Cost Efficient, Adaptive Reasoning Strategies for Pervasive Service Discovery

Luke A Steller, Shonali Krishnaswamy and Mohamed M. Gaber
Monash University,
900 Dandenong Road,
Caulfield East, VIC, Australia.
+61 3990 32000
{Luke.Steller, Shonali.Krishnaswamy, Mohamed.Gaber}@infotech.monash.edu.au

ABSTRACT
With the emergence of high-end smart phones / PDAs there is an emerging opportunity to enrich mobile / pervasive services with semantic reasoning. This paper presents novel strategies for optimising semantic reasoning for realising semantic applications and services on mobile devices. Our mTableaux algorithm optimises the reasoning process to facilitate service selection. Since even optimised reasoning may be too resource intensive to complete, depending on ontology size and resource availability, we also outline our adaptive reasoning strategy which reduces result accuracy when resources become low. We also evaluate the impact of our strategies on performance and accuracy and show that mTableaux significantly improves performance.

Categories and Subject Descriptors
I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence – *Intelligent agents.*

General Terms
Performance, Design, Experimentation.

Keywords
Optimised Mobile Reasoning, Approximate Reasoning, Pervasive Semantic Discovery.

1. INTRODUCTION
The rapid expansion in uptake of mobile devices and in their processing capabilities provides many new opportunities for mobile users to access services and information in mobile environments. However, in order to realise these opportunities several new challenges must be addressed. Mobile environments are extremely dynamic, meaning that often devices have not met the other devices in their area before. As a result, uniform software abstraction and loose coupling is paramount in order to achieve interoperability between the requester and providers by adopting a Mobile Service Oriented Architectures (SOA). Mobile SOA classes range from service centric approaches in which a high-end broker node provides service matching capabilities requiring managed infrastructure provision, to pure peer-to-peer (P2P) approaches in which all nodes are equal and dynamically form an ad-hoc network of requester and provider nodes on the fly [1]. However, reliance on a centralised broker node means that when the broker becomes unreachable, the whole environment is broken. Since mobile nodes are less reliable than fixed nodes, due to constant disconnection and mobility, requiring server centric approaches is often not viable. Conversely the emergence of semantic web languages, which provide a powerful means to describe services by meaning rather than syntactic equivalence, require resource intensive reasoning.

Advances in device capability mean that mobile devices can act not only as service consumers but also as service providers or will do so in the near future [2, 3] and can be migrated to another device if a mobile node moves out of range and becomes unavailable [4]. As such, we advocate the need to support a decentralised approach to service selection, such that reasoning is conducted on the mobile device itself. We identify three decentralised environment examples. Figure 1 illustrates a service based, centralised approach, while figure 2 illustrates each example on-device, decentralised approach.

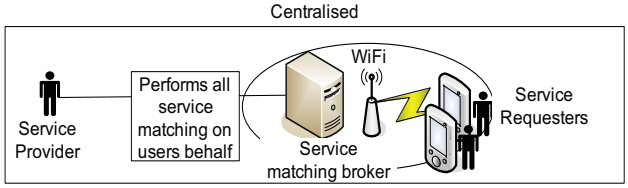

Figure 1. Server-based approach to service selection.

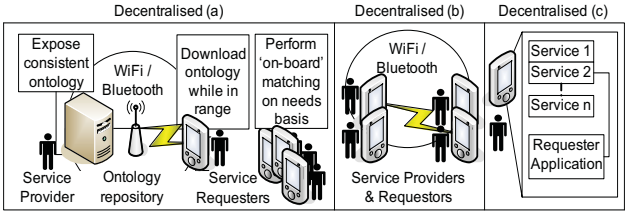

Figure 2. On-device approaches to service selection.

A centralised approach often comes are a significant financial cost to the user and creates a potential performance bottleneck and single point of failure. Alternatively, in figure 2(a) the services

ICPS'09, July 13–17, 2009, London, United Kingdom.

themselves are provided by a fixed node however centralised brokering/matching is not provided. Instead the user's device downloads ontologies and performs the reasoning on-demand at the user's request. This avoids the financial costs in provision of brokering and the drawbacks of a centralised approach. For example, a user's phone downloads ontologies in a foreign city centre as she walks past advertisement points.

Figure 2(b) is a pure mobile ad-hoc network. For instance, students sharing data on a field trip [5], emergency situations, traffic information sharing, etc. In figure 2(c), the services reside on the user's own device, for instance, Google[1] and Yahoo[2] already offer many mobile applications such as blogging, news, finance, sports, etc, which may be installed or removed depending on the user's needs and the Apple iPhone[3] advertises itself as having "35,000 apps. And counting.".

All of these three on-device, decentralised configurations (in figure 2) alleviate the need for high level of infrastructure provision, the risk of no service when out of network range, and reduce consumption of precious device battery power, involved in constant network usage. In addition, it is easy to deploy and extremely scalable because the number of devices has no impact on performance and making it very suitable to some environments.

As such, in this paper, we address the key issue of providing highly optimised semantic reasoning. As a consequence, our reasoner will also function on a mobile device, to provide on-device reasoning. Tableaux algorithm is well known and used by reasoners such as Pellet, RacerPro and FaCT++. We aim to optimise Tableaux in order to enable these reasoners to function in a computationally cost-efficient manner on a mobile device. To this end, we present our mTableaux algorithm, which implements strategies to optimise description logic (DL) reasoning tasks so that relatively large reasoning tasks of several hundred individuals and classes may function on small devices. However, these optimisations may not be sufficient to effectively compare a request against many potential services on a mobile device. For instance a reasoning task may be too large to complete entirely, with the resources available. Therefore, we also provide an adaptive reasoning which matches the most important request conditions (to the user) first to make the best use of processing time available and support partial matching. We present our approach, a prototype and experimental evaluations which demonstrate the feasibility of the semantic service discovery to operate on a mobile device.

This paper takes an important step forward in developing scalable semantic reasoning techniques which are useful for both mobile / pervasive and standard service selection algorithms. The remainder of the paper is structured as follows. In section 2 we describe related work. In section 3 we present our discovery architecture, discussion about semantic reasoning and overview of our strategies. In section 4 we describe our strategies to optimise the Tableaux algorithm. Since further performance gains may be needed for mobile reasoning on large ontologies or requests, we also provide in section 5, strategies which balance efficiency with accuracy. In section 6 we discuss our implementation and provide evaluations of our work. Finally in section 7 we conclude the paper.

