A Weighted Approach to Partial Matching for Mobile Reasoning

Luke Albert Steller, Shonali Krishnaswamy, and Mohamed Methat Gaber

Faculty of Information Technology, Monash University, Melbourne, Australia
{Luke.Steller,Shonali.Krishnaswamy,Mohamed.Gaber}@infotech.monash.edu

Abstract. Due to significant improvements in the capabilities of small devices such as PDAs and smart phones, these devices can not only consume but also provide Web Services. The dynamic nature of mobile environment means that users need accurate and fast approaches for service discovery. In order achieve high accuracy semantic languages can be used in conjunction with logic reasoners. Since powerful broker nodes are not always available (due to lack of long range connectivity), create a bottleneck (since mobile devices are all trying to access the same server) and single point of failure (in the case that a central server fails), on-board mobile reasoning must be supported. However, reasoners are notoriously resource intensive and do not scale to small devices. Therefore, in this paper we provide an efficient mobile reasoner which relaxes the current strict and complete matching approaches to support anytime reasoning. Our approach matches the most important request conditions (deemed by the user) first and provides a degree of match and confidence result to the user. We provide a prototype implementation and performance evaluation of our work.

1 Introduction

The number of mobile subscribers is reaching the 3 billion mark, world wide [20]. This provides significant opportunities for new mobile applications which meet mobile user demands for improved access to information in their environment. The real usefulness of these small devices is in their interaction [23], that is, information sharing. [24] advocates the usefulness of service oriented architectures and accurate service discovery for mobile services, because of the need to support user mobility in dynamic environments. In addition, 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 [19,16]. Most current service discovery architectures perform matching on a high-end broker server. However, "for dynamic ad-hoc mobile environments, assuming the existence of devices that are stable and powerful enough to play the role of the central service registries [or brokers] is inappropriate" [16]. In addition, with the growth of increasing services, it is seen that such a centralised approach could become a bottle neck for the whole system [22]. Cost is also a significant factor which determines whether users are likely to use a mobile service. There are clear evidence in studies that the benefits must exceed the cost [9]. It is also noteworthy,

A. Bernstein et al. (Eds.): ISWC 2009, LNCS 5823, pp. 618–633, 2009.

that in terms of monitoring costs as well as cost energy efficiency, communication from mobile devices to a remote location has been established as more expensive than performing computation on the mobile device.

Employing a partially or completely decentralised approach in which the matching occurs on the mobile device itself overcomes all of these problems. Onboard reasoning would remove a central point of failure, deployment is cheap and scalable because adding more users does not involve any additional network provision and elevates privacy concerns. The ontologies used in the reasoning process could be collected by devices as the user walks past other devices, short-range download points (such as a Kiosk or shopping centre entrances) or downloaded previously (eg at home or work) from the Internet. However, mobile service discovery and matching must meet two important user requirements. Matching must be accurate / useful [9] and fast [14].

Accuracy can be achieved by adopting semantic web ontology languages (OWL) and using logic based reasoners to infer relationships to model requirements and using logic reasoners to infer new information from these ontologies. Most current semantic reasoners utilise description logic (DL) which provides the most expressive, decidable OWL logic. However, these logic reasoners are notoriously slow and resource intensive [25]. As such current reasoners cannot be ported to resource constrained mobile devices such as smart phones and PDAs, in their current form, especially since mobile users generally require a result within 10-15 seconds [14]. Current reasoners give only a *true* or $false$ result, or no result at all, if the inference task is not completed in full. [4] advocates the need to relax these traditional notions of absolute accuracy and completeness for increased performance without reduced expressiveness. Therefore, in this paper, we provide such an approach, a prototype and an evaluation, which has the following features:

1. match the most important request conditions first (as deemed by the user explicitly or implicitly by preferences);
2. handle heterogeneous inaccurate data by continuing to reason after a term failed (if there is sufficient time);
3. handle time/resource pressures by supporting early stopping of the reasoner and providing a current degree of inference to the user.

The remainder of the paper is structured as follows. In section 2 we describe related work. In section 3 we discuss the current approach to reasoning using Tableaux. In section 4, we provides an adaptive reasoning approach to meet the three goals above. In section 5 we discuss our implementation and provide evaluations of our work. Finally in section 6 we conclude the paper.

2 Related Work

While current service discovery architectures such as Jini [1] do not make use of semantic languages, there is a growing emergence of OWL-S semantic matchmakers such as CMU Matchmaker [15] which requires a centralised high-end

node to perform reasoning. Architectures such as DIANE [11] and Gaia [13] provide semantically driven context middleware. EASY [12] takes 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 their reliance on current reasoners.

Gu et al. [6] an RDF reasoner which runs on mobile devices on J2ME with acceptable performance, but it only supports a subset of semantic technologies and forward chaining rule inference. Kleeman et. al. [10] have developed KRHyper, a first order logic (FOL) reasoner for deployment on resource constrained devices. However, performance is comparable with RacerPro. [5] compares ontology terms by Google distance, but this requires Internet access to Google. [7] disregards non-horn clauses, to provide a faster but less accurate result by reducing expressiveness. These works do not implement logical level optimisations to enable mobile reasoning, other than by reducing the language expressively.

The work provided in [18] approximates class membership inference and iteratively matches all inference conditions. The reasoner can be stopped maturely, and a *true* result is given if the current approximation / iteration holds. [21] provides conjunctive query answering and instance retrieval. The main limitation of these approaches is that they rely on the Tableaux algorithm. It is evident in [2] that DL Tableaux checks conditions in depth first order. Current approaches do not take level of importance of each condition into consideration and do not provide degree of match / confidence metrics. Therefore, we provide an approach to address this need. In the next section we describe current Tableaux reasoning then introduce our novel architecture in section 4.

