IGI PUBLISHING **ITJ 5113**
701 E. Chocolate Avenue, Hershey PA 17033-1240, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.igi-global.com

This paper appears in the publication, International Journal on Semantic Web & Information Systems, Volume 5, Issue 2
edited by **Amit Sheth © 2009, IGI Global**

Enabling Scalable Semantic Reasoning for Mobile Services

Luke Albert Steller, Monash University, Australia
Shonali Krishnaswamy, Monash University, Australia
Mohamed Methat Gaber, Monash University, Australia

ABSTRACT

With the emergence of high-end smart phones/PDAs there is a growing opportunity to enrich mobile/pervasive services with semantic reasoning. This article presents novel strategies for optimising semantic reasoning for realising semantic applications and services on mobile devices. We have developed the mTableaux algorithm which optimises the reasoning process to facilitate service selection. We present comparative experimental results which show that mTableaux improves the performance and scalability of semantic reasoning for mobile devices.
[Article copies are available for purchase from InfoSci-on-Demand.com]

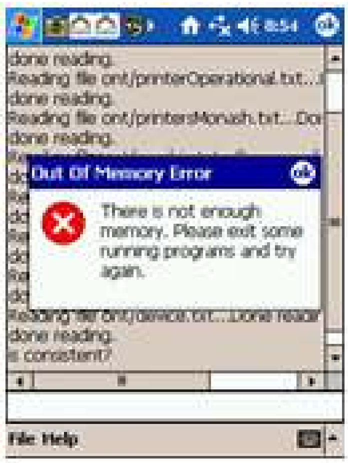
Keywords: *Optimised Semantic Reasoning; Pervasive Service Discovery; Scalable Semantic Matching*

INTRODUCTION

The semantic web offers new opportunities to represent knowledge based on meaning rather than syntax. Semantically described knowledge can be used to infer new knowledge by reasoners in an automated fashion. Reasoners can be utilised in a broad range of semantic applications, for instance matching user requirements with specific information in search engines, matching match client needs with functional system components such as services for automated discovery and orchestration or even providing diagnosis of medical conditions. A significant drawback which prevents the large uptake and deployment of semantically described knowledge is the resource intensive nature of reasoning. Currently available semantic reasoners are suitable for deployment on high-end desktop or service based infrastructure. However, with the emergence of high-end smart phones / PDAs the mobile environment is increasingly information rich. For instance, information on devices may include sensor data, traffic conditions, user preferences or habits or capability descriptions of remotely invokable web services hosted on these devices. This information is can be highly useful to other users in the environment. Thus, there is a need to describe this knowledge semantically and to support scalable reasoning for mobile semantic applications, especially in highly dynamic environments where high-end infrastructure is unsuitable or not available. Computing power is limited to that available on resource constrained devices and as shown in Figure 1, there is insufficient memory on these devices to complete reasoning tasks which require significant time and memory to complete.

Since mobile users are often on the move and in a highly dynamic situation, they generally require information quickly. 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. Therefore, there is a need for mobile reasoners which can meet the twin constraints of time and memory.

For example, consider the following mobile application scenario. A mobile user has just arrived in Sydney airport and wishes to search for food and other products. Sydney airport provides touch screen kiosk terminals which allow the user to search for stores (and other airport facilities) by category. The location of



```
done reading.
Reading file ont/printerOperational.txt...
done reading.
Reading file ont/printersMonash.txt...Do
done reading.
Reading file ont/device.txt...Done readi
done reading.
is consistent?
```

Out Of Memory Error

There is not enough memory. Please exit some running programs and try again.

Figure 1. Error showing that there was not enough memory to perform reasoning when attempting to run Pellet on a PDA (the reasoning task was the Printer inference check given in section 6.1).

the store and facility is then displayed on a map as well as the location of the user (which is the fixed location of the kiosk), as illustrated in Figure 2. These kiosks are not very convenient as they are only located at fixed point locations, are limited in their search options and user request complexity and do not take user context into account. Additionally, they do not scale, as kiosks can only be used by one user at a time.

Alternatively, the increasing abundance of mobile devices such as PDAs and mobile phones as well as their increasing computational and communication capabilities provide new opportunities for on-board service discovery. Consider the case where the information kiosk is a directory/repository of services available in the airport which mobile users can connect to from their phone or PDA. The user can then access, search and use this information using their respective phones at their convenience.

