A WEIGHTED APPROACH FOR OPTIMISED REASONING FOR PERVASIVE SERVICE DISCOVERY USING SEMANTICS AND CONTEXT

Luke Steller, Shonali Krishnaswamy, Simon Cuce, Jan Newmarch
Faculty of Information Technology, Monash University, 900 Dandenong Rd, Melbourne, Australia

Seng Loke
Computer Science & Computer Engineering, La Trobe University, Bundoora, Melbourne, Australia

Keywords: Scalable semantic reasoning, context-aware discovery, service-oriented pervasive discovery architecture.

Abstract: There is an increased imperative for pervasive service discovery architectures, to aid mobile users for time critical tasks in service rich ad-hoc environments. As such, there is a need for an innovative technology for semantically-driven pervasive service discovery by enabling semantic reasoning engines to function in an effective and efficient manner on resource-constrained mobile devices and by incorporating context-awareness to provide relevant services to the user. We outline several optimisation and branch ranking strategies for pervasive reasoning to meet this goal and provide a performance evaluation of our approach.

1 INTRODUCTION

Studies such as (Roto & Oulasvirta, 2005) have established that mobile users typically have a tolerance threshold of about 5 to 15 seconds in terms of response time, before their attention shifts elsewhere, depending on their environment. Thus, service discovery architectures that operate in mobile environments must cope with the very significant challenges of not merely finding relevant services, but being able to do so rapidly in a highly dynamic and varying context.

The limitations of syntactic, string-based matching for web service discovery coupled with the emergence of the semantic web implies that next generation web services will be matched based on semantically equivalent meaning, even when they are described differently (Broens, 2004), using OWL-DL which is based on Description Logic (DL) (Baader, Calvanese, McGuinness, Nardi & Patel-Schneider, 2003). While current service discovery architectures such as Jini (Arnold, O'Sullivan, Scheifler, Waldo & Woolrath, 1999) and UPnP (UPnP, 2007) use either interface or string based syntactic matching, there is a growing emergence of OWL-S semantic matchmakers such as CMU Matchmaker (Srinivasan, Paolucci & Sycara, 2005), LARKS (Sycara, Widoff, Klusch & Lu, 2002), IRS-III (Cabral, Domingue, Galizia, Gugliotta, Tanasescu et al., 2006) and DIANE (Küster, König-Ries & Klein, 2006) which support varying levels of semantic reasoning and approximate matching. However, they all operate on the basis of a centralised high-end node to perform reasoning. There are in addition, architectures developed specifically for the pervasive service discovery domain which are driven by context, such as MobiShare (Doulkeridis, Loutas & Vazirgiannis, 2005), COSS (Broens, 2004) and CASE (Sycara et al., 2002) which are either syntactic or require the presence of a centralised high-end node.

This reliance on a high-end, centralised node for performing semantically driven pervasive service discovery can clearly be attributed to the fact that semantic reasoners used by these architectures (such as FaCT++ (FaCT++, 2007), RacerPro (RacerPro, 2007) and KAON2 (KAON2, 2007)) are all resource intensive. Therefore, they are unsuitable for deployment on small resource constrained devices, such as PDAs and mobile phones. These small devices which are typical in the context of mobile service discovery are quickly overwhelmed when the search space in terms of ontology size and reasoning complexity increases. Alternatively, KRHyper (Kleemann, 2006) is a First Order Logic (FOL) reasoner which implements FOL counterparts of standard DL optimisations (Horrocks & Patel-Schneider, 1999) and functions on a small device. However, it too, suffers from out of memory exceptions when the reasoning task is too large and no response is given. Clearly, this shows that existing reasoning approaches cannot be directly ported to a mobile device in their current form.

The reality of mobile environments is a world characterised by ad-hoc an intermittent connectivity where such reliance on remote/centralised processing (and continuous interaction) may not always be possible or desirable given the need for rapid processing and dynamically changing context. Pervasive service discovery has to necessarily be under-pinned by the current context to meet the all-important criteria of relevance in constantly changing situations. The communication overhead (not to mention the infeasibility/impracticability) of constantly relaying contextual and situational changes of the user/device to a central server will lead to inevitable delays. Furthermore, reasoning on a central server about sensitive historical user profiling data, used to select the best service for the user, raises privacy concerns (Kleemann, 2006).