3 Tableaux Reasoning

The power of semantics means that in addition to explicit definitions and assertions, implicit knowledge can be made explicit using inference. The most expressive OWL language which retains decidability [3] is known as OWL-DL which is based on description logic (DL) [2]. OWL-DL inference proves include Pellet[1], FaCT++[2], RacerPro[3] and KAON2[4]. Most of these reasoners use the Tableaux algorithm which has "dominated recent DL research" [2].

DL comprises a TBox $\mathcal{T}$ containing terminological class concept definitions and an ABox $\mathcal{A}$ comprising assertion definitions. The ABox assertions are called nodes or individuals x which are defined by their membership to a class concept C and by its relations R to other objects to form a graph. These two definitions are given in equation 1.

$$C(x) \qquad or \qquad R(x_1, x_2) \tag{1}$$

[1] http://clarkparsia.com/pellet
[2] http://owl.man.ac.uk/factplusplus
[3] http://www.racer-systems.com
[4] http://kaon2.semanticweb.org

Tableaux [8] is an unsatisfiability algorithm which attempts to prove an inference by refuting it. For instance, in order to prove an inference of the form $C(x)$, its negation is added the ABox $\mathcal{A}$ such that $\mathcal{A}_0 = \mathcal{A} \cup \neg C(x)$. The inference is proven if any attempt to extend the asserted negation into a complete interpretation will lead to a logical contradiction (a concept and its negation are present for the same individual). Alternatively, if a sound and non-contradictory interpretation is found, then this represents a counter example that disproves the conjectured inference.

Tableaux minimises the amount of space used by utilising a Tableaux expansion tree $\mathcal{T}$. This tree imposes an ordering on the application of expansion rules. Disjunctive concepts give rise to expansion and every possible expansion is searched in turn until a fully expanded and clash free tree is found, or all possibilities have been shown to lead to a contradiction. When $\mathcal{A}$ contains the disjunction $(C_1 \sqcup C_2)(x)$, disjunction transformation rule replaces $\mathcal{A}$ with $\mathcal{A}' = \mathcal{A} \cup \{C_1(x)\}, \mathcal{A}'' = \mathcal{A} \cup \{C_2(x)\}$. As such $(C_1 \sqcup C_2)(x)$ represents a branching point node n in the expansion tree $\mathcal{T}$ which contains new ABox states, and the concepts $C_1(x)$ and $C_2(x)$ each represent possible future branching point nodes n. In this paper, we distinguish between branching point nodes $n \in \mathcal{T}$ and individuals $x \in \mathcal{A}$, by referring to n as a node and x as an individual.

Tableaux also labels each concept in an individual and each role in an edge with a dependency set, indicating the branching points on which it depends. A concept $C \in \mathcal{L}(x)$ depends on branching point n if C was added to $\mathcal{L}(x)$ at the branching point n or if C depends on another concept D (or role R), and D (or R) depends on the branching point n. A role $R = \mathcal{L}(\langle x, y \rangle)$ depends on concept D when $\langle x, y \rangle$ was labelled R by the application of an expansion rule that used D. When a clash occurs, the reasoner state is restored to the most recent branch point where exploring another branch might alleviate the cause of the clash.

It is evident in [2] that DL Tableaux employs depth first expansion, which we illustrate in algorithm 1. Initially, this algorithm is started using the call $TabTreeTraverse(f, H, \mathcal{A}, \mathcal{T})$ where f is the top branch node in the expansion tree, H is the assertion $\neg C(x)$ for the inference check $C(x)$, $\mathcal{A}$ is the ABox, which is initialised with all the explicit assertions and relations in the semantic graph (knowledge base) and $\mathcal{T}$ is the expansion tree. The algorithm attempts to expand the tree and apply Tableaux transformation rules in order to generate a clash to prove all alternatives of the the tree, which are all of the conditions of the inference. If it returns *true* then the inference is proven. The algorithm utilises $ApplyTransRules(\mathcal{A})$ which applies the normal Tableaux DL transformation rules as defined in [2], on ABox $\mathcal{A}$ until a clash is detected (return *true*) or there are no more rules to apply (return $false$). The backjumping functionality is provided in algorithm 2. We note that we define a branch point node n identifier as $branchID(n)$ and we define a concept C or role R dependency set using $depBranchIDs(C)$. Let $currBranchNode(\mathcal{A})$ contain the currently active branch point node in the expansion tree for the ABox $\mathcal{A}$.

An example of an dependency directed, expansion tree is shown in figure 1. This figure represents the inference check $C(x)$ where C is a conjunction of the

Algorithm 1. TabTreeTraverse($n, W, \mathcal{A}, \mathcal{T}$)

Inputs: BranchNode n, Assertion W, ABox $\mathcal{A}$, ExpansionTree $\mathcal{T}$ where W is of the form $C(x)$
Outputs: Boolean $clashFound$
Let $clash$ be a clashing concept if one exists
$clash \leftarrow ApplyTransRules(\mathcal{A})$ \∗ runs the standard Tableaux rules ∗\
if $clash \neq null$ **then**
  return true
end if
Let $clashFound \leftarrow$ **false**
Let $branchID(n)$ denote the branch point node identifier for n
if $W = (D_1 \sqcup D_2 \sqcup ... \sqcup D_m)(x)$ \∗ has children ∗\ **then**
  for all $D_j \in W$ **do**
    Let c be a new BranchNode in the expansion tree $\mathcal{T}$ for $\mathcal{A}$, $\mathcal{T} \leftarrow \mathcal{T} \cup \{c\}$
    $branchID(c) \leftarrow branchID(c) + 1$
    $currBranchNode(\mathcal{A}) \leftarrow c$
    $\mathcal{L}(x) \leftarrow \mathcal{L}(x) \cup \{D_j(x)\}$ where x is an individual in $\mathcal{A}$
    $clashFound \leftarrow TabTreeTraverse(c, D_i(x), \mathcal{A}, \mathcal{T})$
    remove c from $\mathcal{T}$
    if $clashFound =$ **false then**
      return false\∗ failed to prove inference for disjunct c ∗\
    else
      $restoreTo(n)$ \∗ backjump to n ∗\
    end if
  end for
end if
return $clashFound$