There are two modes of service matching:

- centralised service matching which occurs on a server on behalf of the user and
- partially or completely decentralised approaches where matching occurs on the resource constrained device itself.

Under a centralised approach (see Figure 3) the kiosk (or a connected machine) is a high-end server which handles all service discovery requests on the mobile user's behalf. However, there are two major drawbacks with this approach. Firstly, although purchase of a server is relatively cheap, there are significant costs involved for this kind of service provision, including scalability to handle potentially thousands of requests, wireless network provision, maintenance costs, security considerations and quality of service issues. The significant costs would outweigh the limited benefit to a central authority such as the Sydney airport. In environments where there is no such central authority this infrastructure may not even be possible (eg a city center or decentralised precinct). Secondly, if users are faced with the choice of paying for wireless access to a service matcher

Figure 2. Sydney airport store finder kiosk. The store search screen is shown on the left, while the search result for an Internet café is on the right. The location of the Internet café is indicated by the computer icon in the bottom right side of the screen.

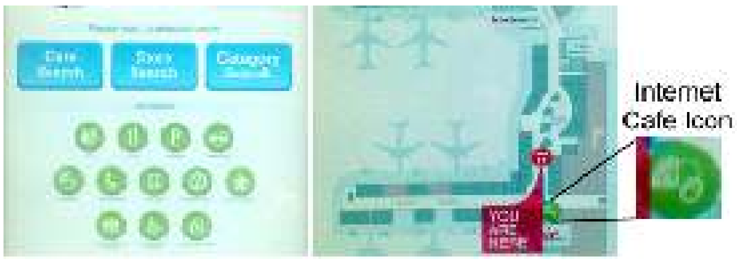


Figure 3. Example: Centralised server-based matching provision

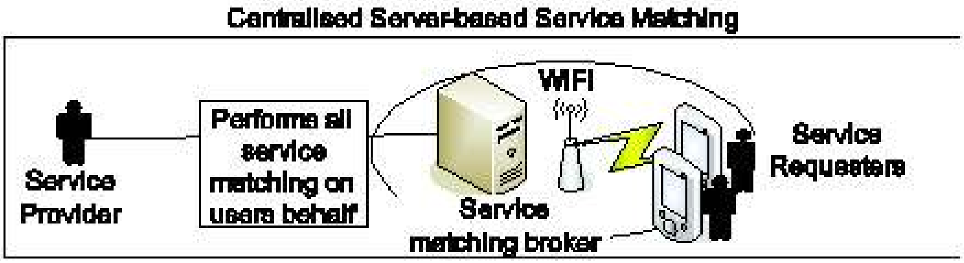
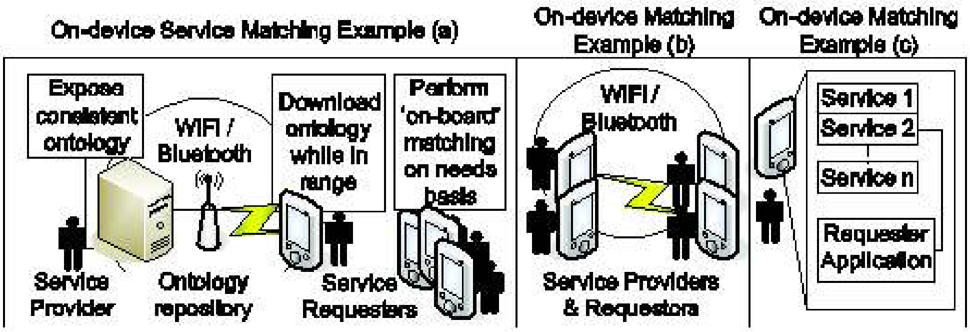

Centralised Server-based Service Matching
Service Provider
Performs all service matching on users behalf
WIFI
Service matching broker
Service Requesters

Figure 4. Three example configurations of on-device matching: (a) partial decentralisation where files are served centrally (by a WiFi/Bluetooth connected server or Internet provider) but matching occurs on-board the device, (b) on-device matching of remote services hosted on other mobile devices in a mobile ad-hoc network (c) on-device matching of services on the same device (local services only).