[1] www.google.com/mobile

[2] www.yahoo.com/mobile

[3] www.apple.com/iphone

2. RELATED WORK

While current service discovery architectures such as Jini [6], UPnP do not make use of semantic languages, there is a growing emergence of DAML-S/OWL-S semantic matchmakers such as CMU Matchmaker [7] which requires a centralised high-end node to perform reasoning using Racer. DIANE [8] is designed for ad-hoc service discovery and defines its own semantic language. Architectures such as Gaia [9] provide semantically driven context middleware utilising FaCT++, and Josef [10] establishes a virtual mobile which resides in part on a high-end node, for resource intensive activities. EASY [11] extends notions from the CMU matchmaker to take context and QoS into consideration and performs indexed classification of the ontology hierarchy offline such that subsequent lookup is much faster. However all of these architectures require the existence of a high-end central node, due to the fact that semantic reasoners used by these architectures (including Prolog, Lisp, Jess, FaCT++, Pellet, RacerPro and KAON2) are all resource intensive. Limited resources are the biggest barrier when enabling mobile semantic web services for mobile terminals [12], because current reasoners cannot be deployed to resource constrained devices in their current form.

Gu et. al. [12] have developed a framework which provides an RDF parser, reasoner and sRDQL query engine which runs on mobile devices on J2ME with acceptable performance. The main drawback is that it only supports a subset of semantic technologies and the reasoning engine supports only forward chaining rule inference. It does not support backward chaining or OWL-DL reasoning.

Kleeman et. al. [13] have developed KRHyper, a novel first order logic (FOL) reasoner for deployment on resource constrained devices. In order to use DL with KRHyper it must be transformed into a set of disjunctive first order logic clauses. It does not utilise caching schemes which incur unnecessary overhead and memory consumption for smaller tasks, but optimise larger tasks. Performance comparisons with RacerPro show that it performs better for small tasks and not as well for larger tasks. This FOL reasoner meets the goal of providing competitive performance results with a DL reasoner. However, it still suffers from "Out of Memory" errors when the reasoning tasks becomes too large for a small device to handle.

Approximate reasoning is suggested to manage the trade-off between efficiency and precision by approximating the answer given. For instance [14] compares ontology terms by Google distance, but this requires Internet access to Google. [15] disregards non-horn clauses, to provide a faster but less accurate result by resulting reasoner expressivity. [16] provides a novel approach which iteratively continues matching conditions until there is no more time to continue and assumes a match if all conditions checked so far, were matched. [17] builds on this work to provide conjunctive query answering and instance retrieval. However, these approaches fail to take into consideration the importance of particular attributes and match these first to provide a service match and level of confidence.

Therefore, there is a need for an optimised semantic reasoner which performs better than currently available reasoners which is also resource-aware and can therefore reduce accuracy of results when a completely accurate result is not possible given resource or time constraints. In the next section we describe our novel architecture to meet this need.

3. COST EFFICIENT AND ADAPTIVE DISCOVERY

In this section we provide an overview of our architecture and semantic reasoning. We also provide a detailed overview of our mTableaux optimisation and adaptive reasoning strategies.

3.1 Architecture

Our service selection architecture comprises several modules, which are illustrated in figure 3. Our architecture supports two main goals 1. optimised service discovery onboard small devices, 2. adaptive discovery, such that result accuracy can be reduced when there is insufficient resources available to complete a service request.

In our architecture, the service requester submits a request *RQ* for a service to the Discovery Manager. This manager loads appropriate potential services and terms from the database of ontologies. The Discovery Manager iteratively compares each potential service with the request *RQ*. It does this by asking the Semantic Reasoner module whether the potential service advertisement can be inferred to be a member of the *RQ* definition. The Discovery Manager interacts with the Context Manager to retrieve resource context information from the user's device or explicit and implicit user preferences. For instance, closer services may be preferred to services which are further away. Therefore, preferred service advertisements (based on user preferences) are checked by the Semantic Reasoner, first.

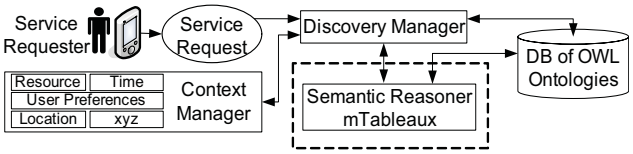

Figure 3. Pervasive Service Discovery Architecture.

The focus of this paper is the optimisation and adaptive reasoning strategies which are implemented in mTableaux, within the Semantic Reasoner module. mTableaux addresses the need for scalable reasoning on a mobile device by providing strategies to optimise the reasoning process and balance accuracy with efficiency. In the next section we describe our mobile reasoning strategies, employed by the mTableaux module, to achieve this.

3.2 Tableaux Reasoners

The effective employment of semantic languages requires the use of semantic reasoners such as Pellet, FaCT++, RacerPro and KAON2. Most of these reasoners utilise the widely used Tableaux [18] algorithm. DL Tableaux reasoners, such as Pellet, reduce all reasoning tasks to a consistency check. Tableaux is a branching algorithm, in which disjunctions form combinations of branches in the tree.

Reasoners contain a knowledge base (KB), let *K* denote this KB, to which inference queries are performed. *K* encompasses terminological knowledge *TBox* and assertional knowledge *ABox*, such that $K = TBox \cup ABox$. *TBox* encompasses class definitions and expressions while *ABox* encompasses individual and literal assertions of class membership and relations. Inferred membership for an individual *I* to class type *RQ* implies $I \in RQ$, where $RQ \in TBox$ and $I \in ABox$. $I \in RQ$ is checked by adding $\neg RQ$ as a type to *I*, in a consistent ontology. If a clash exists for all branches dependant on $\neg RQ$ for I, then membership is proven.

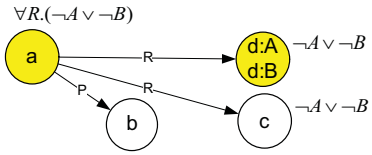

Figure 4. Example Clash.

Figure 4 presents an example in which the individual *a* is connected to the object nodes *d* and *c* via role *R* and to *b* by role *P*. Inference of $a \in RQ$ is checked by asserting $a \in \neg RQ$, where $RQ = \forall R.(\neg A \vee \neg B)$.

Application of the universal quantifier results in $\neg A \vee \neg B$ being added to *R* neighbours of *a*, which are *d* and *c*. Node *d* is a member of type *A* and *B* and therefore clashes for both elements of the disjunction, proving that *a* is a member of *RQ*. Application of any other expressions to node *a* or *d*, or any expressions to nodes *b* and *c* would not lead to a clash and such processing is unnecessary.