Algorithm 2. RestoreTo(u, $\mathcal{A}$)

Inputs: int u and ABox where u is the branch node identifier to restore $\mathcal{A}$ to.
Outputs: ABox $\mathcal{A}$
for all $x_i \in \mathcal{A}$ where x_i is an individual \∗ all individuals in ABox $\mathcal{A}$ ∗\ **do**
  for all $A_j \in \mathcal{L}(x_i)$ where A_j is a class concept **do**
    Let t be the minimum value in $depBranchIDs(A_j)$ where $depBranchIDs(A_j)$ contains the branch dependency identifiers $branchID$ for class concept A_j
    if $t > u$ \∗ A_j added after u ∗\ **then**
      remove A_j from $\mathcal{L}(x_i)$
    end if
  end for
end for
return $\mathcal{A}$

form $C \equiv ((C_1 \sqcap C_2) \sqcap C_3)$. Also assume $\{C_1, C_2, C_3\} \subseteq \mathcal{L}(x)$ where $x \subseteq \mathcal{A}$. Since Tableaux proves inference by refutation C is transformed into a disjunction of the form $\neg C \equiv ((\neg C_1 \sqcup \neg C_2) \sqcup \neg C_3)$. In figure 1a, a clash is detected caused by C_1 which is part of the disjunction $(\neg C_1 \sqcup \neg C_2)$, added to $\mathcal{L}(x)$ at branch

node 2. Therefore, the Tableaux is restored to its earlier state, by a backjump to branch 2. Then in 1b, the second element $\neg C_2$ of $(\neg C_1 \sqcup \neg C_2)$ is applied, giving rise to another clash. The reasoner backjumps to expansion tree node 1 (1c), after which the second disjunct $\neg C_3$ of $((\neg C_1 \sqcup \neg C_2) \sqcup \neg C_3)$ is applied giving rise to a third clash. All branch nodes clashed, so the inference $C(x)$ is proven. Note that grey nodes indicate those which have already been evaluated in a previous iteration. In this example, only one individual x was used for briefness, however in practise the expansion tree may apply disjuncts to several different individuals.

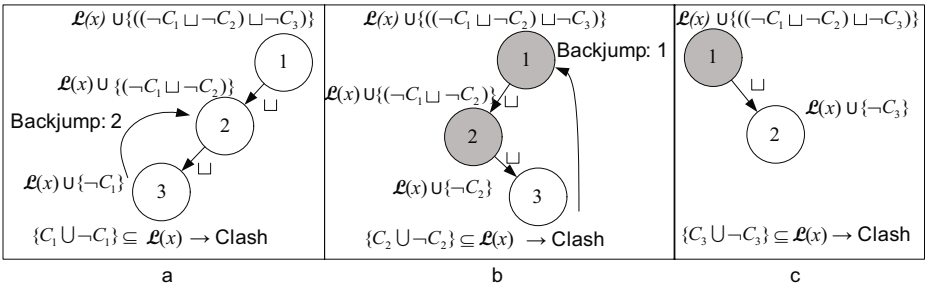

Fig. 1. Standard Tableaux Expansion Search Tree

Current Tableaux has the following shortcomings:

1. Only a true/false answer is provided. A weighted degree of match and a level of confidence in this match based on how much of the inference was checked, should be provided;
2. The order of condition and subcondition evaluation is depth-first. These should be applied in weighted order of importance to the user, so that the highest possible degree of match can be found in the time available;
3. Tableaux returns $false$ as soon as any condition or subcondition in the inference check fails. Since pervasive environments inherently contain dynamic heterogeneous data the reasoner should continue to check subsequent inference conditions, even if some fail (time/resource permitting).

Our approach, which addresses these shortcomings, is introduced in the next section.

4 Weighted Adaptive Reasoning

We propose an approach to Tableaux inference proof, which supports a reduction in result accuracy if greater efficiency is required. Our approach associates a level of importance with each branch leaf and evaluates these branches first. The reasoner will continue even if some conditions fail, and may be stopped at any point in the reasoning process and a degree of match result is provided to the user.

4.1 Weighted Expansion Tree Traversal

Certain inference conditions may have a different level of importance to the user. Therefore, we associate weights with each condition and sub-condition. These weights may be entered explicitly by the user at the time that the user specifies the request as a rating from 1 to 10, high/medium/low priority, or mandatory/non-mandatory. These values are converted to relative weights (see below). Alternatively, weights may be gathered implicitly using historical user preference data. For instance, if for half his or her requests, the user has invoked services which are close in proximity, this characteristic may be inserted to future requests with a weight of 0.5. In summary, weights may be assigned by different means, however, the assignment process is not the principle focus on this paper, therefore, we do not discuss this in further detail.

Let rw denote a relative weight value for each a condition or sub-condition derived from each conjunct or subconjunct in the class concept C, for the inference check $C(x)$. Since Tableaux proves the inference using negation, C is converted into a disjunction where $\neg C \equiv (\neg D_1 \sqcup (\neg D_2 \sqcup \neg D_3) \sqcup ... \neg D_m), 1 \leq j \leq m$. Each disjunct and sub-disjunct are organised as pairs containing the disjunct $\neg D_j$ and relative weight $rw(D_j)$ pairs and stored in descending $rw(D_i)$ order in a queue, such that $\mathcal{Q} = \{\langle \neg D_1, rw(\neg D_1)\rangle, \langle \neg D_2, rw(\neg D_2)\rangle, \langle \neg D_3, rw(\neg D_3)\rangle, ..., \langle \neg D_m, rw(\neg D_m)\rangle\}$, as shown in algorithm 3 where $\neg C$ is passed as input from the inference $C(x)$.

