open-World Assumption

Satisfiability of concept conjunction is the weakest check for the compatibility of two resource descriptions in that it only fails if the descriptions impose contradictory constraints on resources that cannot be jointly fulfilled [17]. Technically, the existence of a single model of the involved knowledge base in which the two descriptions share a captured resource is sufficient. If neither the descriptions nor the background domain knowledge impose sufficient restrictions on the resources involved then such a model is likely to exist, leading to a false positive match in cases where incompatibility was assumed to hold by default.

To give an example, consider the PC product catalogue scenario from Section 3 with the domain ontology O_{PC} that describes basic notions about computers. For a demand $D = \mathit{Laptop}$, requiring a laptop computer, and for a supply $S = \mathit{DesktopPC}$, offering a desktop PC, $match(O_{PC}, D, S)$ would yield a negative result, as expected, since the two concepts Laptop and $\mathit{DesktopPC}$ are declared to be disjoint in O_{PC}. However, for the supply $S' = \mathit{Refrigerator}$, $match(O_{PC}, D, S')$ would yield an undesired positive result, since in O_{PC} there is no evidence that refrigerators and laptops are incompatible. There is some model $\mathcal{I}$ in which the sets $\mathit{Laptop}^{\mathcal{I}}$ and $\mathit{Refrigerator}^{\mathcal{I}}$ have a common element. This shows that the use of intersection matching in such a setting heavily relies on the agreement of the involved parties on carefully modelled domain ontologies. It is not robust

against deviations from the underlying conceptual model, for which it is likely to produce false positive matches. (In [15, 27] an industrial logistics scenario in the context of Semantic Web Service discovery has been realised based on this form of intersection matching with careful modelling of domain ontologies.)

In some cases, such false positive matches could be ruled out by including additional information in form of closure axioms to the domain knowledge or the resource descriptions. However, this is often a tedious task and overloads resource descriptions and domain models. To rule out S', disjointness between refrigerators and computers could be added, but there can be many more such concepts and in a less controlled environment where ontologies from various independent sources are combined disjointness statements can easily be missed.

On the other hand, one could think of using a formalism with pure closed-world semantics, such as a deductive database or logic programming system. However, this would, in turn, hamper the use of abstract descriptions of resources by restricting their properties. For some of the concepts and roles in the domain model, such as *Computer* or *hasComponent*, modelers would not like to list their instances explicitly, but make use of incomplete descriptions instead. There is, for example, no single known computer in the domain model, but still the extension of the concept *Computer* is not necessarily empty, as it would be under closed-world semantics.

Based on earlier work in [18], we propose to use a local closed-world semantics as a better solution to the problem, which allows to assume closure for particular parts of the domain model. In the following, we investigate the problematic cases in which some form of closure is desired in our PC product catalogue example before we elaborate on local closure in Section 5. While in a similar setting in [9] deficiencies of matchmaking with classical DL inferencing have already been investigated, our work differs in that we focus on situations where matches due to standard inferencing are not detected as desired. Contrarily, the work in [9] uses forms of approximation to achieve a more fine-grained ranking of matches that can already be "correctly" detected by means of standard inferences.

Cases of Undesired Matching Behaviour

Consider the example ontologies, supplies and demands from Section 3. For each of the demands D_X, we check to which of the supplies S_Y it matches, applying the previously defined matching function $match(KB, D_X, S_Y)$. The knowledge base KB, with respect to which matching is evaluated, is built up of general domain knowledge about computers and vendor information as well as the knowledge which the two parties bring in.[1]

$$KB := O_{PC} \cup O_{ven} \cup O_Y$$

Notice that the various parties know about the domain ontologies used within the electronic marketplace but that they typically don't know about the modelling of their competitors or potential business partners.

We subsequently investigate the matching of the two demands D_1 and D_2 against all available supplies, identifying cases in which the open-world semantics of the DL formalism shows to be problematic.

[1] In our example, the requesters do not bring in their own ontologies, although in the general case they could.

Matching Results for Demand D_1. The demand D_1 asks for computers with a dual screen graphics adapter and with components all supported by a Windows operating system. Since *hasGfx* is a subrole of *hasComponent*, the demanded graphics adapter needs to be supported by Windows in particular.

The first supply of provider A matches this demand, i.e. $match(KB, D_1, S_{A_1}) =$ true. This meets our intuition, since the desktop PCs offered in S_{A_1} are special kinds of computers and the offered type of dual screen graphics card with additional DVI output also specialises the one requested. Moreover, among the concrete graphics adapters known to the electronic marketplace through O_{ven} there is actually one which is supported by a Windows operating system, as requested. Technically, the positive match holds because there is an interpretation $\mathcal{I} \in \mathcal{M}(KB)$ in which there exists an element c in both the extensions $Computer^{\mathcal{I}}$ and $DesktopPC^{\mathcal{I}}$ that is attached via *hasGfx* to an element g that is in both $DualScreenGfxCard^{\mathcal{I}}$ and $(\exists\, hasOutput.\{DVI\})^{\mathcal{I}}$, and that is in turn attached to an element $o \in WindowsOS^{\mathcal{I}}$ via *supports*. The elements $c, g, o \in \Delta^{\mathcal{I}}$ reflect the concrete resources for a computer, graphics adapter and operating system within the interpretation $\mathcal{I}$, while g and o could map to $2ScreenCard^{\mathcal{I}}$ and $WindowsXP^{\mathcal{I}}$, respectively. An issue of incomplete knowledge within KB is, for example, whether *2ScreenCard* is the only dual screen adapter or whether there are others. There are models of KB in which *ViewTV* is dual screen as well, since this has not been explicitly excluded. This means that there are several ways to interpret this incompleteness all of which result in a graphics adapter captured by both demand and supply. However, even if we would add a closure axiom $DualScreenGfxCard \equiv \{2ScreenCard\}$ to KB, ensuring that the only dual screen graphics adapter is the one known to be, S_{A_1} would still yield a positive match, since with the concrete situation in the ABox for the closed part *DualScreenGfxCard* the conjunction $D_1 \sqcap S_{A_1}$ can be satisfied.

The second supply of provider A, which offers desktop PCs with a particular no name graphics adapter, also matches the demand, i.e. $match(KB, D_1, S_{A_2}) =$ true. Since the knowledge in KB does not determine whether *NoNameGfxCard* is dual screen or not, it can be interpreted in such a way that it is and in such a way that it is not. This is a typical situation of incomplete information in which provider A has simply not specified all details for this particular graphics adapter. As a requester, we would be rather sceptical about *NoNameGfxCard* meeting the requirement specified in D_1, even if there is some way to resolve incomplete knowledge in which it does. Intuitively we would therefore like to eliminate this positive match by imposing some form of closure to the part about dual screen graphics adapters in the demand. A closure axiom that would eliminate the positive match with S_{A_2} is, for example, $\neg DualScreenGfxCard(NoNameGfxCard)$, however, this seems to be quite cumbersome since there can be many more graphics adapters for which we would like to state such non-membership. Moreover, this closure axiom could not even be included by the demander, since the individual *NoNameGfxCard* is an element that provider A has introduced in their ontology. If the demander would use the closure axiom $DualScreenGfxCard \equiv \{2ScreenCard\}$ instead, then this would prevent any suppliers from introducing new graphics adapters that support dual screen.