On-device Service Matching Example (a)
Expose consistent ontology
WIFI / Bluetooth
Download ontology while in range
Perform 'on-board' matching on needs basis
Service Provider
Ontology repository
Service Requesters
On-device Matching Example (b)
WIFI / Bluetooth
Service Providers & Requestors
On-device Matching Example (c)
Service 1
Service 2
Service n
Requester Application

or utilising existing kiosks such as those already at Sydney airport, they are likely to choose the kiosk since it is free (albeit limited in its service provision capability).

For this environment we advocate a partially decentralised approach (see Figure 4a) in which the kiosk is merely a directory or repository, to which users can sync with, to download service advertisements for the airport, using short range

WiFi or Bluetooth. The ontology file provider could also be a provider accessed via the Internet or even shared using secondary storage such as an SD card downloaded previously at home or by another person. Service matching can then occur independently as needed, on the user's device itself. This solution would be inexpensive to deploy and to use as there are no overheads for the service providing authority and there are no connectivity overheads for the user (eg they may simply use Bluetooth for once-off access to the service descriptions). In addition, this model would be better suited to provision of personalised selection by factoring in historical / user preference data.

There are also other application scenarios which demand on-device matching. For instance, a user may wish to discovery services which are hosted remotely by devices in a temporary mobile ad-hoc network (see Figure 4b) such scenarios include: students sharing data on a field trip (Chatti, Srirama, Kensche & Cao, 2006), emergency situations, traffic information sharing, etc. Alternatively, services may be installed or removed from a user's own device on a needs basis. Determining which services should be installed or removed requires comparing current or prospective services to the user's current needs on the device itself (see Figure 4c), for example Google[1] and Yahoo[2] already offer many mobile applications such as blogging, news, finance, sports, etc.

We have provided three examples demonstrating a growing number of situations where there is a clear need for approaches to enable mobile reasoning on resource constrained devices. The next question remains as to how the user will access these services from the mobile device and perform service discovery on the device. There are two main challenges here:

1. the mechanism to perform semantically-driven service selection on a mobile device in an efficient way;
2. the interface challenges of presenting this information to the user.

In order to facilitate the matching of user needs, context and requests with a set of potential services such as those outlined in the scenarios above, our focus is on the first key issue of enabling scalable service discovery mechanisms to operate on a mobile device. This approach requires new strategies to enable mobile reasoning on resource constrained devices, to perform matching of request to services. The Tableaux algorithm is well known and used by reasoners such as Pellet, RacerPro and FaCT++. Therefore this article aims to enable these reasoners to perform mobile semantic reasoning. The key challenge is to enable semantic reasoning to function in a computationally cost-efficient and resource-aware manner on a mobile device.

In this article 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 can be scaled to small resource constrained devices. We present comparative evaluations of the performance of Pellet, RacerPro and FaCT++ semantic reasoners which demonstrate the significant improvement to response time achieved by our mTableaux algorithm. In order to gain efficiency, some strategies reduce completeness, in a controlled manner, so we also evaluate result accuracy using recall and precision. Finally, in our evaluation we present experimental evaluations that demonstrate the feasibility of the semantic service discovery to operate on a mobile device.

This article 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 article is structured as follows. In section 2 we describe related work. In section 3 we present our discovery architecture, followed by a discussion of our optimisation and ranking strategies in section 4. In section 5 we formally define our strategies. Section 6 we provide an implementation and performance evaluations and in section 7 we conclude the article.