In order to illustrate an example of normal Tableaux execution. we wish to now check the truth of the inference $a \in RQ$, where $RQ \equiv (\forall R.A \wedge B) \wedge (C \wedge D)$. When *RQ* is negated it gives rise to the disjunction illustrated in figure 5, which expands into several sub-disjunctions. When a disjunction is applied using Tableaux it gives rise to a new branch. Branches are identified by branch number, which is incremented for each disjunction applied.

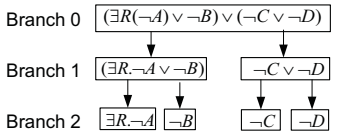

Figure 5. Example Disjunction.

Applying the disjunction from figure 5, gives rise to the Tableaux execution given in figure 6. Note that individual *a* already a member of types *B*, *C*, and *D* and is restricted by the max cardinality of 0 for role *R*. The inference test clashes for all branches and the inference is proven. Notice that types and edge relations are indexed by branch number (shown in brackets before type and edge assertions). Note branch 0, indicates explicit types. The boxes represent an individual (name is indicated before the colon) and contains several types which have been added (after the colon) to the individual either explicitly, or by the execution of the Tableaux algorithm. Individuals may connect to other nodes via roles.

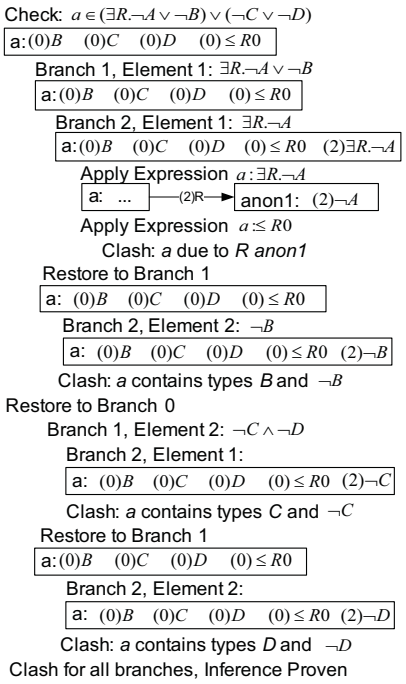

Figure 6. Tableaux Branching Example

As shown in figure 6, when a clash occurs, for disjunction *D*, the reasoner state is restored to the branch number to which disjunction *D* depends on (was expanded from). Restoring reasoner state to an earlier branch state, involves permanently removing any type or edge which was added after the branch being restored to (ie type or edge with a branch number which is larger than the branch number being restored to). For instance in figure 6, notice that application of the existential quantifier at branch 2 results in the creation of a new edge, which is later removed when the reasoner state is restored to branch 1.

In the next section we introduce and motivate our strategies to optimise semantic reasoning such that it can be completed on small devices and adaptively reduce result accuracy for faster query answering where precision is not needed or where there are insufficient resources or time to mandate such efficiency.

3.3 mTableaux Motivation

One of the key challenges facing the semantic web and its associated applications such as semantic web services, which has prevented its uptake on a large scale is performance. Even within an organisational Intranet the processing of ontologies can be extremely resource intensive. As a result current systems often query static semantic data without attempting to draw additional inferences, thereby failing to take advantage of one of the key benefits of semantics. This situation is only exacerbated when we consider the need to reason about a growing pool of interconnected ontologies which are distributed over the web.

Today's reasoners still employ logic processing approaches which assume that an entire ontology (and all ontologies which it references), are loaded into memory. The logic expressions which make up the in-memory ontology, are processed until all possible inferences are established to complete the ontology. This approach does not take into consideration the increasingly distributed, heterogeneous pool of interconnected semantic documents. As this pool continues to grow in size, it is simply not feasible to assume that all related semantic data can be loaded and completed in this way.

The fact that semantic languages allow the use of logic to express information provides a clear benefit over syntactic approaches. However, traditional approaches to logic based reasoning are based on absolutes which is an assumption that must be relaxed when dealing with the web. A key strength of the semantic web languages is their support for distribution. Semantic web ontologies do not need to conform to any centrally managed structure, they can be developed as independent portions of self contained information. The web is also incredibly heterogeneous and ontologies will be created by many different entities which may have a different and sometimes contradicting view of the same or related knowledge. Ontologies will often be created by non-experts or machines, which means that they will contain inaccuracies or may be sloppy in nature. In addition, different users of the web have different requirements. For instance, some users may seek an answer at the highest possible accuracy, while other users may not require the same degree of accuracy but would rather obtain an answer more quickly. These features of distribution, inaccuracies, heterogeneity and different perspectives and requirements which characterise the world wide web run contrary to the absolute and exact nature of logic languages and logic based reasoners. Current reasoners only cater for the notion of exact truth or failure in inference, and there is no provision for partial matching. Given the growing body of semantic knowledge on the web, guaranteeing such exactness may require excessive processing time and may not be feasible in the amount of memory available. A more fine-grained approach for inference matching is needed. Current reasoners understand terms such as "*a* is-a *X*", "*a* is-the-same-as *b*" while it may be more appropriate to say "*a* is-almost-a *Y*", "*a* is-similar-to *b*" and "Yes, except for a few". This situation is illustrated in figure 7, where a service advertisement *a* almost matches a request *RQ*, clearly a must have some relevance to the user is relevant to the user, however current reasoners fail to match *a* with *RQ*.

Figure 7. Advertisement *a* almost matches request *RQ*.

Furthermore, current reasoners do not provide incremental, gradual query answering. Rather, they invest significant time in completing a search before an exact answer of absolute certainty is given, even if the user does not require this level of accuracy answer. Current reasoners do not support notions of importance or degree of match, for instance current reasoners support questions such as "Who has a white van and a red car convertible?" while it may be more appropriate to ask "Who has a white van? if that person has a red convertible as well that would be great but it is not very important to me". Current reasoners consider all requirements to be of equal value and the order in which these are checked is arbitrary. A far better approach is to allow importance to be associated with each request condition and match the most important conditions first in order to make the best use of the processing time available, as shown in figure 8.