Thus there is a clear imperative that for semantically driven pervasive service discovery to meet the very real response-time challenges of a mobile environment, the capacity to perform matching and reasoning must occur on the resource limited device itself. Therefore, there is a need for a pervasive service discovery architecture, which more flexibly manages the trade-off between computation time and precision of results, depending on the available resources on the device.

The remainder of the paper is structured as follows. In section 2 we outline our strategies for optimised reasoning and formally describe these in section 3. Section 4 provides an implementation and performance evaluation and conclude in section 5.

2 APPROACH – REASONING FOR PERVASIVE SERVICE DISCOVERY

In this section we discuss current Tableaux semantic reasoners and present our optimisations and ranking algorithms, for the Tableaux algorithm.

2.1 Semantic Reasoners

The effective employment of semantic languages such as OWL requires the use of semantic reasoners such as Pellet (Pellet, 2007), FaCT++ (FaCT++, 2007), RacerPro (RacerPro, 2007) and KAON2 (KAON2, 2007). Most of these reasoners utilise the Tableaux (Horrocks & Sattler, 2005) algorithm with standard DL optimisations (Horrocks et al., 1999), due to its efficiency. Tableaux reasoners reduce all reasoning tasks to a consistency check, in which disjunctions form combinations of branches in the reasoner. If all branches contain a clash, where a fact and its negation are both asserted, then clash is proven for all models of the knowledge base. Tableaux can be used to check whether an individual I representing a service, matches a request RQ (ie. $I \in RQ$), by negating RQ. If the clash is proven for all models then $I \in RQ$ membership, is proven. In the next section we provide a case study to motivate the need for reasoning in pervasive discovery.

2.2 Case Study – Searching for a Printer

Bob wishes to print a document from his PDA and issues a service request to find a black and white, laser printer which supports a wireless network protocol such as Bluetooth, WiFi or IrDA. This is an extension of (Steller, Krishnaswamy & Newmarch, 2006). Equations 2-3 show Bob's request in DL form, while equation 4 presents a matching printer.

$$\begin{gathered}\text{Request} \equiv \text{WNet} \sqcap \exists \text{hasColour}.\{\text{Black}\} \\ \sqcap \text{LaserPrinterOperational}\end{gathered} \tag{1}$$

$$\begin{gathered}\text{WNet} \equiv \exists \text{hasComm}.\{\text{BT}\} \sqcup \\ \exists \text{hasComm}.\{\text{WiFi}\} \sqcup \\ \exists \text{hasComm}.\{\text{IrDA}\}\end{gathered} \tag{2}$$

$$\begin{gathered}\text{LaserPrinterOperational} \equiv \text{Printer} \sqcap \\ \exists \text{hasCartridge}.\{\text{Toner}\} \sqcap \\ \geq 1\ \text{hasOperationContext}\end{gathered} \tag{3}$$

$$\begin{gathered}\text{Printer(LaserPrinter1)}, \\ \text{hasColour (LaserPrinter1, Black)}, \\ \text{hasCartridge(LaserPrinter1, Toner)}, \\ \text{hasComm(LaserPrinter1, BT)}, \\ \text{hasOperationContext(LaserPrinter1, Ready)}\end{gathered} \tag{4}$$

Equation 1 defines three attributes in the request, the first is unfolded into equation 2, requiring support for either Bluetooth, WiFi or IrDA, the second attribute specifies a black and white requirement and the third is unfolded into equation 3, specifying a printer which has a toner cartridge and at least one operational context. Equation 4 fragment defines the LaserPrinter1 individual as meeting the service request. A DL Tableaux excerpt proving the truth of LaserPrinter1 ∈ Request, shows a clash for all branches, as follows (only two request attributes included for briefity):

```
Add: ¬Request to Individual:
      LaserPrinter1
   ¬Request ≡ ¬WNet ∪
     ¬∃hasColour.{Black}
   Apply Disjunction Element:
     ¬WNet ≡ ∀hasComm.{¬BT} ∩
         ∀hasComm.{¬WiFi} ∩
         ∀hasComm.{¬IrDA}
    Add: ¬BT to Nominal: BT, CLASH
   Apply Disjunction Element:
     ∀hasColour.¬{Black}
     Add: ¬{Black} to Nominal:
        Black, CLASH
```