RELATED WORK IN PERVASIVE SERMANTIC SERVICE REASONING

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) and will include support for partial matching in the absence of an exact match. While current service discovery architectures such as Jini (Arnold, O'Sullivan, Scheifler, Waldo & Woolrath, 1999), UPnP (UPnP, 2007), Konark (Lee, Helal, Desai, Verma & Arslan, 2003), SLP (Guttman, 1999), Salutation (Miller & Pascoe, 2000) and SSDM (Issarny & Sailhan, 2005), UDDI (UDDI, 2009) and LDAP (Howes & Smith, 1995) use either interface or string based syntactic matching, there is a growing emergence of DAML-S/OWL-S semantic matchmakers. DReggie (Chakraborty, Perich, Avancha & Joshi, 2001) and CMU Matchmaker (Srinivasan, Paolucci & Sycara, 2005) are examples of such matchmakers which support approximate matching but they require a centralised high-end node to perform reasoning using Prolog and Racer, respectively. Similarly, LARKS (Sycara, Widoff, Klusch & Lu, 2002) which is designed to manage the trade-off between result accuracy and computation time, employs a centralised approach but defines its own language and reasoner. IRS-III (Cabral, Domingue, Galizia, Gugliotta, Tanasescu et al., 2006) is based on WSMX (WSMO, 2009) and utilises Lisp. DIANE (Küster, König-Ries & Klein, 2006) is designed for ad-hoc service discovery and defines its own semantic language. It captures request preferences as fuzzy sets defining acceptable ranges. DIANE also supports dynamic attributes, which are realised at runtime. Anamika (Chakraborty, Joshi, Yesha & Finin, 2004) is an ad-hoc architecture which utilises an ontological approach for routing and discovery based on service type but does not perform complex reasoning or support context.

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) which utilised RDF subclass relations for service type, with no reasoning, COSS (Broens, 2004) which utilises semi-OWL for service type, inputs and outputs with lattice structures for ranking Boolean context attributes, and CASE (Sycara et al., 2002) and Omnipresent (Almeida, Bapista, Silva, Campelo, Figueiredo et al., 2006) which utilise OWL with Jena (Jena, 2009) rules. However all of these architectures too, require the existence of a high-end central 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 (including Prolog, Lisp and Jess, as well as more newly available OWL reasoners such as FaCT++ (2008), RacerPro (2008) and KAON2 (2008)) are all resource intensive. These reasoners cannot be deployed onto small resource constrained devices in their current form, due to the twin constraints of memory and processing time.

Kleeman et. al. (Kleemann, 2006) 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 implements the common DL optimisations of backjumping, semantic branching, Boolean constraint propagation, lazy unfolding and absorption as described in (Horrocks & Patel-Schneider, 1999). These optimisations are also implemented by widely used reasoners such as FaCT++ and Pellet. A performance evaluation shows that it performs first order reasoning quickly, solving 35% of satisfiable horn clauses, 29% of unsatisfiable clauses, 54%, non-horn satisfiable problems, 39% of non-horn unsatisfiable problems in 10 seconds. It does not utilise caching schemes which incur additional 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 exhausts all memory when the reasoning task becomes too large for a small device to handle and fails to provide any result.

Therefore, there is a need for an optimised semantic reasoner which performs better than currently available reasoners. This reasoner must also support adaptation to the environment, to reduce memory consumption of the processing required (which may reduce result accuracy) according to resource or time constraints. In the next section we outline our novel architecture to meet this need.

RESOURCE-AWARE AND COST-EFFICIENT PERVASIVE SERVICE DISCOVERY

Our pervasive service discovery architecture is illustrated in Figure 5. The modules in this diagram all reside on the user's device. The database of ontologies includes those collected from service repositories or kiosks or other sources, as described in section 1.

In this model, the mobile user submits a request to his or her device and discovery manager utilises the semantic reasoner to match the request with services from the database of collected ontologies. The discovery manager takes available resources such as available memory, CPU usage, remaining battery life or remaining time (provided by the context manager), into consideration. It may load the entire ontology into memory in the beginning, or if memory is low it will load portions of ontology on demand. The adaptive discovery manager also may stop matching a particular request with a service after the service failed to match a particular request attribute or it may instruct the mTableaux reasoner to reduce the accuracy of its result when resources become low (eg low memory) or when the result is taking too long to process. The semantic reasoner module contains our mTableaux algorithm, which incorporates our optimised reasoning strategies. It also includes strategies to reduce result accuracy to meet resource constraints.

In summary, our architecture addresses two main goals. Firstly, it addresses the need for scalable reasoning on a mobile device by providing strategies to optimise the reasoning process. Secondly, when there are not enough resources or time remaining to complete a request, our architecture provides strategies to reduce the result's accuracy in order to utilise less resources and time. This article concentrates on providing a semantic reasoner that is able to operate in on a mobile device (mTableaux module) and discuss this in more detail in the next section. As a simple extension to this reasoner we also discuss adaptive accuracy reduction to reduce resource or time consumption where there are insufficient resources to complete a task in full.