Also for the third supply advertised by provider A, offering desktop PCs that have a graphics adapter with TV output, we get a positive match, i.e. $match(KB, D_1, S_{A_3}) =$ true. Since dual screen graphics adapters are in general not excluded from having a TV output, the knowledge in KB can be interpreted

in such a way that there is a graphics adapter with both the two features, and thus, the conjunction $D_1 \sqcap S_{A_3}$ is satisfiable. Looking at the particular situation of concrete graphics adapters in the ABox of *KB*, however, we would intuitively rather like S_{A_3} to not match the demand because there is actually no known dual screen graphics adapter which is stated to have TV output. If we would ask the knowledge base for such a graphics adapter by means of the query $(\mathit{DualScreenGfxCard} \sqcap \exists\, \mathit{hasOutput}\,.\{\mathit{TV}\})(?x)$ in a closed-world data base sense, we would get an empty result. This undesired positive match could be eliminated by adding the closure axiom $\exists\, \mathit{hasOutput}\,.\{\mathit{TV}\} \sqsubseteq \neg \mathit{DualScreenGfxCard}$ but this would lead to a contradictory knowledge base whenever some provider introduces their own dual screen graphics adapter with TV output.

Another supply that yields a positive match with D_1 is S_{B_1}, which offers laptops with components all supported by Linux. Again, there is a way to interpret the knowledge in *KB* such that some graphics adapter has support from both Linux and Windows, and thus, the conjunction $D_1 \sqcap S_{B_1}$ is satisfiable, yielding $\mathit{match}(\mathit{KB}, D_1, S_{B_1}) = \mathsf{true}$. However, there is no graphics adapter known in *KB* which has support of both Linux and Windows. Also here, we would intuitively prefer a closed-world interpretation of the role *supports*, expecting that a hardware component is not supported by a particular operating system if this is not explicitly stated by a vendor in O_{ven}. The introduction of explicit closure information in form of additional axioms would result in a similar situation as in the former case.

Finally, the supply S_{B_2} offering workstations with single output graphics adapters yields a negative match, i.e. $\mathit{match}(\mathit{KB}, D_1, S_{B_2}) = \mathsf{false}$, which meets our intuition. Indeed, the requested dual screen graphics adapter needs to have at least two outputs by the knowledge in *KB*, which contradicts its restriction to a single output in the supply.

Matching Results for Demand D_2. The demand D_2 asks for RAID-enabled desktop PCs that run an operating system which supports dual screen graphics adapters and RAID storage devices.

The first supply of provider A does not match this demand, since the advertised mini towers are restricted to storage devices with a maximum capacity of 20 GB in O_A while the capacity of the requested RAID storage system is stated to be at least 40 GB in O_{PC}. Therefore, the conjunction $D_2 \sqcap S_{A_1}$ is unsatisfiable and $\mathit{match}(\mathit{KB}, D_2, S_{A_1}) = \mathsf{false}$, as desired.

Also the second supply of provider A does not match the demand D_2, since the embedded operating system WindowsCE, which it requires the advertised pocket PC to run, does not support the requested RAID storage, according to O_{PC}. As intuitively expected, we get $\mathit{match}(\mathit{KB}, D_2, S_{A_2}) = \mathsf{false}$.

The third supply of provider A, offering desktop PCs with TV out graphics adapters that run WindowsXP, does yield a positive match for demand D_2. Even if we would interpret the support of graphics adapters in a closed-world sense, we would intuitively expect this match to hold, since there is actually a known graphics adapter, namely "ViewTV", that has TV output and is supported by WindowsXP according to the knowledge in O_{ven}. Notice that we would not like to interpret the support of storage devices, required in the demand, in a closed-world sense because the domain knowledge is such that no concrete storage devices occur in form of explicit instances.

The first supply advertised by provider B does not match the demand, i.e. $match(\mathcal{KB}, D_2, S_{B_1}) = \mathsf{false}$, simply because the offered laptops are incompatible with the requested desktop PCs by the disjointness stated in O_{PC}.

Finally, for the second supply of provider B, requiring PCs to run Linux, we also get a positive match $match(\mathcal{KB}, D_2, S_{B_1}) = \mathsf{true}$, since there is a way to interpret the knowledge in $\mathcal{KB}$ such that some Linux operating system supports both dual screen graphics adapters and RAID storage. Again, by the concrete situation of the known operating systems mentioned in the ABox of $\mathcal{KB}$, we could be sceptical about this way of interpretation because there is no concrete Linux operating system stated to have these two features. Intuitively, we would rather like to interpret at least the support of graphics adapters in a closed-world sense, since vendors state the support of their graphics adapters in O_{ven} explicitly. In this sense, we would prefer supply S_{B_2} to not match demand D_2.

Observations. In summary, we have seen that in the resource descriptions for supply and demand at the electronic market place in the example, requesters and providers of PC products make use of incomplete descriptions when specifying their intended PC configuration. Things like computers or storage devices, for which the domain model does not list explicit instances, are described in an abstract way by restricting their properties, such as components or capacity. On the other hand, for some knowledge in the domain model explicitly listed instances are present in form of ABox assertions. For example, information about concrete graphics adapters and their support by operating systems is explicitly stated by vendors. For this part of the domain knowledge, we would sometimes prefer a closed-world view, allowing to conclude that, for example, a graphics adapter is *not* supported by a particular operating system if this is not explicitly stated.

In many of the cases of matching supply and demand in the example, the open-world semantics under which intersection matching is performed results in positive matches where we would intuitively expect matching to fail. We have seen that the modification of the domain knowledge by means of additional closure axioms does not provide a satisfying solution to the problem, since it is tedious and likely to introduce inconsistencies in the knowledge base. Rather, we would like to have a means to identify certain parts of the domain model to be interpreted in the closed-world sense dynamically, while the open-world semantics is maintained for the rest of the domain model. In the following section, we show how this can be achieved by using nonmonotonic extensions to the description logic formalism.

5 Improved Matching with Local Closed-World Reasoning

As a fragment of first-order predicate logic, description logics inherit a semantics based on the open-world assumption, under which no conclusion is drawn unless there is sufficient evidence. On the contrary, there are formalisms, such as logic programming or deductive database systems, which adhere to the closed-world assumption, under which some conclusions are drawn if there is no counter-evidence. A generalisation of both is the so called *local closed-world assump-*

tion [13], which starts from the open-world assumption but allows for the closure of explicitly selected parts of the domain model.