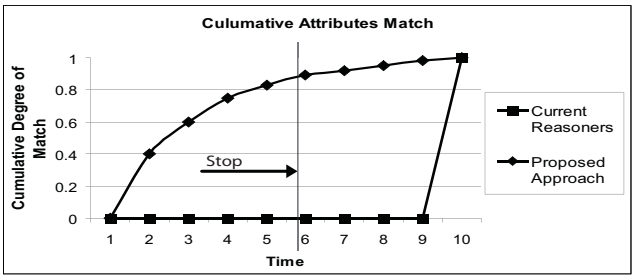

Figure 8. Gradual vs exact inference

Current reasoners give no result until the final step. Conversely, if the most important request conditions are matched first, a gradual result can be returned to the user if the reasoner is stopped prematurely. Moreover, current reasoners would return false or a 0.0 match if stopped early, while as shown in figure 8, more than half of the degree of match is found in the first 3 seconds, and the remaining half is found in the last 7 seconds. If the reasoner is stopped after 5 seconds the result 0.84 should be returned.

In summary, our work focuses on two main objectives:

- Toward the goal of improved performance, optimisation strategies are required to make semantic reasoning feasible for both high-end Internet and mobile environments, so that the semantic web can better reach its potential for data inference.
- To support situations in which an inference task is too large, even with optimisations, there must be some flexibility introduced to reasoners. Moreover, current reasoning approaches need to be relaxed to handle premature ending of a reasoning task, inconsistent data and partial inferences, as well as providing approaches to reason about the relevant data set rather than entire ontologies. Where there is insufficient time or resources available to complete a reasoning task, a less accurate result should be provided to the user with a degree of match rating a confidence level of this result.

In this paper we briefly introduce our optimisation strategies from previous work, which include: 1. selective application of consistency rules, 2. skipping disjunctions, 3. applying logical expressions are most likely to lead to a clash, first, by searching for potential clash paths from: 3a. disjunction branch element terms, as they are applied, and 3b. disjunctions as they become applicable (by the selective consistency strategy). However, the main contribution of this paper is to provide adaptive strategies for more flexible reasoning which include: 1. adaptive reasoning, 2. on-demand ontology loading and 3. clash detection pre-processing. The optimisation strategies are discussed in the next section.

4. OPTIMISED MATCHING

Our optimisation strategies address the goal of dramatically improving the performance of inference tasks to remove performance as a factor which prevents the user of reasoners for processing semantic information in both desktop/Internet and mobile environments. Our optimisation strategies achieve this using two approaches. The first two strategies drop expressions which are not considered relevant to the reasoning task in order to reduce the size of this task. The third strategy attempts to re-order the application of expressions in an attempt to apply first the expressions which are more likely to lead to a clash. The strategies which drop expressions, do not guarantee completeness, however in realistic datasets such as that used in [20] and section 6 we found no deterioration in result accuracy. We briefly outline our optimisation strategies in the following (more details can be found in [20]).

Our selective consistency strategy is as follows. Expansion of expressions and application of completion rules assert class types to individuals, which may generate these clashes. The only construct which, when applied, results in assertion of types to individuals other than the one which it is applied to, is the universal quantifier, of the form $\forall R.C = \{\forall b.(a,b) \in R \rightarrow b \in C\}$ [19], where R denotes a relation and C denotes a class concept. For instance, if $I \in \forall R.C$ and if the triple <I, R, Z>, exists in K denoting that I has a relation R to the object Z, application of $\forall R.C$ for I, results in asserting $Z \in C$. Therefore, when $\neg RQ$ is added to I such that $I \in \neg RQ$, we define the subset of relevant as being limited to I and those individuals which branch from this individual as objects of roles specified in universal quantifiers. For example in figure 4 (see section 3.2), nodes a, c and d would be in the subset of relevant individuals, because they are connected to a via role R in $\forall R.(\neg A \vee \neg B)$, while node b would not be in the subset.

Our skip disjunction strategy is described as follows. Disjunctions are only applied, if they relate to the request type RQ, otherwise it is skipped. A disjunction is applied when one of its elements contains a type which can be decomposed from RQ. Decomposition involves selecting and unfolding atomic class types, elements of conjunctions/disjunctions or role fillers of universal quantifiers. For instance in figure 4 (see section 3.2), the disjunction $\neg A \vee \neg B$, would be applied by this strategy, while $\neg C \vee \neg D$ would not.

Our weighted strategies can be described as follows. The order in which logical expressions are applied is determined by weightings associated with these expressions. Weightings are established by searching for pathways to potential future clashes, and incrementing the weight of all expressions associated with this pathway. Figure 4 (see section 3.2) illustrates an example pathway from node a to d, via $\forall R.(\neg A \vee \neg B)$ and $\neg C \vee \neg D$. Under our disjunction weighting approach, all applicable (as deemed by selective consistency) disjunctions may be checked for potential clash pathways. Alternatively, a more specific approach is term weighting, where a search for pathway from a specific term to a clash occurs. We perform this search on terms which are asserted as a result of the application of a disjunction branch element. The assumption is that disjunction elements asserted to the knowledge base, may give rise to a clash which is not apparent until subsequent expressions on specific individuals are applied. Application of these expressions should be fast tracked. Term weighting may be utilised even when disjunction weighting is enabled, based on the assumption that once a disjunction branch element term is applied there is an increased incentive to improve the weight of any potential pathway from this element to a clash. For example, in figure 4 (see section 3.2), assume the application of element $\neg A$ to node d, required application of other

expressions before a clash is found, the weight of these rules should be further increased since processing has already been invested in this pathway. [20] provides more detail on our optimisation strategies.

While these strategies optimise the reasoning process, it may still be the case that mTableaux enabled optimised reasoning may still require too much time to complete, especially when reasoning with large ontologies and requests. Therefore, we leverage this approach by providing adaptive strategies to balance result accuracy with efficiency. We discuss our adaptive strategies in the next section.

5. ADAPTIVE REASONING

This section describes our approach for taking resource availability into consideration during the service matching process, in order to manage the trade-off between efficiency and accuracy. We begin by we discussing the depth-first strategies which current Tableaux reasoners and approximate reasoning approaches such as [16], utilise to iteratively check branch combinations. Then, in the subsequent sections we introduce our own novel weighted approach which matches the most important conditions first.

5.1 Current Depth-first Reasoning

This section is an overview of depth-first reasoning approaches which are utilised by current Tableaux reasoners and approximate reasoning approaches such as [16]. Assume the user wishes to find a movie cinema with Internet café. The request query is shown in expression 1, asking for retail outlets which sell coffee, Internet over WiFi and movie screenings.

$$\text{RetailOutlet} \wedge \exists \text{ sellsProduct.Coffee} \wedge \exists \text{ sellsProduct.MovieScreening} \wedge \exists \text{ sellsProduct.(Internet} \wedge \exists \text{ supportsComm.WiFi)} \quad (1)$$

Tableaux reasoners utilise a depth-first approach in evaluating the above expression, as shown in figure 9.