Weights are said to be relative, because the sum of the $rw(D_j)$ values for all children nodes much equal the $rw(p_e)$ of the parent node. Let p_e denote a node, and let $\mathcal{C}$ denote a set of child nodes for p_e such that $\mathcal{C} = \{c_1, c_2, ..., c_t \mid (1 \leq s \leq t)\}$. rw values must be defined such that $rw(p_e) = \sum_{c_s \in \mathcal{C}} rw(c_s)$ where $\langle p_e, rw(p_e)\rangle$. Relative weights ensure that a disjunct D_1 in $P = (D_1 \sqcup D_2)$, cannot be applied before its parent disjunction P is itself applied.

Algorithm 3. CreateQueue(A)

```
1: Inputs: ClassConcept A
2: Outputs: Queue Q
3: if A = (D_1 ⊔ ... ⊔ D_m) then
4:    for all D_j ∈ D do
5:       Q ← Q ∪ {⟨D_j, rw(D_j)⟩}
6:    end for
7: end if
```

Algorithm 4 presents a general illustration of our adaptive reasoning search tree expansion process, which replaces depth first with condition relative weight order. This algorithm is invoked by the call $AdapTreeTraverse(\mathcal{Q}, x, \mathcal{A}, \mathcal{T})$ where $\mathcal{Q}$ is the pre-initialised queue as specified above, of disjuncts and sub-disjuncts which have been expressed in $\neg C$ for the inference check $C(x)$, and their relative weights. x is the individual in the inference check $C(x)$. $\mathcal{A}$ is the ABox, which contains the explicit assertions and relations in the semantic graph (knowledge

base) and $\mathcal{T}$ is the expansion tree. Unlike Algorithm 1, AdapTreeTraverse (algorithm 4) is not a recursive algorithm, rather it iterates through all the concepts in the queue $\mathcal{Q}$, as specified above, in descending relative weight order. It attempts to expand the tree and apply Tableaux transformation rules to generate clashes to prove each condition. The algorithm continues until all conditions are checked, or the algorithm is stopped prematurely when $KeepMatching()$ results a $false$.

Algorithm 4 uses the functions defined as follows. Let $currBranchNode(\mathcal{A})$ be a holder containing the currently active branch point node in the expansion tree for ABox $\mathcal{A}$. Let $ApplyTransRules(\mathcal{A})$ denote a function which applies the normal Tableaux DL transformation expansion rules on ABox $\mathcal{A}$ until a clash is detected (return $true$) or there are no more rules to apply (return $false$). Functions $GetDegMatch()$ and $GetConf()$ return the degree of match and the confidence values (see section 4.3), which also make use of the $clashingConcept(W)$ and $checkedConcept(W)$ return value.

The function, $KeepMatching()$ is used to determine whether the reasoner should stop execution prematurely, based on constraints. There may be several constraint parameters P, such that $P = \{p_1, p_2, ..., p_n \mid (1 \leq i \leq n)\}$ and threshold values T for parameters, such that $T = \{t_1, t_2, ..., t_n \mid (1 \leq i \leq n)\}$. Each p_i corresponds to a threshold value t_i. A p_i represents a ratio $(0 \leq p_i \leq 1)$, such as memory used over total memory, current confidence over total total possible confidence, time elapsed over total time available (user specified), and battery life remaining over total battery life. Each t_i is specified by application defaults or explicit user requirements, such that $0 \leq t_i \leq 1$. $KeepMatching()$ returns $false$ iff any $t_i \geq p_i$. In the current implementation of the algorithm (see section 5), we utilise time as our only resources parameter, such that $p_1 = timeElapsed/totalTime$, where $totalAvailTime$ is user specified, eg "I want a result in 10 seconds".

Figure 2 illustrates an example utilising our adaptive reasoning approach, for the inference check $C(x)$ where $\neg C \equiv ((\neg C_1 \sqcup \neg C_2) \sqcup \neg C_3)$ is given the relative weights, such that $\{\langle \neg C, 1.0\rangle, \langle (\neg C_1 \sqcup \neg C_2), 0.7\rangle, \langle \neg C_1, 0.5\rangle, \langle \neg C_3, 0.3\rangle, \langle \neg C_1, 0.2\rangle\}$. Assume also that $\{C_1, C_2, C_3\} \subseteq \mathcal{L}(x)$ where $x \in \mathcal{A}$. The nodes are executed in relative weight rw order (weights and node identifiers are displayed on the nodes in figure 2). Notice that in contrast to depth-first Tableaux (see figure 1), concept $\neg C_3$ is applied in step b, before the concept $\neg C_2$ from $(\neg C_1 \sqcup \neg C_2)$ is applied in step c. This is because $\neg C_3$ had a relative weight of 0.3, which was higher than 0.2 for $\neg C_2$. The grey nodes denote those nodes which have already been expanded in a previous step and do not need to be re-applied. In this example, only one individual x was used for briefness, however in practise the expansion tree may apply disjuncts to several different individuals.

4.2 Branch Identifiers and State Management

The, relative weight values control execution, as opposed to the traditional depth or breath-first ordering. Therefore, it is likely that there may be multiple unfinished expansion branches. As shown in section 3, in depth-first reasoning,

Algorithm 4. AdapTreeTraverse($\mathcal{Q}, x, \mathcal{A}, \mathcal{T}$)