We show how local closed-world reasoning, realised by either autoepistemic or circumscriptive DLs, improves the behaviour of matchmaking in our running example on an electronic marketplace for PC products.

5.1 Forms of Local Closure for Matchmaking

In the discussion of matching supply and demand in Example 1, we have seen a static attempt to realise closure based on explicit closure axioms which permanently affect the domain knowledge in an undesired way. When we speak of local closure in the following, however, we mean a way to tag parts of the domain model such that they are treated in the closed-world sense dynamically at reasoning time, without the need for adding any axioms to a knowledge base.

We identify some general forms of local closure which occur in the context of matchmaking, based on cases of matching supply and demand in our running example. In particular, we focus on cases in which the open-world semantics produces undesired positive matches that can be filtered out by a form of local closure, yielding a more intuitive matching behaviour.

Local Concept Closure If a concept C is locally closed, only such objects should occur in the extension of C for which there is evidence to be in there.

Closing Atomic Concepts
The simplest form of local concept closure occurs when an atomic concept is to be interpreted under closed-world semantics. In our running example this is desired in the case of matching demand D_1 with supply S_{A_2}, in order to filter out the no name graphics adapter for which the provider didn't state whether it supports the requested dual screen feature or not. Here we would like to interpret the atomic concept $\mathit{DualScreenGfxCard}$ in the closed-world sense, concluding that a graphics adapter does not support dual screen if this has not been explicitly stated. This local closure would then filter out S_{A_2} as a false positive match.

Closing Complex Concepts
Similarly, a concept that is identified by a complex concept expression might be the target for local closure. In our example, this is the case when matching demand D_1 with supply S_{A_3}, requiring graphics adapters to be explicitly stated to have both the requested dual screen feature and the provided TV output at the same time. Here the closure involves the complex concept $\exists\, \mathit{hasOutput}.\{TV\}$, which is to be interpreted under closed-world semantics to also filter out S_{A_3} as a false positive match, since there is no concrete graphics adapter which is known to have both the features.

Local Role Closure If a role r is locally closed, only such pairs of objects should occur in the extension of r for which there is evidence to be in there.

Closing Roles as a Whole

A role can be locally closed "as a whole", in which case it is entirely interpreted under closed-world semantics, no matter in which context it is used. In our example, this applies to the case of matching demand D_1 with supply S_{B_1}, where the graphics adapter used has to support both Linux and Windows. Since vendors state the support of graphics adapters in O_{ven} but there is no concrete graphics adapter known to be supported by both operating systems, the local closure of the entire role *supports* filters out the false positive match. Any graphics adapter for which there is no evidence to be supported by a particular operating system is by default concluded to not have this support.

Closing Roles Partially

Sometimes it is desired to close a role partially, affecting only those pairs of individuals that share a common domain or range concept. In the example, the role *supports* that connects operating systems with hardware components has a rather broad range. For some parts of its range, such as graphics adapters, closure can be desirable, as shown in the previous case, since vendors state the support of graphics adapters explicitly in the domain model. For other parts of the range, however, closure of the role should be avoided, such as for storage devices, which becomes apparent in the matching results for demand D_2. When matching this demand with the supply S_{B_2}, a closed-world interpretation of the part of the role *supports* that concerns graphics adapters correctly filters out the positive match because no dual screen graphics adapter is stated to have support from a Linux system. Also when matching D_2 with the supply S_{A_3}, this partial closure correctly keeps the positive match, since WindowsXP does support a dual screen graphics adapter. A closure of the role *supports* as a whole, however, would also eliminate the positive match with S_{A_3} because vendors do not state support for concrete storage devices, and thus, WindowsXP is not known to support RAID storage, as requested.

5.2 Matching with Local Closure by Epistemic Operators

Autoepistemic DLs introduce epistemic operators, in particular the **K**-operator, which can be used to refer to objects that are *known* to have certain properties. We show how the **K**-operator can be applied to modify resource descriptions for realising forms of local closure in the context of the supply and demand situation in our running example for matchmaking in an electronic marketplace.

Local Closure with Epistemic Operators

To achieve local closure with autoepistemic DLs, epistemic operators can directly be applied to the part of the domain model to be closed. Since for intersection matching we are concerned with concept satisfiability as a standard DL reasoning task, we show how the application of the **K**-operator affects the satisfiability of concepts.

Consider, for example, the two concepts *DualScreenGfxCard* and *RAIDStorage* taken from Example 1. In the knowledge base defined as $KB := O_{PC} \cup O_{ven}$, these two concepts are satisfiable. When imposing a local closure by applying the **K**-operator, the concept $\mathbf{K}\,DualScreenGfxCard$ is still satisfiable with respect to KB, while the concept $\mathbf{K}\,RAIDStorage$ is not. The reason is that in KB there

is an individual *known* to be a dual screen graphics adapter, while there is no individual *known* to be a RAID storage device, which reflects the desired closed-world view on this part of the domain model. To see this, recall from the definition of the formal semantics of autoepistemic DLs that epistemic concepts are interpreted as an intersection of concept extensions over all models of a knowledge base. The interpretation of the concept $\mathbf{K}\mathit{DualScreenGfxCard}$ intersects all extensions $\mathit{DualScreenGfxCard}^{\mathcal{I}}$ for models $\mathcal{I} \in \mathcal{M}(\mathit{KB})$, filtering out those individuals that are outside of the extension of $\mathit{DualScreenGfxCard}$ for some models. The only remaining individual is $\mathit{2ScreenCard}$, that is explicitly asserted to the concept $\mathit{DualScreenGfxCard}$, which makes $\mathbf{K}\mathit{DualScreenGfxCard}$ satisfiable. For the concept $\mathit{RAIDStorage}$, however, there is no explicitly asserted instance, and thus, no individual remains in the intersection of extensions $\mathit{RAIDStorage}^{\mathcal{I}}$ over the models $\mathcal{I} \in \mathcal{M}(\mathit{KB})$, which makes $\mathbf{K}\mathit{RAIDStorage}$ unsatisfiable.

As the **K**-operator can also be directly applied to roles, the local closure of roles can be realised due to a similar argumentation.

Autoepistemic Closure in the Example

Local closure by means of epistemic operators can be used to filter out undesired positive matches in our running example.