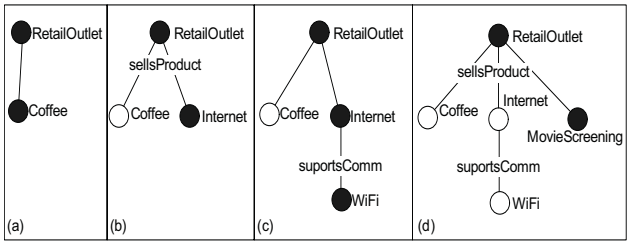

Figure 9 Depth-first Example

When matching a particular individual *I* with a request expression *RQ* the reasoner continues to match each iteration shown in figure 9, until all conditions have been checked or an open branch is found (no clash). Therefore, those results given by each of the steps shown in figure 9 are subsets of the results given by the previous steps. Query containment can be described as follows.

Let RQ_1, RQ_2 denote request queries over a knowledge base *K* of triples. Let *K* denote the knowledge base, *CS* the set of class definitions, *RS* the set of relation definitions and *OS* the object assertions, that *K* = <*CS, RS, OS*>. Let Q'_1, Q'_2 denote the results for queries RQ_1 and RQ_2, respectively. RQ_1 is a subset of RQ_2 if RQ'_1 is a subset of RQ'_2 for all objects in the knowledge base as shown in equation 2 and query equality is shown in equation 3.

$$Q_1 \subseteq Q_2 \text{ iff } \{ \forall OS.(Q'_1 \text{ v } Q'_2) \} \quad (2)$$

$$Q_1 \equiv Q_2 \text{ iff } Q_1 \subseteq Q_2 \wedge Q_2 \subseteq Q_1 \quad (3)$$

Approaches such as [16] employ approximate reasoning approaches to support partial and anytime matching, which are based on the assumption that each of the previous iterations are subsets of the final iteration in figure 9, when checking membership of I $\in$ RQ. Therefore, if reasoning is stopped early the reasoner returns a true match result, if all of the checked conditions were proven. This result is said to be an approximation or super result for the true result. However there shortcomings with this approach:

1. particular request conditions or sub-conditions may have a different level of importance to the user: the most important should be checked first
2. approximate reasoning approaches still provide only a Boolean result and do not provide a degree of match result to the user. If the approximation matches, the final result is said to be true
3. pervasive environments are inherently dynamic and contain heterogeneous data, therefore there may be instances where the reasoner should check subsequent conditions in the query, even if some of them were not met by the request.

We employ our novel weighted adaptive reasoning approach to overcome these shortcomings, which is introduced in the next section.

5.2 Weighted Adaptive Reasoning

Since certain query conditions may have a different level of importance to the user, we allow the association of user specified weightings with each condition and sub-condition. Weighs control query execution order, rather than depth or breath-first ordering. Assume that the user deems that while access to the Internet is more important than coffee, coffee is more important than WiFi Internet (the user may be happy to use a fixed PC). This gives rise to the following request query execution (note for illustrative purposes, coffee has been extended to include Tea, instant coffee and percolated coffee):

RQ_1: RetailOutlet

RQ_2: RetailOutlet $\wedge$ $\exists$ sellsProduct.MovieScreening

RQ_3: RetailOutlet $\wedge$ $\exists$ sellsProduct.MovieScreening $\wedge$ $\exists$ sellsProduct.Internet

RQ_4: RetailOutlet $\wedge$ $\exists$ sellsProduct.MovieScreening $\wedge$ $\exists$ sellsProduct.Internet $\wedge$ $\exists$ sellsProduct.(Tea $\wedge$ (InstantCoffee $\wedge$ PercolatedCoffee))

RQ_5: RetailOutlet $\wedge$ $\exists$ sellsProduct.MovieScreening $\wedge$ $\exists$ sellsProduct.(Internet $\wedge$ $\exists$ supportsComm.WiFi) $\wedge$ $\exists$ sellsProduct. (Tea $\wedge$ (InstantCoffee $\wedge$ PercoluatedCoffee))

Since Tableaux checks inferences by asserting the negation of the inferred type to an individual, we provide the negation of RQ_5 in expression 4.

$$\neg\text{RetailOutlet} \sqcap \forall \text{ sellsProduct. } ((\neg\text{InstantCoffee} \sqcup \neg\text{Percolated}) \sqcup \neg\text{Tea}) \sqcap \forall \text{ sellsProduct. } \neg\text{MovieScreening} \sqcap \forall \text{ sellsProduct. } (\neg\text{Internet} \sqcap \forall \text{ supportsComm. } \neg\{\text{WiFi}\}) \quad (4)$$

Expression 4 gives rise to the disjunctions and sub-disjunctions illustrated in figure 10, where elements in the first level disjunction expand into other disjunctions. When a disjunction element expands into another disjunction, the expanded disjunction is said to be dependant on the disjunction from which it was expanded from. For instance element 9 is dependant on element 7, and 7 on 5.

Figure 10 also illustrates the weights given to each condition and sub-condition. Let *w* denote the user specified weight given to each condition or sub-condition. If a user has not specified a weight, 1.0 is used by default. Let *rw* denote a relative weight, which is the user specified weight *w* multiplied by the relative weight *rw* of the element which it depends (*rw* is equivalent to *w* if it is the top level disjunction). Rather than depth-first ordering (see previous section), we wish to execute disjunction elements in relative weight *rw* descending order. Let *nrw* denote a normalized relative weight. Normalisation implies that the sum of all *nrw* values for a disjunction *D* (ie the *nrw* of each of its elements) is equal to the *nrw* of the disjunction to which *D* depends (*nrw* is used in the next section). We also define a special identifier in order to index each element uniquely. We call this index a *tree-id*.