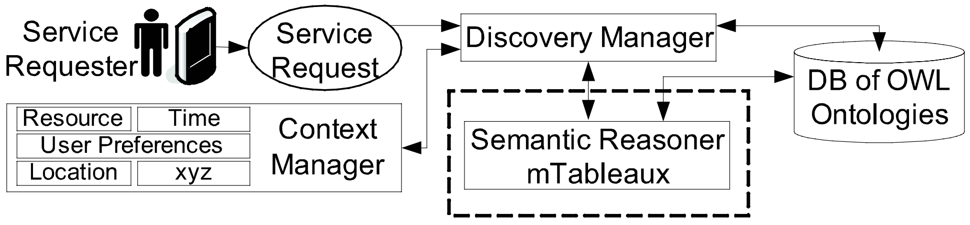
MTABLEAUX – REASONING FOR PERVASIVE SERVICE DISCOVERY

In this section we discuss current Tableaux semantic reasoners and present mTableaux, our algorithm for enabling Tableaux reasoning on mobile devices.

Semantic Reasoners

The effective employment of semantic languages requires the use of semantic reasoners such as Pellet (2003), FaCT++ (2008), RacerPro (2008) and KAON2 (2008). Most of these reasoners employ the widely used Tableaux (Horrocks & Sattler, 2005) algorithm. These reasoners are shown in Figure 6, which is a detailed version of the semantic reasoner and ontology database components from Figure 5 and illustrates the component parts required for

Figure 5. Pervasive service discovery architecture



OWL reasoning. Reasoners can be deployed on servers and interacted with via DL Implementation Group (DIG) interface specification which uses XML over HTTP. Alternatively, interaction may be facilitated directly using native APIs, which requires RDF/XML parsing functionality to load OWL files into the reasoner. Pellet utilises either Jena or OWL-API for interaction and RDF parsing.

Semantic OWL Reasoners contain a knowledge base K which 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. The knowledge base is stored as a set of triples $<C, R, O>$, where C is the set of classes, R is a set of roles and O is the set of object assertions. The object assertions are organised into a graph structure of the form $<O_1, R, O_2>$ where O_1 is an object connected to O_2 by role R. 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. 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 for I, in an otherwise consistent ontology. If the assertion of $I{:}\neg RQ$ results in a clash for all branches dependant on $\neg RQ$ for I, then class membership $I \in RQ$ is proven.

Figure 7 presents an example containing individuals d, e, f, g, h, i, j, k, n, m, o which are connected by roles Q, R, S, P and some individuals are asserted to be members of class types A, B, C, T. For instance, individual d is connected to f by role R and f is a member of class A. Assume we want to find the truth of $d \in RQ$ where $RQ = \exists P.(\geq 1P) \wedge \exists R.(A \wedge \exists R.(B \wedge C)$, using the Tableaux algorithm, $\neg RQ$ is first added asserted as a type label to individual d, where $\neg RQ = \forall P.(\leq 0P) \vee \forall R.(\neg A \vee \forall R.(\neg B \vee \neg C))$. Tableaux applies the first element of the disjunction, a universal quantifier: $\forall P.(\leq 0P)$, which asserts the max cardinality rule $\leq 0P$ to node e, because e is a P-neighbour to individual h. h violates the max cardinality of 0 for P and creates a clash, because e has a P-neighbour h. All remaining disjunction

Figure 6. Semantic reasoner components

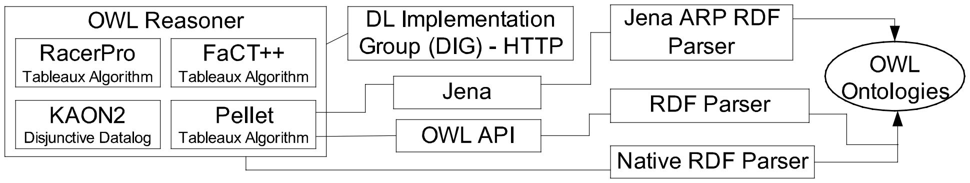
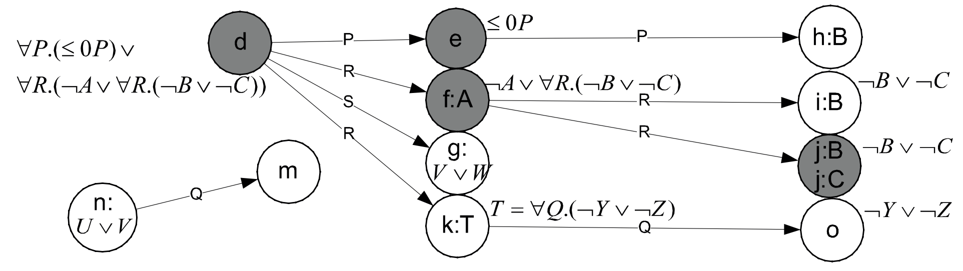