Concept Closure

Closing Atomic Concepts
The undesired positive match between D_1 and S_{A_2} can be eliminated using an epistemic operator to close the concept $\mathit{DualScreenGfxCard}$, as shown in the following modified demand.

$$\begin{aligned} D_1' \quad = \quad & \mathit{Computer} \sqcap \exists\, \mathit{hasGfx}.\mathbf{K}\mathit{DualScreenGfxCard} \\ & \sqcap \forall\, \mathit{hasComponent}.(\exists\, \mathit{supports}^{-}.\mathit{WindowsOS}) \end{aligned}$$

By asking for computers with graphics adapters *known* to be dual screen, D_1' filters out supply S_{A_2} that specifies $\mathit{NoNameGfxCard}$ to be the graphics adapter used. Indeed, for $\mathit{NoNameGfxCard}$ there is no evidence to support dual screen, i.e. $\mathit{KB} \not\models \mathit{DualScreenGfxCard}(\mathit{NoNameGfxCard})$, and thus, it cannot be in the extension of the concept $\mathbf{K}\mathit{DualScreenGfxCard}$. Since the role hasGfx is declared functional, there can also not be used another graphics adapter besides $\mathit{NoNameGfxCard}$ that would fulfil the demanded requirement. Therefore the conjunction $D_1' \sqcap S_{A_2}$ is unsatisfiable with respect to KB, and $\mathit{match}(\mathit{KB}, D_1', S_{A_2}) = \mathsf{false}$ as desired.

In this case, the closure can easily be done on the requester's side by augmenting a concept with an epistemic operator, while leaving the domain knowledge as well as the provider's resource descriptions and ontologies untouched.

Closing Complex Concepts
The modified demand D_1' is not sufficient to also filter out S_{A_3} as a positive match, which specifies the graphics adapter to be anyone with TV output. Namely, the concrete graphics adapter $\mathit{2ScreenCard}$ can well be used in a configuration captured by both D_1' and S_{A_3}, since it is known to support dual screen and there is also the possibility that it has TV output. To make sure that all

the intended graphics adapters have both features, the supply S_{A_3} has to be also modified, as follows.

$$\begin{aligned} S'_{A_3} \quad = \quad & \mathit{DesktopPC} \sqcap \exists\, \mathit{hasGfx}\,.\mathbf{K}(\exists\, \mathit{hasOutput}\,.\{\mathit{TV}\}) \\ & \sqcap \forall\, \mathit{runsOS}\,.\mathit{WindowsOS} \end{aligned}$$

In this modified supply, the complex concept $\exists\, \mathit{hasOutput}\,.\{\mathit{TV}\}$ is closed by using an epistemic operator. It asks for desktop PCs with graphics adapters that are *known* to have a TV output. Together with the modified demand D'_1, it can be used to filter out the third offer from provider A as a positive match. There is no graphics adapter that is both *known* to support dual screen and *known* to have TV output. Thus, the conjunction $D'_1 \sqcap S'_{A_3}$ is unsatisfiable, and $\mathit{match}(\mathcal{KB}, D'_1, S'_{A_3}) = \mathsf{false}$ as desired.

The closure of a complex concept can immediately be achieved by using epistemic operators, since an epistemic operator can be directly applied to it. The mutual closure on both the requester's and the provider's side in this case, however, shifts some of the control over closure from the requester to the provider, since the provider's resource description has to be modified.

If the requester is very sceptical about supplies to support dual screen graphics adapters, then they could want to close the complex concept expressed through $\exists\, \mathit{hasGfx}\,.\mathit{DualScreenCard}$, ensuring their intended configurations to be *known* to have dual screen graphics adapters. Using epistemic operators, this would yield the following modified demand.

$$\begin{aligned} D_1^* \quad = \quad & \mathit{Computer} \sqcap \mathbf{K}(\exists\, \mathit{hasGfx}\,.\mathit{DualScreenGfxCard}) \\ & \sqcap \forall\, \mathit{hasComponent}\,.(\exists\, \mathit{supports}^-\,.\mathit{WindowsOS}) \end{aligned}$$

One could think that the demand D_1^* would properly rule out those supplies that do not explicitly state that they offer dual screen graphics adapters, including the non-modified supply S_{A_3}. However, realised with epistemic operators, this closure is too strict because there are simply no known individuals in the knowledge base for which this property could actually hold. Together with S_{A_3}, the modified demand D_1^* would rule out any of the supplies, i.e. $\mathit{match}(\mathcal{KB}, D_1^*, S) = \mathsf{false}$ for all of the supplies from the example. In this particular scenario, the concept *Computer* is modelled such that it does not have explicit instances assigned. On the other hand, epistemic operators rule out those individuals that are not known to the knowledge base. By asking for computers that are *known* to have certain properties using epistemic operators, there is no individual left in the intersection of extensions over models $\mathcal{I} \in \mathcal{M}(\mathcal{KB})$. Thus, a drawback of epistemic operators in the context of matchmaking is that they cannot be used for the closure of concepts for which there are no explicitly asserted instances present in the ABox.

Role Closure

Closing Roles as a Whole

The undesired positive match between D_1 and S_{B_1} can be eliminated by further modifying the demand D'_1, closing the role *supports*. The following resulting demand D''_1 shows how this is done using epistemic operators.

$$\begin{aligned} D''_1 \quad = \quad & \mathit{Computer} \sqcap \exists\, \mathit{hasGfx}\,.\mathbf{K}\,\mathit{DualScreenGfxCard} \\ & \sqcap \forall\, \mathit{hasComponent}\,.(\exists\, \mathbf{K}\mathit{supports}^-\,.\mathit{WindowsOS}) \end{aligned}$$

The demand D_1'' asks for computers with graphics adapters *known* to be dual screen all of whose components are *known* to be supported by a Windows operating system. Moreover, also the supply S_{B_1} needs to be adjusted in the same style, to require components to be *known* to have Linux support, which results in the following modified supply.

$$S'_{B_1} = \mathit{Laptop} \sqcap \forall\, \mathit{hasComponent}\,.(\exists\, \mathbf{K}\mathit{supports}^{-}\,.\mathit{LinuxOS})$$

Now, the supply S'_{B_1} is filtered out when matched against D_1'', since among the *known* dual screen graphics adapters there is none that is *known* to be supported by Windows and Linux at the same time. Therefore, there is no individual left that could fill the *hasGfx* role such that the conjunction $D_1'' \sqcap S'_{B_1}$ would be satisfied, and hence, $\mathit{match}(\mathcal{KB}, D_1'', S'_{B_1}) = \mathsf{false}$.

Again, the closure here affects both the demand and supply descriptions but leaves domain ontologies as they are. Closing a role "as a whole" by means of epistemic operators means to apply an epistemic operator to any occurrence of the role in a resource description.