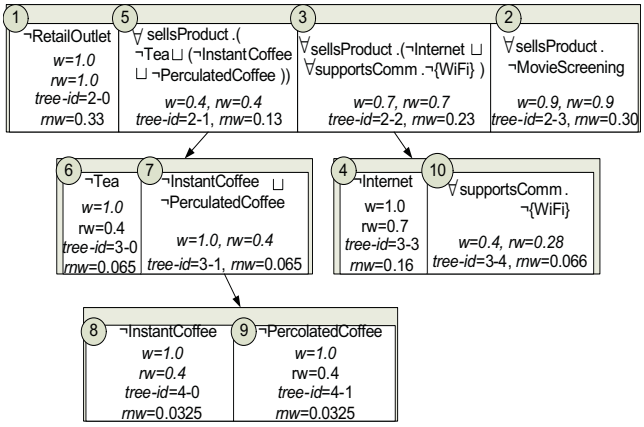

Figure 10. Weighted Approach to Query Execution

Recall from section 3.2, when a disjunction is applied it gives rise to a branch in the reasoner. Each disjunction element is applied as a separate branch. The order in which elements (branches) in figure 10 were applied, is given by the number on the top left of each element (the order of elements with the same *rw* is random). In figure 10, each request query *RQ* includes the following elements: $RQ_1 = \{1\}$, $RQ_2 = \{1, 2\}$, $RQ_3 = \{1, 2, 3, 4\}$, $RQ_4 = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, $RQ_5 = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. That is, if the request query RQ_i is satisfied, then all of the elements to which it relates, have also been applied and satisfied.

As described in the previous section, Tableaux is a depth-first branching algorithm, in which branches depend on previous branches in the tree. This was also shown in section 3.2. Therefore, for instance, after element 4 is found to clash, the reasoner state is restored to element 1, before element 5 is executed. Therefore, the state transition from element 9 to element 10, requires the re-application of element 3 (and any completion rules which were fired between element 4 and 5 arising from other concepts in the ontology. Such an approach therefore results in extra processing, which is unnecessary. Therefore, ordering disjunction elements by importance in conjunction with a depth-first approach would potentially increase processing rather than decrease it. In order to overcome this problem, we have developed a new mechanism to manage reasoner state, which replaces depth-first. Under depth-first, reasoner state is maintained according to branch number iteration (see section 3.2). Under our approach state is maintained by *tree-id*. Our novel approach to state maintenance is described in the next section.

5.3 State Maintenance

Under our approach we index reasoner state by element identifier *tree-id*, rather than by branch iteration (see section 3.2), as in current Tableaux reasoners. And rather than restoring reasoner state to an earlier state (as under depth-first), we preserve an indexed reasoner state for each branch until all dependant nodes have been evaluated. The reasoner state is indexed in order to provide different state views, depending on which branch node is being evaluated.

As shown figure 10 each element is associated with a unique *tree-id*, which we utilise to index reasoner state. Let $e^{\text{tree-id}}$ denote the *tree-id* for element *e*. A *tree-id* comprises two or three positive integer values, each separated with a dash. The digit before the dash indicates the disjunction level of the element and the digit after the dash indicates the element position in the level. Note the position index is incremented for each element including those in different disjunctions on the same level. We denote the third value as *copyid*, an optional value used as a delimiter to differentiate between same disjunctions that are applied to multiple individuals. This only occurs when a universal quantifier applies a disjunction as its role filler, for example if $\forall R(A \sqcup B)$ applies the disjunction $A \sqcup B$ to two individuals, element *A* is given a *tree-id*=x-x-0 for one individual and *tree-id*=x-x-1 for the other.

Each element depends on previous elements, for example ¬InstantCoffee depends on elements {0-0, 1-0, 2-1, 3-1, 4.0}. Let *dep-set* denote this set of elements and let $e^{dep\text{-}set}$ denote the *dep-set* for an element *e*. *tree-id* 0-0 is reserved for explicit type assertions (with no dependency) and *tree-id* 1-0 is reserved as a reference point to which the first disjunction depends. Therefore, all elements depend on 0-0 and 1-0.

In order to maintain an indexed reasoner state, to provide different state views based on *tree-id*, we define a set *S*, which is indexed by *tree-id*. Each value *v* in the set *S* is associated with a *tree-id*, to form the pair <*v*, *tree-id*>. Multiple *v* elements in the set *S* can have the same *tree-id*. The view of *S* at any one time, depends on the $e^{dep\text{-}set}$ where *e* is the current/last disjunction/branch element in the reasoner. This view is given by *view(S, dep-set)* and contains only those values *v* which have a *tree-id* contained within $e^{dep\text{-}set}$. For example, when applying the ¬InstantCoffee element 8 in figure 10, $e^{tree\text{-}id}$ = 4-0 and $e^{dep\text{-}set}$ = {1-1, 2-1, 3-1, 4-0}. And for example, where $S = \{<v_1, 1\text{-}0>, <v_2, 1\text{-}1>, <v_3, 1\text{-}1>, < v_4, 1\text{-}2>,$

$<v_5, 2\text{-}1>$, $<v_6, 2\text{-}3>$, $<v_7, 3\text{-}1>$, $<v_8, 4\text{-}0>$} then *sv*(*S*, *dep-set*) = {$<v_2, 1\text{-}1>$, $< v_3, 1\text{-}1>$, $< v_5, 2\text{-}1>$, $< v_7, 3\text{-}1>$, $< v_8, 4\text{-}0>$ }.

We utilise instances of *S* to maintain:

1. the type labels which are added to a particular individual in the knowledge base
2. the graph of edges (role assertions) in the knowledge base.

This replaces the branch number indexing of types and edges which occurs when depth-first is employed, as discussed in section 3.2.

As stated in earlier sections, under our approach we support partial matching. Therefore, we continue matching even if one request condition (disjunction element) does not match. This is distinct from the current Tableaux depth-first approach which stops and returns a false result as soon as any condition does not match. However, introducing partial matching can involve significant processing which may be unnecessary. For instance if several important attributes fail to match, there is no sense in checking all other remaining insignificant attributes. Therefore, in the next section we introduce a strategy for deciding whether to continue or stop matching.

5.4 Low Resources: Stop Matching

In the case that important request conditions failed to match a particular service advertisement, a better use of resources may be to try another service instead of continuing to attempt to match the remaining conditions which are less important. We specify the decision as to whether to keep matching a service as being a trade-off between the three factors 1. the current service result/degree of match, 2. the weight of the next attribute and 3 resource availability.

Let *service-result* denote the degree of match which is calculated by the sum of the normalised-relative *nrw* (see section 5.1.2) weights for each condition which was found to successfully match the service advertisement, and let *potential-result* denote the total possible sum of all normalised-relative condition weights *nrw* (including any unchecked or non-matching conditions). We denote *curr-result* as the ratio of *service-result* / *potential-result*.