2.3 Optimisation Strategies

We observed that DL Tableaux reasoners leave scope for further optimisation to enable reasoning on small/resource constrained devices with a significant improvement to response time and avoiding situations such as "Out of Memory" errors encountered in (Kleemann, 2006). Our algorithm involves a range of optimisation strategies such as: 1. associating weight values with individuals and disjunctions, 2. selective application of consistency rules, 3. ranking disjunctions, 4. ranking individuals, and 5. skipping disjunctions.

Weighted individuals and disjunctions can be established using a weighted queue. Disjunctions with the highest weight are branched on first.

Application of consistency rules to only a subset CX of individuals, and only branching on disjunctions related to those individuals, reduces the size of the consistency problem. This subset can be established using the universal quantifier construct of the form $\forall R.C = \{\forall b.(a, b) \in R \rightarrow b \in C\}$ (Baader et al., 2003), where R denotes a role relation and C denotes a class concept. It implies that all object fillers of role R, are of type C, resulting in adding the role filler type C to all objects for the given role R. Since this can give rise to an inconsistency, we define the subset CX as limited to the original individual I being checked for membership to the request RQ and all those individuals which relate to this individual I, via roles R specified in universal quantifiers.

Disjunctions and individuals in weighted queues, can be ranked by recursively checking an unapplied disjunction or element for a potential future clash. If a pathway to a clash is found, the weighted values of all individuals and disjunctions involved in this path are increased. Disjunctions can also be applied or skipped, according to whether they relate to the request type RQ, or not, respectively.

3 TABLEAUX OPTIMISATION AND RANKING ALGORITHMS

In this section we formally describe each optimisation strategy from the previous section.

3.1 Weighted Queue

A queue contains a weighted object and weight value object value pair. Let WO_i and WV_i denote these objects, where i denotes the current object. Let Q denote the queue where $Q = \{WO_1, WO_2.. WO_n\}$. Let WO_{next} denote the next WO to be returned by the queue. Each WV object is associated with an integer weight value representing the current weight of the object, let IV_i denote this value. Let MV denote the highest IV in the queue. $MV = \max(IV_{[1..n]})$ and IV has the range $0 \leq IV_{[1..n]} \leq MV$. Let NW denote a normalised decimal weight value where $0 \leq NW \leq 1$, calculated on the fly, with: $NW_i = IV_i / MV$.

Let Q^{ind} denote the individual queue and let Q^{disj} denote the disjunction queue. Let WO^{ind} and WO^{disj} denote individuals and disjunctions, respectively, as weighted objects, such that $Q^{ind} = \{WO^{ind}_1, WO^{ind}_2.. WO^{ind}_n\}$ and $Q^{disj} = \{WO^{disj}_1, WO^{disj}_2 .. WO^{disj}_n\}$ where n may be different for each queue Q. Each WO^{ind}_i contains a separate Q^{disj}. Weighted objects WO_i in any queue Q are ordered by their NW_i in descending order [1..0] and several WO_i objects can have the same NW_i. When a WO^{disj} is applied it is removed from the queue and there are no more disjunctions to apply when $Q^{disj} \equiv \{\}$. In a Q^{disj} the next disjunction is $WO^{disj}_{next} = WO^{disj}_1$.

The individual queue Q^{ind} has a floating weight threshold value, let WT denote this value. WT is initially set to the highest weight in the queue, $WT = \max(NW_{[1..n]})$. Let WTS denote the set of individuals WO^{ind}_i, with $NW_i \geq WT$. The next individual WO^{ind}_{next} is the next element in WTS, which is repeatedly iterated over. When all WO^{ind}_i elements in WTS contain a $Q^{disj}_i \equiv \{\}$, WT is set to the next highest weight NW, in Q^{ind}, NW = max($NW_{[1..n]}$) < WT.