Figure 7. Example clash



$\forall P.(\leq 0P) \vee$
$\forall R.(\neg A \vee \forall R.(\neg B \vee \neg C))$
d
P
R
S
R
e
$\leq 0P$
P
h:B
f:A
$\neg A \vee \forall R.(\neg B \vee \neg C)$
R
R
i:B
$\neg B \vee \neg C$
j:B
j:C
$\neg B \vee \neg C$
g:
$V \vee W$
m
Q
n:
$U \vee V$
k:T
$T = \forall Q.(\neg Y \vee \neg Z)$
Q
o
$\neg Y \vee \neg Z$

elements and sub-elements also clash thereby proving $d \in RQ$ as true.

The shaded nodes in Figure 7 indicate those which contribute to a clash. Application of any expansion rules to other nodes results in unnecessary processing. The full Tableaux extract for the standard Tableaux method is listed in Box 1.

All elements of the negated request generate a clash, so $d \in RQ$ is proven to be true. Those disjunction branches and expansion rules which contributed to clashes proving $d \in RQ$ are bolded. The processing involved in applying all other rules did not contribute to the proof of $d \in RQ$.

mTableaux Strategies

The work in this article concentrates on optimisations for the Tableaux algorithm. As observed in section 4.1 (see Figure 7), Tableaux reasoners leave scope for optimisation by dropping rules which do not contribute to an inference check, or applying first the rules which are more likely to create a clash. In addition, since inference proofs relate only to a subset of the ontology, it is not necessary to load the entire ontology into memory. Minimising the processing time and memory consumption are the twin goals of our reasoning approach as this enables scalable deployment of reasoners to small/resource constrained devices. We provide an overview of our optimisations as follows.

Box 1.

```
Assert d: ∀P.(≤ 0P) ∨∀R.(¬A ∨ ∀R.(¬B ∨ ¬C)).
Apply Unfolding Rule k: ¬X ∨∀Q.(¬Y ∨¬Z)
Apply Universal Quantifier o: ¬Y ∨ ¬Z.
Apply Branch 1, Element (1/2) o:¬Y, no clash.
Apply Branch 2, Element (1/2) n:U, no clash.
Apply Branch 3, Element (1/2) i: ∀P.(≤ 0P)
    Apply Universal Quantifier j:≤ 0P
    Apply Max Rule j:≤ 0P, CLASH.
Apply Branch 3, Element (2/2) i: ∀R.(¬A ∨∀R.∀R.(¬B ∨ ¬C).
    Apply Universal Quantifier g:¬A ∨∀R.(¬B ∨ ¬C).
        Apply Branch 4 Element (1/2) g:¬A, CLASH.
        Apply Branch 4 Element (2/2) g: ∀R. (¬B ∨ ¬C).
            Apply Universal Quantifier l,j: ¬B ∨¬C.
            Apply Branch 6 Element (1/2) i:¬B, CLASH.
            Apply Branch 6 Element (2/2) i:¬C, no clash.
            Apply Branch 7 Element (1/2) j:¬B, CLASH.
            Apply Branch 7 Element (2/2) j:¬C, CLASH.
```

Semantic reasoners initially check ontologies for overall consistency. Since this check need only occur once for each ontology, we assume this has already been performed on the kiosk (i.e., the location from which the ontology is downloaded) or by the service advertiser before the ontology is released for download. Alternatively, there may be a service that is able to provide consistent versions of ontologies. Our mTableaux algorithm provides strategies to for reducing processing time and memory consumption for inference checks of the form: $I \in RQ$ by providing strategies for:

- **optimisation:** by dropping and reordering tableaux expansion rules and
- **adaption:** to reduce result accuracy when resources become low and only load ontology subsets which are relevant to the inference task.