We model a linear relationship between *curr-result* and the normalised-relative weight of the next request condition to match *nrw(next)*, on a graph as shown in figure 10, where the x axis denotes the *curr-result* and the y axis denotes the attribute weight *nrw(next)* required, in order to keep matching. A linear line which crosses the points (x_1, y_1) and (x_2, y_2). These points are determined as follows: y_1 and x_2 are always 1 ($y_1 = 1$ and $x_2 = 1$) and the coordinates x_1 and y_2 are determined on the fly based on the amount of resources or time remaining where $0 \leq x_1 \leq 1$ and $0 \leq y_2 \leq 1$.

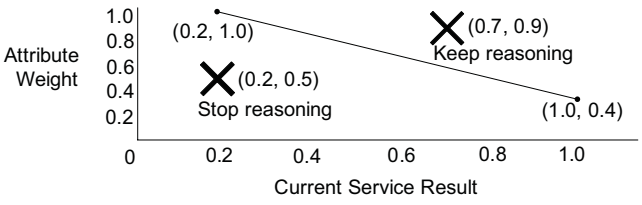

Figure 11. Deciding whether to stop matching.

As illustrated in figure 11, if the weight of the next request condition to match *nrw(next)* is above the line we keep matching, while if *nrw(next)* is below the line stop matching.

In the next section we discuss the implementation of the optimisation and adaptive strategies from sections 4 and 5, and provide an evaluation to illustrate their effect on performance and accuracy.

6. Implementation and Evaluation

In this section we discuss the implementation of our mTableaux optimisation and adaptive strategies and provide a performance evaluation, of these strategies. Our evaluation utilizes a Product case study implemented in an ontology comprising 204 classes and 241 individuals. In our case study, Bob is searching for a Movie Cinema with Café which provides WiFi internet and a public phone, in a foreign city centre. The request is an extension of that given in Q_5 (negated in expression 4) with each condition associated with the weights provided in figure 10, in section 5.2 (condition weights are only applicable to the adaptive reasoning strategy).

6.1 Implementation

We have implemented both our optimisation strategies and our adaptive strategies as an extension to the Pellet 1.5 reasoner which supports OWL-DL with SHOIN expressivity. Pellet is open source, allowing us to provide a proof of concept and compare performance with and without the strategies enabled. We selected Pellet over FaCT++ because it is written in Java, making it easily portable to small devices such as PDAs and mobile phones, while FaCT++ is written in C++. An addition, we are using Jena as the ontology repository used by Pellet to read the ontology. We implemented the optimisation strategies from section 4. We evaluate the impact these have on performance in the next sections.

6.2 Mobile Performance and Accuracy Results

We performed an evaluation on a HP iPAQ hx2700 PDA, with Intel PXA270 624Mhz processor, 64MB RAM, running Windows Mobile 5.0 with Mysaifu Java J2SE Virtual Machine (JVM) [21], allocated 15MB of memory. We present results for the optimisation and adaptive strategies the next two sections, respectively.

6.2.1 Optimisation Strategies - Results

Figure 12 illustrates the performance results for the optimisation strategies. We performed a number of tests, each with different combinations of the strategies enabled. The initials above each test on the graph indicate the optimisation strategies which were enabled: selective **C**onsistency, **S**kip disjunctions, **D**isjunction ranking and **T**erm ranking.

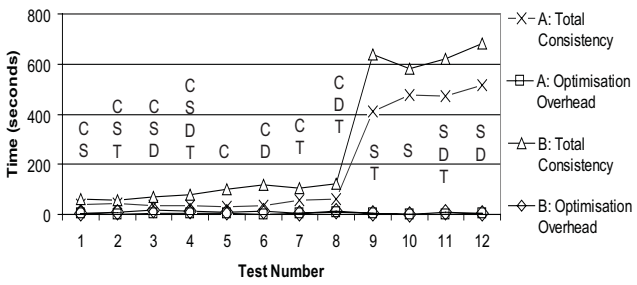

Figure 12. Processing time for Product scenario.

We performed each test against a: A. matching service individual and B. non-matching service individual. Where no optimisation strategies were enabled, an "Out of Memory" error was returned. Our results show that the selective consistency and skip disjunction strategies work best and drastically reduce processing time, with very little overhead. More detailed results can be found in [22] and [20].

6.2.2 Adaptive Strategies - Results

To demonstrate the feasibility of our adaptive reasoning strategy we performed a number of tests as outlined in table 1. We performed one test in which all conditions matched and 3 where one condition was not matching. Note, that the optimisation strategies were enabled for these tests and all service result.

In our tests we used the resource preferences illustrated in figure 13, in order to determine whether to keep matching. Therefore, matching stopped early in tests 2 and 3.

Table 1. Adaptive Reasoning Results

Test	Service Result		Time (excl. init.)	Total Time
	Actual	Theoretical		
1. Match	1.0	1.0	24s	53s
2. No Movie Cinema	0.33	0.69	7s	33s
3. No Internet	0.63	0.76	17s	46s
4. No Coffee	0.86	0.86	25s	54s

In table 1, service result is the sum of normalised-relative *nrw* weights associated with each request condition which was successfully matched against the service advertisement. We also provide the processing time incurred to perform the consistency check (checking whether a request condition matched, excluding initialisation), and the total time (including initialisation). The "theoretical" service result indicates the degree of match value given if all request conditions are checked, while the "actual" service result is that which was returned by the discovery architecture. The actual result returned is sometimes lower than the theoretical degree of match because reasoning can be stopped early and some conditions not checked.

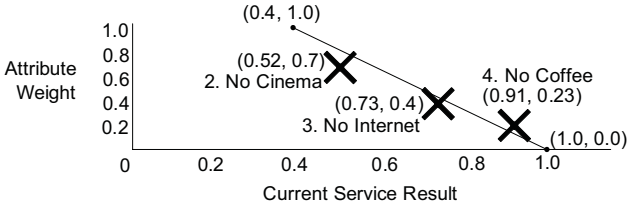

Figure 13. Preferences for deciding whether to stop matching**.**

Our results show that performance of the adaptive strategy is similar to the results from the previous section where all conditions match the potential service. However, when important conditions do not match, valuable processing time is saved by avoiding the checking of less important conditions, at the expense of result accuracy. Our findings therefore show that the adaptive reasoning strategy meets our goal of balancing the trade-off between efficiency and accuracy.

In summary our optimisation strategies drastically reduced the processing time required to match request with potential service individuals, such that this can be completed on a small device. When optimised reasoning is still too large to perform in the available time, we provide adaptive strategies which iteratively match the most important conditions (as deemed by the user) first and avoid matching less important conditions when important conditions failed. This allows the reasoner to move on to checking another potential service, instead of wasting scarce resources on continuing to check a service that has already failed vital conditions. Evaluation of our adaptive approach shows that our strategies effectively meet this goal.