3.2 Selectively Apply Consistency Rules

Individuals are iteratively added to the weighted individual queue Q^{ind}, as follows. Let CX denote the set of individuals which were last added to Q^{ind}. Originally, CX = {X}. Loop individuals in set CX. Let Y denote the current individual from the CX. Let AV denote a universal quantifier expression and let AVS denote the set of all universal quantifiers for individual Y such that AVS = {AV_1, AV_2...AV_m}. Let R_i denote the relation to which AV_i relates to and let RS denote a set of all distinct R relations to which any AV_i in the set AVS, relates. Let OS denote the a set containing those individuals which are objects O of any role R_j in RS, for the individual Y, such that OS = {O_1, O_2..O_{mn}}. Add all of the elements in OS to the weighted individual queue Q^{ind} and increment the weight value WV of these individuals by 1. Set CX = OS.

3.3 Rank Individuals and Disjunctions

The following algorithm is used by both the rank individual and rank disjunction strategies to establish a path of individuals and disjunctions to a potential clash. Let CS denote this path. The path may be via conjunction/disjunction elements and role fillers of universal quantifiers. When ranking individuals let C denote the last applied non-clashing disjunction element, let WO^{ind} denote the individual to which the disjunction relates, let CS = {} and execute ClashDetect once. When ranking disjunctions, execute ClashDetect once, for each disjunction WO^{disj}_i in the disjunction queue Q^{disj}_{ii}, of each individual WO^{ind}_{ii} in the individual queue Q^{ind} and let WO^{ind} = WO^{ind}_{ii}, C = WO^{disj}_i and CS = {}. If the algorithm returns a non-empty clash path set CS ≠ {}, increment the weight value WV for all the elements (individuals and disjunctions) within CS by 1. The pseudo code for ClashDetect is given below:

```
ClashDetect:
Inputs(WO^ind, C, CS), Outputs(Set).
If C is primitive, negation, normal or
value, then:
   If WO^ind has type negation of C,
     then:
     CS = WO^ind + CS. return CS.
   Else:
     UCS{} = unfold(C).
     for each UC in UCS:
        CS = ClashDetect(WO^ind,
           UC, CS).
     If CS is not null,
        then: return CS.
If C is a disjunction, then:
   For each disjunction element E
     in C:
     CSNEW{} = ClashDetect(WO^ind,
        E, CS).
     If CSNEW is null,
        then: return null.
        CS = CS + CSNEW.
If C is a conjunction, then:
   Create CSS (a set).
   For each conjunction element E in C:
     CSNEW{} = ClashDetect(WO^ind,
        E, CS).
     CSS = CSS + CSNEW.
   CS = SelectBestCS(CSS).
   Return CS.
If C is a universal quantifier,
   then:
   AVR = Role of C.
   AVC = Role filler type of C.
   OS{} = all the objects of
     WO^ind for role AVR. CSS = {}.
   For each individual O in OS:
     CSNEW{} = ClashDetect(O,
        AVC, CS).
     CSS = CSS + CSNEW.
   CS = SelectBestCS(CSS).
   Return CS + WO^ind.
```

Note SelectBaseCS selects the best clash pathway which has fewest disjunctions and depends on types which were added by the earliest branches.

3.4 Disjunction Skipping

The following strategy determines whether disjunctions are applied to create a new branch or skipped. Let C denote the service request type definition, which is a conjunction. Let DC = {T_1, T_2.. T_n} denote a set of valid class types. Let DI denote any disjunction being added to the disjunction queue Q^{disj}, by the reasoner or other optimisation strategy. DI is of the form DI = {D_1, D_2..D_m}, where D_i is a disjunction element. Let NND_i denote D_i in non-negated form (as a positive term). If DC contains any $NND_{[1..m]}$ from the disjunction DI, then DI is added to Q^{disj} with a weighting of 1, otherwise it is skipped.

The set DC is filled using the following population algorithm. Where C (service request) is a conjunction or disjunction list of the form C = {E_1, E_2..E_o}, let E_j denote an element of the list. Let NNE_j denote E_j in non-negated form. Add all elements $NNE_{[1..o]}$ to DC, set DC = DC + NNE_j

Table 1: Optimisation strategies enabled in each test.