Inputs: Queue $\mathcal{Q}$, Individual x, ABox $\mathcal{A}$, ExpansionTree $\mathcal{T}$.
Outputs: $\langle$Decimal $degMatch$, Decimal $conf\rangle$
* $\mathcal{Q}$ is in descending order by $rw(n_j)$ where $\mathcal{Q} = \{\langle D_1, rw(D_1)\rangle, \langle D_2, rw(D_2)\rangle, ..., \langle D_m, rw(D_m)\rangle \mid (1 \leq j \leq m)\}$ and D is a class concept *\
for all $D_j \in \mathcal{Q}$ **do**
  if $KeepMatching() =$ **false then**
    return $\langle GetDegMatch(\mathcal{Q}), GetConf(\mathcal{Q})\rangle$
  end if
  Let h be a new BranchNode in the expansion tree $\mathcal{T}$ for $\mathcal{A}$ such that $\mathcal{T} \leftarrow \mathcal{T} \cup \{h\}$
  $currBranchNode(\mathcal{A}) \leftarrow h$
  $\mathcal{L}(x) \leftarrow \mathcal{L}(x) \cup \{D_j\}$ * where x is an individual in $\mathcal{A}$ *\
  $checkedConcept(D_j) \leftarrow$ **true*** for confidence *\
  Let $clash$ be a clashing concept, if one exists
  $clash \leftarrow ApplyTransRules(\mathcal{A})$ * runs the standard Tableaux rules *\
  if $clash \neq null$ **then**
    $clashingConcept(clash) \leftarrow$ **true*** for degree of match & confidence *\
  end if
end for
return $\langle GetDegMatch(\mathcal{Q}), GetConf(\mathcal{Q})\rangle$

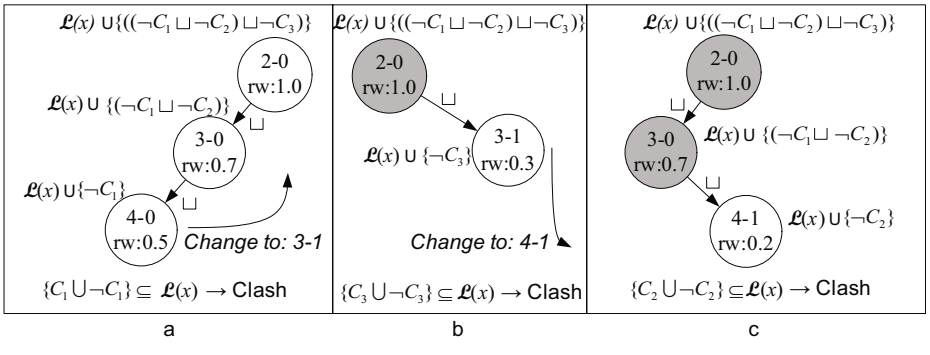

Fig. 2. Tableaux Adaptive Expansion Search Tree

a branch is continually expanded using transformation rules such that $\mathcal{A}' = \mathcal{A} \cup \{...\}$. This depth first expansion occurs until either a clash is found or there are no more branches to apply. Where a clash is found the ABox $\mathcal{A}_n$ is restored to an earlier branch node n_i state $\mathcal{A}_i$, and the current state $\mathcal{A}_n$ is discarded. This means that all ABox assertions added after $branchID(n_i)$ are removed. However, under our adaptive reasoning strategy, the expansion tree is expanded in branch node weight order, rather than depth first. This implies that there may be several unfinished open branches at one time, for instance

in figure 2, branch point 3-0 from step a, is preserved for step c, while a new branch is opened in step b. This was because concept C_3 was considered more important than C_2.

Support for this functionality requires modifications to:

- Branch point node identifiers *branchID*
- State of node labels $\mathcal{L}(x)$ and $\mathcal{L}(\langle x, y\rangle)$ where x and y are individuals

In terms of branch point node identifiers, a $branchID(n)$ must be unique. However, since a traditional $branchID(n)$ is only a depth count, it is no longer unique if multiple branches are present in the tree $\mathcal{T}$. Therefore, we specify new branch identifiers $adapBranchID(n)$, of the form given in expression 2, to take breath into account. Let $adapBranchID(n)$ denote the branch identifier for a node n in a tree formation where $depth(n)$ denotes the distance from the top most object in the tree down to node n and $breath(n)$ is a count which increases for every node in the tree which has the same $depth(n)$, such that $\{2 \leq depth(n)$, $0 \leq breath(n)$. Note that *adapBranchID* 0-0 and 1-0 are reserved for explicit assertions and pre-processing. Let $adapDepBranchIDs(n)$ contain a dependency list of $adapBranchID(n)$ for the node n. $adapDepBranchIDs(n)$ contains the branch identifier for node n and for all of its parents in the expansion tree $\mathcal{T}$, such that $adapDepBranchIDs(n) = adapBranchID(n) \cup \{ \bigcup_{p \in P} adapBranchID(p) \}$, where P contains all parent nodes of n in $\mathcal{T}$.

$$adapBranchID(n) = depth(n) \ominus breath(n) \tag{2}$$

In terms of type label state management standard Tableaux does not support multiple unfinished open branches. The state of type labels $\mathcal{L}(x)$ and $\mathcal{L}(\langle x, y\rangle)$ where x and y are individuals in ABox $\mathcal{A}$ is determined by parent branch point nodes. Class concepts and role assertions are progressively added to type labels by tree branch expansions and by other transformation rules as they occur, and removed when a backjump (restore) occurs. However, in order to support multiple open branches, previously applied branch nodes much be retained. Type labels support multiple states, and their contents (state) at any one time must reflect the current branch point node without requiring re-application of all parent node expansions and transformations, when moving back to a previously partially expanded branch. Therefore, we redefine type labels, such that concepts C and role assertions $R(x, y)$ added to type labels $\mathcal{L}(x)$ and $\mathcal{L}(\langle x, y\rangle)$, respectively, are indexed by branch identifier.