7. CONCLUSION AND FUTURE WORK

We have presented novel strategies for improving the scalability of the Tableaux algorithm for mobile semantic reasoning. mTableaux was shown to significantly improve the performance of pervasive discovery reasoning tasks so that they can be completed on small resource constrained devices. However, in some cases the request may still be too complex or there may exist too many potential services to check each service completely, within a reasonable amount of time. In addition, for the semantic web to reach its true potential it must effectively handle explicit and inferred relationships between heterogeneous data and support partial and incomplete matching. Therefore, as the main contribution for this paper we have developed a strategy for adaptive reasoning. As opposed to currently used Tableaux depth-first strategies, our we use a weighted approach which matches the most important inference conditions first, to make the best use of the processing time and resources available. If the reasoning process is stopped prematurely, a known service result is given to the user. In the evaluations of our prototype we demonstrate the significant processing savings achieved by stopping the reasoning process as soon as important request conditions fail to match.

In future work, we are developing more formal metrics for assessing degree of match and in particular a confidence or uncertainty level in the degree of match given. We seek to create and evaluate more scenarios and stop reasoning based on two constraints: 1. time bound responses in which a user seeks a result within a specified time frame, and 2. confidence bound results, in which the user seeks a result only to a certain level of confidence,

eg the degree of match after checking 90 percent of the request conditions. In addition it is not feasible to require the entire ontology to be loaded into memory in such a distributed environment as the web, or constrained memory of a mobile device. Only a small part of the ontology (or those ontologies imported) may be relevant to the reasoning ask. Therefore, we are developing an on-demand ontology loading strategy which only loads terms from an ontology as these are needed by the reasoner.

8. REFERENCES

[1] Gehlen G. and Pham L. Mobile Web Services for Peer-to-Peer Applications. In proc. Consumer Communications and Networking Conference (CCNC), 3 - 6 January, 2005. p. 427 - 33.

[2] Tergujeff R., Haajanen J., Leppanen J. and Toivonen S. Mobile SOA: Service Orientation on Lightweight Mobile Devices. In proc. International Conference on Web Services (ICWS), 9 - 13 July, Salt Lake City, USA, IEEE, 2007. p. 1224 - 5.

[3] Srirama SN., Jarke M. and Parinz W. Mobile Web Service Provisioning. In proc. Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW), 19 - 25 February, 2006.

[4] Kim Y-S. and Lee K-H. A Light-weight Framework for Hosting Web Services on Mobile Devices. In proc. 5th European Conference on Web Services (ECOWS), 26 - 28 November, IEEE, 2007. p. 255 - 63.

[5] Chatti MA., Srirama S., Kensche D. and Cao Y. Mobile Web Services for Collaborative Learning. In proc. 4th International Workshop on Wireless, Mobile and Ubiquitous Technology in Education November, IEEE, 2006. p. 129 - 33.

[6] Arnold K., O'Sullivan B., Scheifler RW., Waldo J. and Woolrath A. The Jini Specification, Addison-Wesley 1999.

[7] Srinivasan N., Paolucci M. and Sycara K. Semantic Web Service Discovery in the OWL-S IDE. In proc. 39th Hawaii International Conference on System Sciences, Hawaii, 2005.

[8] Küster U., König-Ries B. and Klein M. Discovery and Mediation using DIANE Service Descriptions. In proc. Second Semantic Web Service Challenge 2006 Workshop, June 15 - 16, Budva, Montenegro, 2006.

[9] Ranganathan A. and Campbell RH. A Middleware for Context-Aware Agents in Ubiquitous Computing Environments. In proc. ACM/IFIP/USENIX International Middleware Conference, June, Rio de Janeiro, Brazil, 2003. p. 143 - 61.

[10] Noll J., Alam S. and Chowdhury MMR. Integrating Mobile Devices into Semantic Services Environments. In proc. 4th Internation Conference on Wireless and Mobile Communications (ICWMC), July 27 2008 - August 1, Athens, IEEE, 2008. p. 137-43.

[11] Mokhtar SB., Preuveneers D., Georgantas N. and Issarny V. EASY: Efficient SemAntic Service DiscoverY in Pervasive Computing Environments with QoS and Context Support. Journal Of System and Software. 2008, 81(5).

[12] Gu T., Kwok Z., Koh KK. and Pung HK. A Mobile Framework Supporting Ontology Processing and Reasoning. In proc. 2nd Workshop on Requirements and Solutions for Pervasive Software Infrastructure (RSPS) in conjunction with the 9th International Conference on Ubiquitous Computing (Ubicomp '07), September, Austria, 2007.

[13] Kleemann T. Towards Mobile Reasoning. In proc. International Workshop on Description Logics (DL2006), May 30 - June 1, Windermere, Lake District, UK, 2006.

[14] Gligorov R., Kate Wt., Aleksovski Z. and Harmelen Fv. Using Google Distance to Weight Approximate Ontology Matches. In proc. 16th international Conference on World Wide Web ACM, 2007.

[15] Hitzler P. and Vrandecic D. Resolution-Based Approximate Reasoning for OWL DL In proc. Semantic Web - ISWC, Springer Berlin / Heidelberg, 2005.

[16] Stuckenschmidt H. and Kolb M. Partial Matchmaking for Complex Product and Service Descriptions. 2008.

[17] Wache H., Groot P. and Stuckenschmidt H. Scalable Instance Retrieval for the Semantic Web by Approximation In proc. Web Information Systems Engineering – WISE 2005 Workshops, Springer Berlin / Heidelberg, 2005.

[18] Horrocks I. and Sattler U. A Tableaux Decision Procedure for SHOIQ. In proc. 19th International Conference on Artificial Intelligence (IJCAI 2005), 2005.

[19] Baader F., Calvanese D., McGuinness DL., Nardi D. and Patel-Schneider PF. The Description Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press 2003.

[20] Steller L. and Krishnaswamy S. Pervasive Service Discovery: mTableaux Mobile Reasoning. International Conference on Semantic Systems (I-Semantics). Graz, Austria 2008.

[21] Mysaifu. Mysaifu JVM. (2009), http://www2s.biglobe.ne.jp/~dat/java/project/jvm/index_en.html

[22] Steller L. and Krishnaswamy S. Optimised Semantic Reasoning for Pervasive Service Discovery. International Conference on Service Oriented Computing (ICSOC). Sydney, Australia 2008.