Test #	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
Selective Consistency	×	×	×	×	×	×		×	×						
Rank Disjunctions			×	×				×	×	×		×			×
Rank Individuals		×	×			×			×	×			×	×	×
Skip Disjunctions	×	×	×	×			×			×				×	

Recursively reapply this DC population algorithm on each $E_{[1..o]}$ by setting C = E_j. If E_j is a primitive or nominal type, unfold each E_j and NNE_j into a new set UCS, such that UCS = {UC_1, UC_2..UC_p}, and then also recursively reapply this population algorithm on each unfolded type UC_k by setting C = UC_k.

4 PERFORMANCE EVALUATION

We implemented the optimisation strategies defined in section 3, in the Pellet v1.5 reasoner. We selected Pellet because it is open source while the other reasoners were not, allowing us to provide a proof of concept and compare performance with and without the strategies enabled.

The evaluation was performed on a HP iPAQ hx2700 PDA, with Intel PXA270 624Mhz processor, 64MB RAM, running Windows Mobile 5.0 with Mysaifu Java Virtual Machine (JVM), J2SE allocated 15mb of memory.

We implemented the scenario outlined in section 2.3 to create ontologies containing 141 classes, 126 roles and 337 individuals. Due to the resource intensive nature of XML parsing, we pre-parsed OWL XML files into text files of triples and postpone XML parsing to future work.

We executed a single consistency check to compare matching individual LaserPrinter1, against the service request, 15 times using various randomly selected combinations of our strategies, as shown in table 1, where test 11 represents normal Tableaux execution (no optimisations). Tests 1-11 provided the expected positive matching result and 12-15 did not complete due to lack of memory.

Figure 1 illustrates the performance of each successful test in terms of time (seconds). Consistency time involves application of consistency rules and branching, to perform the Tableaux consistency check for LaserPrinter1 $\in$ Request. This also encompasses the overhead cost for performing the optimisations, which is also shown separately. The total includes consistency time as well as the time required for preparing the reasoner (eg loading triple text files into the reasoner).

As observed in Figure 1, the our optimisation strategies considerably reduce the consistency time required to find a clash for all models of the query compared to test 11 which represents normal execution of Tableaux. We found that consistency time was influenced by the number of branches and rules applied.

Figure 2 presents a breakdown of how much time each strategy contributed to the overhead cost. Tests 8, 9 and 10 suffered particularly costly optimisation overhead, due to ranking disjunctions. This was because either selective consistency or skip disjunctions, or both, were disabled, resulting in more disjunctions to rank.

Test 5, 7 and 2 show that selective consistency and skip disjunctions are the most effective optimisations, especially when used together, and have low overheads. Rank disjunction and individual strategies were found to reduce the number of branches applied, but did not provide any performance improvement due to the high overhead.

Our performance tests show that some of our optimisations and ranking algorithms are very effective in improving Tableaux reasoning performance on resource limited devices, compared with no optimisations. This makes mobile reasoning feasible on resource constrained devices.

5 CONCLUSION AND FUTURE WORK

Our optimisation strategies were shown to significantly improve the performance of reasoning tasks on small resource constrained devices, making deployment on these devices feasible. Some strategies performed well, while others require future work and we implementing a greater number of scenarios to test our strategies more thoroughly. We will also leverage our weighted approach, to adaptively skip branches which have a lower weight, when resources are low. This will provide a result with a level of uncertainty rather than “Out Of Memory” errors, to better manage the trade-off between resource availability and result precision. In addition, although we demonstrated the use of

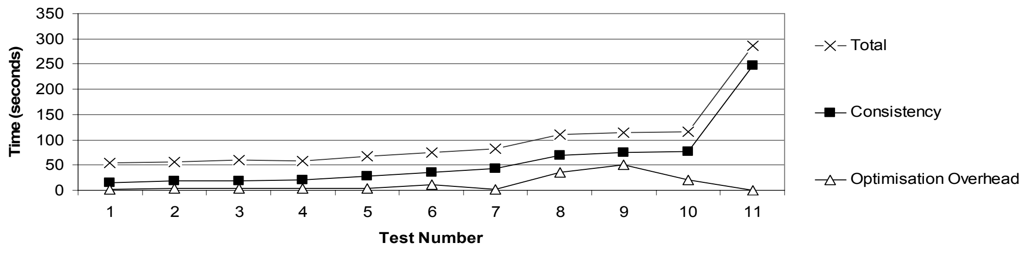

Figure 1: Processing time taken to perform each test.