In this section we refer to both $\mathcal{L}(x)$ and $\mathcal{L}(\langle x, y\rangle)$, simply as $\mathcal{L}$, such that $\mathcal{L} = \{\langle adapBranchID(\alpha_1), \alpha_1\rangle, ..., \langle adapBranchID(\alpha_p), \alpha_p\rangle \mid (1 \leq k \leq p)\}$. Let $V(w)$ be a subset of the elements α_k (concepts and relations) in $\mathcal{L}$, where w is the currently active branch node in the expansion tree (see algorithm 4) and where w has a set of branch node dependencies and all the elements in $V(w)$ must have a branch identifier which is in this set of dependencies. This is defined in algorithm 5, such that $V(w) = \{\alpha_1, ..., \alpha_v \mid (1 \leq u \leq v)\}$ where $adapBranchID(\alpha_u) \in adapDepBranchIDs(w)$. Therefore, $V(w)$ can be considered as the state of $\mathcal{L}$ when the currently active expansion node is w, to support

swapping between simultaneous unfinished branches. For example, if $\mathcal{L} = \{\langle$2-1, $\alpha_1\rangle, \langle$2-3, $\alpha_2\rangle$, $\langle$3-1, $\alpha_3\rangle$, $\langle$4-0, $\alpha_4\rangle\}$ and $adapDepBranchIDs(w) = \{$2-1, 3-1$\}$ where $w = currBranchNode(\mathcal{A})$, then $GetTypeLabel$ returns $\{\alpha_1, \alpha_3\}$

Algorithm 5. GetTypeLabel($\mathcal{L}$, ABox $\mathcal{A}$)

```
1: Inputs: TypeLabel L, ABox A
2: Outputs: Set V
3: Let w = currBranchNode(A) \* currently active branch point node for A *\
4: for all β_i ∈ L where β_i = ⟨adapBranchID(α_i), α_i⟩ do
5:    if adapBranchID(α_i) ∈ adapDepBranchIDs(w) then
6:       V ← V ∪ {α_i}
7:    end if
8: end for
9: return V
```

4.3 Degree of Match and Match Confidence

In our adaptive reasoning strategy we provide a degree of inference match and a level of confidence in this result. Degree of match is the known, weighted degree to which a particular inference $C(x)$, holds. It is a normalised value based on those conditions actually checked in the processing time available, where 1 denotes a complete match and 0 denotes no match. A degree of match is the sum of all relative weight rw values for all the concepts C where a clash was detected. Let Q denote a set containing all conditions and sub-conditions to be checked, as defined in defined in section 4.1, $\mathcal{Q} = \{\langle D_1, rw(D_1)\rangle, \langle D_2, rw(D_2)\rangle,$ ..., $\langle D_p, rw(D_p)\rangle \mid (1 \leq k \leq p)\}$ as defined in section 4.1 where $rw(D_k)$ is the relative weights for each condition specified in $\neg C$ for the inference check $C(x)$. Let $getDegMatch(\mathcal{T})$, given in formula 3, return the known degree of match for a particular inference. Let $clashingConcept(W)$ contain $true$ if the concept W caused a clash, set by algorithm 4.

$$GetDegMatch(\mathcal{Q}) = \sum_{e_j \in \mathcal{Q}} rw(D_k) \text{ iff } clashingConcept(D_k) = true \quad \text{where } e_j = \langle D_k, rw(D_k)\rangle \tag{3}$$

Confidence is the normalised ratio providing a measure indicating the proportion of the total reasoning process, which has been completed. Confidence is the ratio of worst-case weighted number of conditions to execute over the number actually executed. Let $GetConf(\mathcal{Q})$ denote the confidence value, given in formula 4. Let $checkedConcept(W)$ return $true$ if the class concept W was actually applied by the algorithm 4.

$$GetConf(\mathcal{Q}) = \frac{\sum_{e_j \in \mathcal{Q}} rw(D_k) \text{ iff } checkedConcept(D_k) = true}{\sum_{e_j \in \mathcal{Q}} rw(D_k)} \quad \text{where } e_j = \langle D_k, rw(D_k)\rangle \tag{4}$$

In this section we provided our adaptive reasoning approach. Both our adaptive algorithm (see algorithm 4) and standard Tableaux (see algorithm 1) have a worst case complexity of $O(n)$ where n is the total number of branch point nodes which can be expanded in the tree $\mathcal{T}$. However, if the level of confidence required is reduced, the adaptive reasoning strategy has a best case complexity of $O(1)$ while Tableaux has a best case complexity of $O(m)$ where m is the average number of nodes in a branch in the expansion tree $\mathcal{T}$.

In the next section we discuss the implementation of our adaptive reasoning approach and provide an evaluation to illustrate the effect on performance and result accuracy.

5 Implementation and Evaluation

In this section we discuss the implementation of our adaptive reasoning strategy provide a performance evaluation using a case study.

5.1 Case Study

To demonstrate our adaptive reasoning we approach we developed a case study in which Bob is searching for particular attributes in a foreign city centre, such as a movie cinema, printing service, Internet cafe etc. The case study comprises several request queries as follows (relative weight in brackets):

1. RetailOutlet (0.33), MovieCinema (0.3), Internet (0.23), Cafe (0.14)
2. RetailOutlet (0.40), MovieCinema (0.32), Internet (0.2), Cafe (0.04), Colour Fax WiFi Printer (0.04)
3. Internet (0.5), Colour Fax WiFi Printer (0.3), Cafe (0.2)

Our scenario also has 8 different service advertisements which we label A, B, C, D, E, F, G, H, where A matches all requests, B-G only partially match each request, while H does not match any attribute in any request. The ontologies containing these service offerings comprise 204 classes and 241 individuals.

5.2 Implementation

We have implemented our adaptive strategies as an extension to the Pellet[5] 1.5 reasoner which supports OWL-DL with $\mathcal{SHOIN}$ expressiveness [2]. Pellet is open source, allowing us to provide a proof of concept. 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++[6] is written in C++. All our evaluations were conducted 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)[7], allocated 15MB of memory. Note, that in the next section, results for both standard reasoning and adaptive reasoning utilise optimisations from previous work which enable mobile reasoning [17].