Closing Roles Partially

In the case of matching demand D_2 against the available supplies, a partial closure of the role *supports* is desired, as for graphics adapters support is stated explicitly, whereas for storage devices it is not. By means of epistemic operators, such a partial closure is achieved through the following modified demand.

$$\begin{aligned} D'_2 \;=\; & \mathit{DesktopPC} \sqcap \exists\, \mathit{hasStorage}\,.\mathit{RAIDStorage} \\ & \sqcap \exists\, \mathit{runsOS}\,.(\exists\, \mathbf{K}\mathit{supports}\,.\mathit{DualScreenGfxCard} \sqcap \exists\, \mathit{supports}\,.\mathit{RAIDStorage}) \end{aligned}$$

An epistemic operator has only been applied to the restriction of the role *supports* that concerns dual screen graphics adapters and not to the one that concerns RAID storage. Thus, demand D'_2 asks for RAID-enabled desktop PCs that run an operating system which (potentially) supports RAID storage and which is *known* to support dual screen graphics. By this closure, the undesired positive match with supply S_{B_2} can be eliminated, in which workstations with single output graphics adapters are offered that run a Linux operating system. Indeed, there is no Linux operating system in the domain model that is known to support a dual screen graphics adapter, as requested, and therefore the conjunction $D'_2 \sqcap S_{B_2}$ is unsatisfiable.

If we applied an epistemic operator also to the part of the role restriction that concerns RAID storage, closing the role *supports* as a whole, we would run into a similar problem as with the epistemic closure of a concept for which there are no explicit instances present. In fact, the domain model does not speak of concrete storage devices in form of explicit instances, and asking for desktop PCs with operating systems that are both *known* to support dual screen graphics and *known* to support RAID storage would yield undesired negative matches, as in the former case. In particular, the positive match with supply S_{A_3} would be ruled out because *WindowsXP* is not *known* to support RAID storage nor any other form of storage. Thus, closing the role *supports* only partially as within D'_2 keeps the supply S_{A_3} as a positive match, i.e. $\mathit{match}(\mathcal{KB}, D_2, S'_{A_3}) = \mathsf{true}$, but eliminates the supply S_{B_2}, i.e. $\mathit{match}(\mathcal{KB}, D'_2, S_{B_2}) = \mathsf{false}$, as desired.

5.3 Matching with Local Closure by Circumscription

Circumscriptive DLs introduce the concept of minimisation of predicates, identified in circumscription patterns, which can be used to reduce a predicate's extension to those elements for which there is evidence to be contained in the extension. We show how the application of circumscription patterns to knowledge bases can be used for realising forms of local closure in the context of the supply and demand situation in our running example for matchmaking in an electronic marketplace.

Local Closure with Circumscription

To achieve local closure with circumscriptive DLs, the parts of the domain model to be closed can be identified for minimisation in a circumscription pattern. Again, in the light of intersection matching, we are concerned with concept satisfiability as a standard DL reasoning task and show how the minimisation of predicates affects the satisfiability of concepts.

Consider again the two concepts *DualScreenGfxCard* and *RAIDStorage* taken from Example 1, which are satisfiable with respect to the knowledge base *KB*. When imposing a local closure by applying the circumscription pattern $\mathsf{CP} = (M = \{\mathit{DualScreenGfxCard}, \mathit{RAIDStorage}\}, F = \emptyset)$ to *KB*, *DualScreenGfxCard* is still satisfiable with respect to $\mathrm{circ}_{\mathsf{CP}}(\mathit{KB})$, while *RAIDStorage* is not. The reason is that in *KB* there is evidence for an individual, namely *2ScreenCard*, to be a dual screen graphics adapter, while there is no evidence for the existence of any RAID storage device. To see this, recall from the definition of the formal semantics of circumscriptive DLs that the extensions of minimised predicates are reduced by eliminating models that are "smaller" in the extension of minimised predicates than others, leaving only those with the smallest extension possible. For the concept *DualScreenGfxCard*, the smallest possible extension must at least contain the individual *2ScreenCard*, and thus, it is satisfied in the remaining models. For the concept *RAIDStorage*, however, the smallest possible extension is the empty set because there is no evidence for the existence of an instance. Any model $\mathcal{I} \in \mathcal{M}(\mathit{KB})$ with $\#(\mathit{RAIDStorage}^{\mathcal{I}}) > 0$ is eliminated by some model $\mathcal{J} \in \mathcal{M}(\mathit{KB})$ with $\mathit{RAIDStorage}^{\mathcal{J}} = \emptyset$ due to the condition $\mathcal{J} <_{\mathsf{CP}} \mathcal{I}$, and no model remains in which *RAIDStorage* is satisfied.

As circumscription patterns can identify both concepts and roles for minimisation, the local closure of roles can be realised in a similar way.

Circumscriptive Closure in the Example

Local closure by means of circumscription can be used to filter out undesired positive matches in our running example.

Concept Closure

Closing Atomic Concepts

When circumscription is used as a technique for realising local closure, the false positive match between D_1 and S_{A_2} can be eliminated by applying a circumscription pattern to the knowledge base $\mathit{KB} := O_{PC} \cup O_{ven} \cup O_A$ with respect to

which matching is evaluated. The following circumscription pattern closes the concept *DualScreenGfxCard* by minimisation.

$$\mathsf{CP}_1 = (M = \{\mathit{DualScreenGfxCard}\}, F = \emptyset)$$

By minimising the concept *DualScreenGfxCard*, only those graphics adapters support dual screen for which there is evidence in *KB* that they do. For the no name graphics adapter introduced by provider A there is no evidence of dual screen support, i.e. $\mathit{KB} \not\models \mathit{DualScreenGfxCard}(\mathit{NoNameGfxCard})$, and therefore it is taken out of the extension of *DualScreenGfxCard* during minimisation. As a result, the conjunction $D_1 \sqcap S_{A_2}$ is unsatisfiable with respect to the circumscribed knowledge base $\mathrm{circ}_{\mathsf{CP}_1}(\mathit{KB})$, and thus, the supply S_{A_2} no longer yields a positive match, i.e. $\mathit{match}(\mathit{KB}, D_1, S_{A_2}) = \mathsf{false}$.

Also here, the local closure can be initiated on the requester side, since all the predicates that occur in the circumscription pattern CP_1 are known to the requester at the time of formulating the demand description.