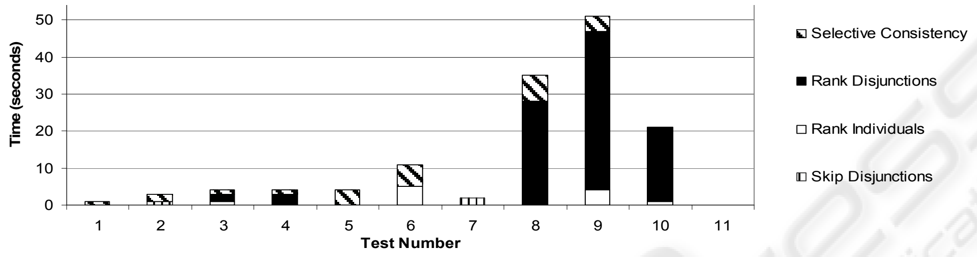

Figure 2: Optimisation strategy overhead breakdown.

context-aware attributes in our scenario, we will use context to pre-emptively rank the pool of discoverable services based on user preferences and current user context, before a request takes place. These services will be matched first as they are more likely to meet the user's needs.

REFERENCES

Web Ontology Language (OWL). from http://www.w3.org/2004/OWL.

Arnold, K., B. O'Sullivan, R. W. Scheifler, J. Waldo & A. Woolrath (1999). *The Jini Specification*, Addison-Wesley.

Baader, F., D. Calvanese, D. L. McGuinness, D. Nardi & P. F. Patel-Schneider (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*, Cambridge University Press.

Broens, T. (2004). Context-aware, Ontology based, Semantic Service Discovery. Enschede, The Netherlands, University of Twente**:** 87.

Cabral, L., J. Domingue, S. Galizia, A. Gugliotta, V. Tanasescu, C. Pedrinaci, et al. (2006). IRS-III: A Broker for Semantic Web Services based Applications. *5th International Semantic Web Conference (ISWC 2006)*, Athens, GA, USA.

Doulkeridis, C., N. Loutas & M. Vazirgiannis (2005). A System Architecture for Context-Aware Service Discovery.

FaCT++. (2007). Retrieved May 1, 2007, from http://owl.man.ac.uk/factplusplus/.

Horrocks, I. & P. F. Patel-Schneider (1999). Optimising Description Logic Subsumption. *Journal of Logic and Computation 9(3)*, 267 - 293.

Horrocks, I. & U. Sattler (2005). A Tableaux Decision Proceedure for SHOIQ. *19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, Morgan Kaufman.

KAON2. (2007). Retrieved June 21, 2007, from http://kaon2.semanticweb.org.

Kleemann, T. (2006). Towards Mobile Reasoning. *International Workshop on Description Logics (DL2006)*, Windermere, Lake District, UK.

Küster, U., B. König-Ries & M. Klein (2006). Discovery and Mediation using DIANE Service Descriptions. *Second Semantic Web Service Challenge 2006 Workshop*, Budva, Montenegro.

Pellet. (2007). Retrieved October, 2007, from http://pellet.owldl.com.

RacerPro. (2007). Retrieved May 23, 2007, from http://www.racer-systems.com.

Roto, V. & A. Oulasvirta (2005). Need for Non-Visual Feedback with Long Response Times in Mobile HCI. *International World Wide Web Conference Committee (IW3C2)*, Chiba, Japan.

Srinivasan, N., M. Paolucci & K. Sycara (2005). Semantic Web Service Discovery in the OWL-S IDE. *39th Hawaii International Conference on System Sciences*, Hawaii.

Steller, L., S. Krishnaswamy & J. Newmarch (2006). Discovering Relevant Services in Pervasive Environments Using Semantics and Context. *3rd International Workshop on Ubiquitous Computing (IWUC-2006). In conjunction with ICEIS 2006.*, Paphos, Cyprus, INSTICC Press.

Sycara, K., S. Widoff, M. Klusch & J. Lu (2002). LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace. *Autonomous Agents and Multi-Agent Systems 5*, 173-203.

Universal Plug and Play (UPnP). (2007). Retrieved March 12, 2007, from http://www.upnp.org.