[5] http://clarkparsia.com/pellet
[6] http://owl.man.ac.uk/factplusplus
[7] http://www2s.biglobe.ne.jp/ dat/java/project/jvm/index_en.html

5.3 Evaluation

Figure 3 shows the degree of match obtained by comparing request 1 with service advertisements A-H (see section 5.1), for standard reasoning and adaptive reasoning. Both standard and adaptive reasoning stop on failure (stop as soon as any request condition fails to match). Standard reasoning provides a meaningless $false$ result when stopped prematurely, while our adaptive reasoning strategy provides a degree of match result for those request conditions, which were successfully matched in the available time. This demonstrates a far better reporting of inference results to the user. The "Check All" results, figure 3, demonstrate our support for heterogeneous data (continue matching when some conditions fail). Figure 4 demonstrates that our adaptive reasoning strategy matches the most important conditions first, for the services A-H, as well as supporting premature stopping and a degree of match result to the user. For most service advertisements, most of the degree of match result is obtained in the first 10 seconds of processing.

Figure 5 compares request 2 and 3 against service advertisement A, showing similar results. For instance in 5(left), a degree of match of almost 80% was reached in the first 10s of operation, while the remaining 20% required 40 seconds to establish. Figure 5(right) shows that when condition weights are similar, the correspondence between degree of match and processing time is more uniform. When standard reasoning was used, no result was provided until the reasoning was completed in full, requiring 40 seconds. Figure 6 further highlights the way in which our adaptive reasoning strategy matches the most important attributes first to provide a higher degree or match more quickly, in order to make the best use of processing time available. This figure compares adaptive reasoning against the result which would be obtained if standard Tableaux reasoners provided a degree of match. Where request 2 is matched with service A (left), after 10 seconds the adaptive reasoner found a degree of match of almost 80% while the standard reasoner had successfully matched conditions with weights that added up to under 20%. This is because standard reasoners match attributes in depth-first, arbitrary order. Therefore, it is often the case that attributes which contribute greatly to the degree of match, are checked late in the process. Also note while a cumulative result is shown for standard reasoners for illustrative purposes only, standard reasoners cannot be stopped early.

The strategies presented in this paper are shown to effectively meet the following challenges:

1. match the most important request conditions first
2. handle heterogeneous inaccurate data by continuing to reason after a term failed (if sufficient time).
3. handle time/resource pressures by supporting early stopping of the reasoner and providing a current degree of inference to the user.

In this section we have clearly demonstrated that our adaptive reasoning approach provides a significant advantage over current reasoning strategies for mobile users. Our approach effectively provides flexible reasoning to meet the resource and time challenges which characterise mobile environments.

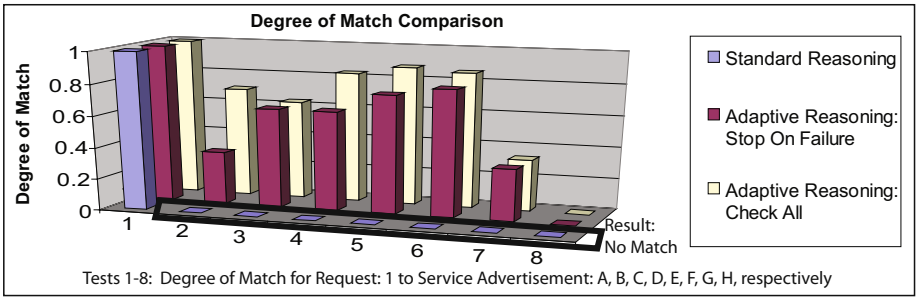

Fig. 3. Comparison of degree of match for standard (stop after any request condition fails) and adaptive reasoning (check all conditions), where Request 1 is compared against 8 different potential services A-H see section 5.1

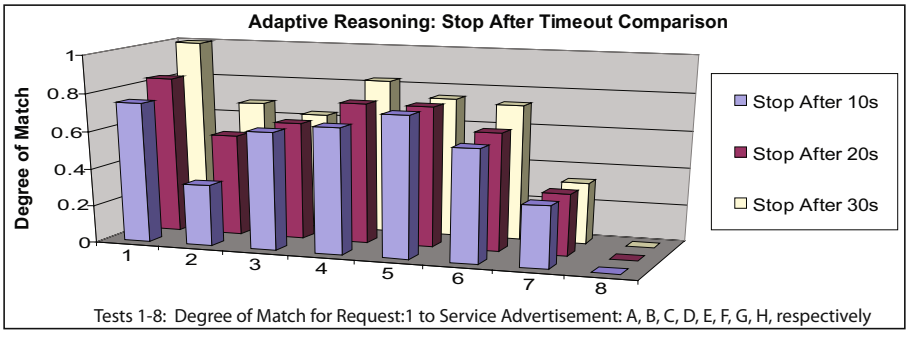

Fig. 4. Comparison of degree of match for adaptive reasoning after a timeout of 10, 20 and 30 seconds, where Request 1 is compared against 8 different potential services: A-H see section 5.1

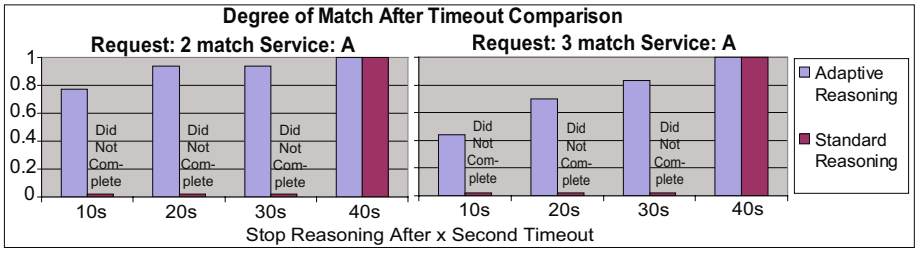

Fig. 5. Comparison of degree of match for adaptive reasoning after a timeout of 10, 20, 30 and 40 seconds, for requests 2 (left) and 3 (right), matched against service advertisement A