Closing Complex Concepts

As we have seen before, the single closure of the concept *DualScreenGfxCard* on the requester's side is not sufficient to also rule out supply S_{A_3} as a positive match for D_1. To achieve this, the property of having TV output for graphics adapters expressed through the complex concept $\exists\, \mathit{hasOutput}.\{\mathit{TV}\}$, as advertised on the provider's side, also needs to be closed. A complex concept can be minimised with circumscription by setting it equivalent to a newly introduced atomic concept, which is then minimised. The following circumscription pattern minimises the two concepts *DualScreenGfxCard* and $\exists\, \mathit{hasOutput}.\{\mathit{TV}\}$ in parallel, while a new concept name A is introduced together with the axiom $\alpha_A = A \equiv \exists\, \mathit{hasOutput}.\{\mathit{TV}\}$.

$$\mathsf{CP}_2 = (M = \{\mathit{DualScreenGfxCard}, A\}, F = \emptyset)$$

In addition to graphics adapters with dual screen support, this pattern also minimises all objects that have a TV output, leaving only those for which there is evidence to have one. As there is no graphics adapter in *KB* which necessarily has TV output and which also necessarily supports dual screen, there is no individual left that meets the requirements of both the supply and demand. As a result, the conjunction $D_1 \sqcap S_{A_3}$ is unsatisfiable with respect to the circumscribed knowledge base $\mathrm{circ}_{\mathsf{CP}_2}(\mathit{KB} \cup \alpha_A)$, and thus, the supply S_{A_3} is filtered out, i.e. $\mathit{match}(\mathit{KB}, D_1, S_{A_3}) = \mathsf{false}$ as desired.

Here, the closure cannot easily be initiated on the requester's side because the use of the complex concept occurs in the provider's description of the supply.

In the case where a requester is very sceptical about supplies to support dual screen graphics adapters, circumscription can be used to realise the closure of the complex concept $\exists\, \mathit{hasGfx}.\mathit{DualScreenCard}$, by minimising those things which have dual screen graphics adapters. In opposite to the use of epistemic operators, circumscription does not strongly rely on the presence of explicitly asserted instances in an ABox of a knowledge base, and thus, it can cope with the lack of known computers in *KB*. As before, the complex concept is given a name through the axiom $\alpha_A = A \equiv \exists\, \mathit{hasGfx}.\mathit{DualScreenCard}$ and the newly introduced concept name A is minimised. To deal with the lack of explicit

instances of the concept *Computer* and its subconcepts, we use the technique of fixing predicates during minimisation. Namely, we fix the concept in the resource description of the supply S_{A_3}, giving it a name through the axiom $\alpha_B = B \equiv \mathit{DesktopPC} \sqcap \exists\, \mathit{hasGfx}.(\exists\, \mathit{hasOutput}.\{\mathit{TV}\}) \sqcap \forall\, \mathit{runsOS}.\{\mathit{WindowsXP}\}$. Overall the following circumscription pattern minimises A and fixes B.

$$\mathsf{CP}_3 = (M = \{A\}, F = \{B\})$$

By minimising all things that have a dual screen graphics adapter, instances of the concept specified in supply S_{A_3} are excluded from having this property, since they are not explicitly required to. Hence, the conjunction $D_1 \sqcap S_{A_3}$ is unsatisfiable with respect to $\mathrm{circ}_{\mathsf{CP}_3}(\mathcal{KB} \cup \alpha_A \cup \alpha_B)$, and the supply S_{A_3} is filtered out as desired. If we would not fix the concept B then any other supply, in particular S_{A_1}, would be filtered out as well, which is not desired. In case B were a varying concept when D_1 is matched with S_{A_1}, all possible instances of B would be taken out of B's extension because B implies the property of having a dual screen graphics adapter, captured by the concept A, which is minimised. Since there is no evidence for the existence of any instance in A, B would be empty as well, and thus unsatisfiable.

Role Closure

Closing Roles as a Whole
Using circumscription, roles can be closed as a whole by minimising them as predicates. The role *supports*, used in the supply S_{B_1} to denote components that are supported by Linux, can be closed by means of the following circumscription pattern in order to rule out S_{B_1} as a positive match for D_1.

$$\mathsf{CP}_4 = (M = \{\mathit{supports}\}, F = \emptyset)$$

By minimisation of the role *supports*, which connects operating systems and hardware components, we achieve that only those components are supported by operating systems for which their support has been explicitly stated in the domain knowledge, e.g. in O_{ven}. Since there is no graphics adapter that is stated to be both supported by Windows and by Linux, the conjunction $D_1 \sqcap S_{B_1}$ is unsatisfiable with respect to the circumscribed knowledge base $\mathrm{circ}_{\mathsf{CP}_4}(\mathcal{KB})$.

Closing Roles Partially
Using circumscription, roles cannot directly be closed partially because roles cannot be constructed in the same way as concepts can. In a circumscription pattern only atomic predicates are allowed to occur. In the case of concepts, complex concepts can be minimised by introducing new atomic concepts and defining them to be equivalent to the complex concept to be closed by means of additional axioms. This is not possible for a role, and thus, roles can only be closed as a whole within the framework of circumscriptive DLs. In this sense, circumscriptive closure can only be applied in the granularity of the domain model, and a partial closure of a role requires a more fine-grained modelling of the domain by introducing subroles, which could then be minimised.

5.4 Discussion

In summary, we have investigated two different nonmonotonic extensions to the DL formalism that can be used to realise local closed-world reasoning, namely autoepistemic DLs and circumscriptive DLs. Both approaches allow to eliminate models of a knowledge base in which extensions of predicates to be closed encompass (pairs of) individuals for which there is no evidence to be in the extension. In the context of intersection matching, this technique can be used to achieve that a conjunction $R_1 \sqcap R_2$ of concepts for resources is unsatisfiable by default, eliminating undesired positive matches.

As a result of this investigation, we observe that the two nonmonotonic formalisms have complementary properties with regard to the different cases of local closure we have identified in matching. Namely, in situations in which a domain model lacks explicit instances of a predicate to be closed, a closure by epistemic operators leads to unsatisfiability of the predicate even if some objects are existentially inferred to be in the predicate's extension. The semantics of autoepistemic DLs is defined such that only known individuals remain in the intersection of predicate extensions over the models of a knowledge base. Circumscription, on the other hand, allows to keep existentially inferred objects in a predicate's extension by fixing predicates. Therefore, circumscriptive DLs provide more flexible ways for imposing local closure in such situations.

Moreover, if a predicate is to be closed only partially or in a certain context, epistemic operators can be applied to some occurrences of the predicate, while other occurrences are left as they are. Since epistemic operators directly apply to concepts and roles within arbitrarily complex expressions, autoepistemic DLs allow for a fine-grained selection of the parts of the domain model to be locally closed. In circumscription patterns, on the other hand, predicates can only be minimised as a whole, and once a predicate is selected for local closure the open-world semantics does not apply anymore for this part of the domain model, no matter in which context it is used. In this sense, autoepistemic DLs provide more flexible ways for imposing local closure in terms of a more fine-grained access to the parts of the domain model.

As a result from this discussion, the classification of various cases of closure together with the identification of the specifics of the two investigated approaches to local closed-world reasoning can be seen as a first step towards developing methodological guidelines for describing resources in a locally closed domain model. It remains future research to devise a complete methodology on when to apply the **K**-operator and how to formulate appropriate circumscription patterns such that the details of the logics are hidden from the modeller.

Abstracting from the specific setting of matchmaking, we also note that the state of the art with respect to the two formalisms is also quite different. For autoepistemic DLs, there exists a reasonable body of work including further extensions and reasonable algorithms, as reported e.g. in [23]. It is to be noted, however, that efficient implementations are to date still missing. For circumscriptive DLs, only basic results have been achieved so far. Furthermore, only naive algorithms exist, and the development of reasonably efficient algorithms and implementations has not even been started yet.

Either of the approaches can be used for extending any given DL, in principle, i.e. in terms of the semantics of the extensions. This is the reason why, in this paper, we have been very liberal with the use of DL constructors in our

examples. However, decidability and computational complexity properties are directly dependent on the DL which is extended.

In terms of decidability, which was a major design criteria for OWL-DL, both approaches have been researched to a considerable extent. For autoepistemic DLs, the **K**-operator can be used for concept and for role closure, with some restrictions, while retaining decidability [11, 28]. For circumscriptive DLs, concept closure retains decidability while role closure, i.e. minimising roles, in general leads to undecidability [7], which is undesirable as role closure is important for matchmaking, as we have observed in our investigations.

In terms of worst-case computational complexities, local closed world extensions generally make matters worse. For circumscriptive DLs results have been reported in [7]; a typical result would be that concept satisfiability is NExp^{NP}-complete for circumscriptive $\mathcal{ALCIO}$. For autoepistemic DLs, many complexity results are reported in [28, 11, 23]. To be noted is that certain interesting fragments remain of reasonable complexity, e.g. when epistemic operators are not allowed to occur in the knowledge base, as in our examples.

6 Related Work

In this paper, we have investigated the use of nonmonotonic extensions to description logics, that facilitate forms of local closed-world reasoning, for the matchmaking of semantically annotated resources. While most essential references have been provided in the text, we here summarise related work on both nonmonotonic extensions to description logics and description logic based matchmaking.

Closed-World Reasoning in Description Logics

In essence, there exist two different approaches to adding closed-world reasoning aspects to description logics: hybrid languages and extensions of description logics.

Hybrid approaches are based on the interplay between two knowledge representation paradigms, the one of which uses open-world reasoning, while the other one uses closed-world reasoning. The single most prominent hybrid approach to date is reported in [12], which interfaces answer set programming and description logics in an intricate way. The resulting paradigm, however, is of an entirely different flavour, and not as closely tied to OWL as the paradigms we have studied in this paper. Its usability for ontology-based applications in general and for matchmaking in particular remains largely to be investigated.

Concerning extensions of description logics, the currently most developed approach are autoepistemic description logics. While in this paper we have treated the simple case with the **K**-operator not occurring within the knowledge base, as introduced in [10], more sophisticated forms of autoepistemic DLs are reported in [11, 28], including an additional epistemic operator **A** related to negation -as-failure, which allows for default rules and integrity constraints. The latter have been applied in a Semantic Web context in [16]. The most recent advances on combining autoepistemic DLs with logic programming rules are reported in [23]. We have also covered circumscriptive description logics as another recent approach reported in [7]. On the historic side, work on incorporating default

rules into description logics reported in [1] shall also be mentioned. It remains to be analysed how such explicit default rules can be utilised for modelling resource semantics to feature local closed-world reasoning in the matchmaking context.

Description Logic Based Matchmaking

In the context of the description logic framework, the problem of matchmaking for the purpose of resource retrieval was first investigated in [14, 30] and [29], where resources represented as DL concepts have been matched using DL inferencing tasks. Following this line, in [25] and [20] various DL inferences were assigned different degrees of match in the context of service discovery in the Semantic Web.

A similar classification of different matches has been proposed in [24], where the notion of "partial match" was proposed together with a strategy for ranking non-intersecting resource descriptions in an approximate way. Further work on approximate matching techniques was carried out in [4] with an operator for concept difference, and in [8] based on a structural matching algorithm with penalty functions. As the most elaborate work in this line of research, the approach reported in [9] makes use of the non-standard DL inferences of concept contraction and abduction, approximating the notions of intersection and subsumption to provide fine-grained rankings for partial and potential matches.

In [17], the standard DL inferences for intersection matching and subsumption matching were investigated with respect to the variance imposed by incomplete descriptions due to the open-world assumption, and some problems with classical inferences were identified. As a follow-up, the use of local closed-world reasoning based on autoepistemic DLs was proposed in [18] to cope with these problems, which laid the basis for our current work, while in [15] patterns for modelling semantic descriptions by means of classical DL were recommended.

Furthermore, in the context of Semantic Web Services there are works such as [6, 2, 19] that tailor the use of DLs for semantic descriptions of services with special constructs to capture service-specific notions like input/output parameters or actions.

7 Summary and Outlook

We have presented an approach to matchmaking of semantically annotated resources based on description logics by means of an elaborate example on electronic market places taken from the domain of eCommerce. We have identified problematic cases due to the open-world assumption in which classical DL inferencing yields an unintuitive matching behaviour, and where forms of local closure are desired. Hence, we have analysed the state of the art in local closed-world reasoning in description logics with respect to its applicability to matchmaking. Summarising, we can say that both autoepistemic and circumscriptive DLs allow to solve some of the problems which occur in matchmaking due to the open-world assumption of OWL, however, they differ in expressive features for modelling resources. Namely, autoepistemic DLs allow for a fine-grained closure of parts of a domain model, whereas circumscriptive DLs can handle unknown objects not known to any involved ontology.

As future research, we see the need for developing a methodology which can help the practitioner in using local closed-world modelling for practical matchmaking. This is particularly important for making the formulation of resource descriptions accessible to non-specialists by introducing an additional layer of abstraction. Here, the identification of various cases of closure and the specifics of the two presented nonmonotonic approaches with regard to unknown objects could be used as a starting point.

Moreover, OWL corresponds to a quite expressive DL and the nonmonotonic features for the autoepistemic and circumscriptive case have only been spelled out for restricted variants. Corresponding investigations concerning decidability and complexity issues remain to be done, in particular with respect to reasonable fragments of autoepistemic or circumscriptive OWL-DL, as this will provide guidance for achieving scalable system behaviour.

Furthermore, we have seen that the algorithmisation of local closed-world reasoning for matchmaking with description logics needs to be advanced in order to allow for implementation and feasibility testing on larger examples. While for the autoepistemic case there exist reasoning algorithms on paper, for the circumscriptive case no practical algorithm has been proposed so far. For both approaches it would be desirable to see them integrated into optimised state-of-the-art DL reasoning tools, once the remaining fundamental issues have been addressed and resolved.

References

[1] F. Baader and B. Hollunder. Embedding Defaults into Terminological Knowledge Representation Systems. *J. Automated Reasoning*, 14:149–180, 1995.