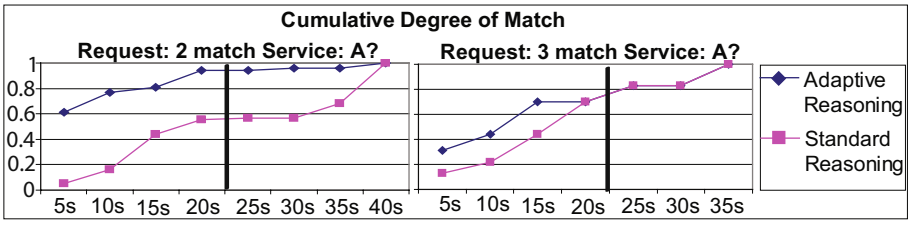

Fig. 6. Illustration of the cumulative degree of match for adaptive reasoning and standard reasoning, where requests 2 (left) and 3 (right) are matched against service advertisement A

6 Conclusion

We have presented a novel adaptive reasoning strategy which supports partial matching, premature stopping and provides a degree of inference match, based on those request conditions already checked. Our evaluations demonstrate that our strategies effectively meet these goals, by matching the most important request conditions first, to achieve the highest possible degree of match result within the time available. This is a significantly more effective approach than current approximate reasoning techniques which match request conditions arbitrarily and completely before providing any results.

References

1. Arnold, K., O'Sullivan, B., Scheifler, R.W., Waldo, J., Woolrath, A.: The Jini Specification. Addison-Wesley, Reading (1999)
2. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F.: The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, Cambridge (2003)
3. Donini, F.M., Lenzerini, M., Nardi, D., Schaerf, A.: Reasoning in description logics (1997)
4. Fensel, D., van Harmelen, F.: Unifying reasoning and search to web scale. Internet Computing, IEEE 11(2), 96 (2007)
5. Gligorov, R., ten Kate, W., Aleksovski, Z., van Harmelen, F.: Using google distance to weight approximate ontology matches. In: 16th international Conference on World Wide Web. ACM, New York (2007)
6. Gu, T., Kwok, Z., Koh, K.K., Pung, H.K.: A mobile framework supporting ontology processing and reasoning. In: 2nd Workshop on Requirements and Solutions for Pervasive Software Infrastructure (RSPS) in conjunction with the 9th International Conference on Ubiquitous Computing (Ubicomp 2007), Austria (2007)
7. Hitzler, P., Vrandecic, D.: Resolution-Based Approximate Reasoning for OWL DL. In: Semantic Web - ISWC. Springer, Heidelberg (2005)
8. Horrocks, I., Sattler, U.: A tableaux decision procedure for shoiq. In: 19th International Conference on Artificial Intelligence, IJCAI 2005 (2005)

9. Kargin, B., Basoglu, N.: Factors affecting the adoption of mobile services. In: Portland International Center for Management of Engineering and Technology, pp. 2993–3001. IEEE, Los Alamitos (2007)
10. Kleemann, T.: Towards mobile reasoning. In: International Workshop on Description Logics (DL 2006), Windermere, Lake District, UK (2006)
11. Kuster, U., Konig-Ries, B., Klein, M.: Discovery and mediation using diane service descriptions. In: Second Semantic Web Service Challenge 2006 Workshop, Budva, Montenegro (2006)
12. Mokhtar, S.B., Preuveneers, D., Georgantas, N., Issarny, V.: Easy: Efficient semantic service discovery in pervasive computing environments with qos and context support. Journal Of System and Software 81(5) (2008)
13. Ranganathan, A., Campbell, R.H.: A middleware for context-aware agents in ubiquitous computing environments. In: ACM/IFIP/USENIX International Middleware Conference, Rio de Janeiro, Brazil, p. 143–161 (2003)
14. Roto, V., Oulasvirta, A.: Need for non-visual feedback with long response times in mobile hci. In: International World Wide Web Conference Committee (IW3C2), Chiba, Japan (2005)
15. Srinivasan, N., Paolucci, M., Sycara, K.: Semantic web service discovery in the owl-s ide. In: 39th International Conference on System Sciences, IEEE, Hawaii (2005)
16. Srirama, S.N., Jarke, M., Parinz, W.: Mobile web service provisioning. In: Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services, AICT/ICIW (2006)
17. Steller, L., Krishnaswamy, S., Gaber, M.M.: Enabling scalable semantic reasoning for mobile services. International Journal of Semantic Web Information Systems - Special Issue on Scalability and Performance of Semantic Systems (2009)
18. Stuckenschmidt, H., Kolb, M.: Partial matchmaking for complex product and service descriptions. Multikonferenz Wirtschaftsinformatik (2008)
19. Tergujeff, R., Haajanen, J., Leppanen, J., Toivonen, S.: Mobile soa: Service orientation on lightweight mobile devices. In: International Conference on Web Services (ICWS), pp. 1224–1225. IEEE, Salt Lake City (2007)
20. Veijalainen, J.: Mobile ontologies: Concept, development, usage, and business potential. International Journal on Semantic Web & Information Systems 4(1), 20–34 (2008)
21. Wache, H., Groot, P., Stuckenschmidt, H.: Scalable instance retrieval for the semantic web by approximation. In: WISE 2005. Springer, Heidelberg (2005)
22. Wang, Z., Hu, Y.: An approach for semantic web service discovery based on p2p network. In: 4th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM 2008), pp. 1–4. IEEE, Los Alamitos (2008)
23. Weiser, M.: The computer of the 21st century. Scentific American 3(265), 66–75 (1991)
24. Xiaosu, C., Jian, L.: Build mobile services on service oriented structure. In: International Conference on Wireless Communications, Networking and Mobile Computing, vol. 2, pp. 1472–1476. IEEE, Los Alamitos (2005)
25. Zacharias, V., Abecker, A., Vrandecic, D., Borgi, I., Braun, S., Schmidt, A.: Mind the web. In: ASWC 2007 and ISWC 2007, vol. 291. IEEE, Los Alamitos (2007)