[2] F. Baader, C. Lutz, M. Milicic, U. Sattler, and F. Wolter. A Description Logic Based Approach to Reasoning about Web Services. In *Proceedings of the WWW 2005 Workshop on Web Service Semantics (WSS2005)*, Chiba City, Japan, 2005.

[3] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[4] B. Benatallah, M.-S. Hacid, C. Rey, and F. Toumani. Request Rewriting-Based Web Service Discovery. In *Proceedings of the 2nd International Semantic Web Conference (ISWC)*, pages 242–257, 2003.

[5] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[6] P. Bonatti. Towards Service Description Logics. In *JELIA '02: Proceedings of the European Conference on Logics in Artificial Intelligence*, pages 74–85, London, UK, 2002. Springer-Verlag.

[7] P. Bonatti, C. Lutz, and F. Wolter. Expressive Non-Monotonic Description Logics Based on Circumscription. In Patrick Doherty, John Mylopoulos, and Christopher Welty, editors, *Proceedings of the Tenth Inter-*

national Conference on Principles of Knowledge Representation and Reasoning (KR'06), pages 400–410. AAAI Press, 2006.

[8] A. Calì, D. Calvanese, S. Colucci, T. Di Noia, and F.M. Donini. A Description Logic based Approach for Matching User Profiles. In *Proc. of the 17th Intl. Workshop on DescriptionLogics (DL'04)*, volume 104. CEUR Workshop Proceedings, 2004.

[9] S. Colucci, T. Di Noia, E. Di Sciascio, M. Mongiello, and F. M. Donini. Concept Abduction and Contraction for Semantic-Based Discovery of Matches and Negotiation Spaces in an E-Marketplace. In *ICEC '04: Proceedings of the 6th international conference on Electronic commerce*, pages 41–50, New York, NY, USA, 2004. ACM Press.

[10] F. M. Donini, M. Lenzerini, D. Nardi, W. Nutt, and A. Schaerf. An Epistemic Operator for Description Logics. *Artificial Intelligence*, 100(1-2):225–274, 1998.

[11] F.M. Donini, D. Nardi, and R. Rosati. Description Logics of Minimal Knowledge and Negation as Failure. *ACM Transactions on Comput. Logic*, 3(2):177–225, 2002.

[12] T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. Effective Integration of Declarative Rules with External Evaluations for Semantic-Web Reasoning. In York Sure and John Domingue, editors, *The Semantic Web: Research and Applications, 3rd European Semantic Web Conference, ESWC 2006, Budva, Montenegro, June 11-14, 2006, Proceedings*, volume 4011 of *Lecture Notes in Computer Science*, pages 273–287. Springer, 2006.

[13] O. Etzioni, K. Golden, and D. Weld. Tractable Closed World Reasoning with Updates. In *Proceedings of the 4th International Conference on Knowledege Representation and Reasoning (KR1994)*, pages 178–189. Morgan Kaufmann, 1994.

[14] J. Gonzlez-Castillo, D. Trastour, and C. Bartolini. Description Logics for Matchmaking of Services. In *Proc. of the KI-2001 Workshop on Appl. of DL*, 2001.

[15] S. Grimm. Intersection-Based Matchmaking for Semantic Web Service Discovery. In *Proceedings of the 2nd International Conference on Internet and Web Applications and Services (ICIW'07)*. IEEE Computer Society, May 2007.

[16] S. Grimm and B. Motik. Closed World Reasoning in the Semantic Web through Epistemic Operators. In Bernardo Cuenca Grau, Ian Horrocks, Bijan Parsia, and Peter Patel-Schneider, editors, *OWL: Experiences and Directions*, Galway, Ireland, Nov 2005.

[17] S. Grimm, B. Motik, and C. Preist. Variance in e-Business Service Discovery. In *Proceedings of the 1st Int. Workshop SWS'2004 at ISWC 2004*, November 2004.

[18] S. Grimm, B. Motik, and C. Preist. Matching Semantic Service Descriptions with Local Closed-World Reasoning. In York Sure and John Domingue, editors, *The Semantic Web: Research and Applications, 3rd European Semantic Web Conference, ESWC 2006, Budva, Montenegro, June 11-14, 2006, Proceedings*, volume 4011 of *Lecture Notes in Computer Science*, pages 575–589. Springer, 2006.

[19] D. Hull, E. Zolin, A. Bovykin, I. Horrocks, U. Sattler, and R. Stevens. Deciding Semantic Matching of Stateless Services. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'06), Boston, Massachusetts, USA*, pages 1319–1324, 2006.

[20] L. Lei and I. Horrocks. A Software Framework for Matchmaking Based on Semantic Web Technology. *International Journal of Electronic Commerce (IJEC)*, 8(4):39–60, 2004.

[21] J. McCarthy. Circumscription – A Form of Non-Monotonic Reasoning. *Artificial Intelligence*, 13(1–2):27–39, 1980.

[22] R. Moore. Semantical Considerations on Nonmonotonic Logic. *Artificial Intelligence*, 25(1), 1985.

[23] B. Motik and R. Rosati. A Faithful Integration of Description Logics with Logic Programming. In M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI-07, Hyderabad, India, January 2007*, pages 477–482. Morgan Kaufmann, 2007.

[24] T. Di Noia, E. Di Sciascio, F.M. Donini, and M. Mogiello. A System for Principled Matchmaking in an Electronic Marketplace. *International Journal of Electronic Commerce (IJEC)*, 8(4):9–37, 2004.

[25] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic Matching of Web Service Capabilities. In *Proceedings of the 1st International Semantic Web Conference (ISWC)*, pages 333–347, 2002.

[26] P. F. Patel-Schneider, P. Hayes, I. Horrocks, and F. van Harmelen. OWL Web Ontology Language; Semantics and Abstract Syntax, W3C Candidate Recommendation. http://www.w3.org/TR/owl-semantics/, November 2002.

[27] C. Preist, J. Esplugas-Cuadrado, S. Battle, S. Grimm, and S. Williams. Automated B2B Integration of a Logistics Supply Chain Using Semantic Web Services Technology. In *Proc. of the 4th Int. Semantic Web Conference (ISWC)*, 2005.

[28] R. Rosati. Autoepistemic Description Logics. *AI Communications, IOS Press*, 11(3–4):219–221, 1998.

[29] E. Di Sciascio, F. M. Donini, M. Mongiello, and G. Piscitelli. A Knowledge Based System for Person-to-Person E-Commerce. In *Proc. of the KI-2001 Workshop on Applications of Description Logics*, 2001.

[30] D. Trastour, C. Bartolini, and C. Preist. Semantic Web Support for the Business-to-Business E-Commerce Lifecycle. In *Proceedings of the Eleventh International Conference on World Wide Web*, pages 89–98, 2002.