The optimisation strategies include: 1. selective application of consistency rules, 2. skipping disjunctions, 3. associate weights with disjunctions and other expansion rules (such as existential quantifiers and cardinality restrictions) and increasing the weight of those which are likely to lead to clashes if applied in order to apply these first, by 3a. searching for potential clashes from specific disjunctions and 3b. searching from a specific term. The first two strategies drop expansion rules (disjunctions, existential quantifiers and maximum cardinality restrictions), therefore completeness cannot be guaranteed (soundness is in tact) because some clashes may not be found. The third optimisation alters the order in which expressions are applied, but does not skip any, thereby maintaining both completeness and soundness. We note, that most reasoners such as FaCT++ and RacerPro perform ontology realisation, in which all individuals are checked for inferred membership to every class type in the ontology. mTableaux does not require nor perform full ontology realisation, rather only specific individual I to class type RQ membership $I \in RQ$ is performed, where RQ is a user request and I denotes a set of potential service individuals to be checked.

In the first strategy (selective consistency), application of consistency rules to a subset of individuals only, reduces the reasoning task. 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, Calvanese, McGuinness, Nardi & Patel-Schneider, 2003), where R denotes a relation and C denotes a class concept. The quantifier implies that all object fillers of relation R, are of type C. Application of this rule adds role filler type C to all objects for the given role R, which can give rise to an inconsistency. Therefore, we define the subset as being limited to the original individual being checked for membership to a class, and all those individuals which branch from this individual as objects of roles specified in universal quantifiers.

The second optimisation (disjunction skipping), applies or skips disjunctions, according to whether they relate to the request type. A disjunction may be applied when one of its elements contains a type which can be derived from the request type. Derived types include elements of conjunctions/disjunctions and role fillers of universal quantifiers and their unfolded types.

For the third strategy, expressions are ordered by weight using a weighted queue. To establish weights for expansion rules (disjunctions, existential quantifiers and maximum cardinality restrictions) these expressions are ranked by recursively checking each element in a particular disjunction (rank by disjunction) or asserted term (rank by term) for a potential clash. If a pathway to a clash is found, the weighted value is increased for of all expressions which are involved in this path.

The adaptive strategies involve simple extensions to the optimisation strategies to avoid exhausting the available memory or time by providing a result to the user with a level of uncertainty, when resources become low. We describe our optimisation strategies in detail, in the next section, which the adaptive extensions are briefly discussed in future work (section 7).

MTABLEAUX ALGORITHM - OPTIMISATION AND RANKING STRATEGIES

In this section we formally describe the optimisation strategies listed in the previous section.

Selective Consistency

In the selective consistency strategy, Tableaux completion rules are only applied to a subset of individuals, rather than all those individuals in the ontology, let *SC* denote this set. Completion rules which are added as types to individual *A* are only applied if $A \in SC$.

For the membership inference check $I \in RQ$, before reasoning begins, SC is initially populated using the function *populnds*(*IS*), such that *SC* = *populnds*({*I*}). *populnds*(*IS*) is a function which recursively calls *getInds(e, AV)* to select universally quantified *r*-neighbour individuals of *e*, and those neighbour's universally quantified *r*-neighbours, etc. *populnds(IS)* is given by equation 1, where *e.AV* denotes the set of universal quantifiers of the form $\forall R.C$ which have been added as type labels to an individual *e*.

$$populnds(IS) = \bigcup_{e \in IS} popInds(getInds(e, e.AV)) \qquad (1)$$

getInds(e, AV) is the function which returns the set of *r*-neighbours for the individual *e*, where the relation *r* is restricted by a universal quantifier of the form $\forall r.c$, which has been added as a type to the individual *e*. The function is given by equation 2, where *OS* is the set of objects in the triple <*e, r, OS*> that contains *e* and *r*, and *av* must be a universal construct. A universal quantifier can be added to *e* by the unfolding of a concept already added to *e* or by application of another expansion rule.

$$getInds(e, AV) = \bigcup_{av \in AV} OS, < e, r, OS >, av \rightarrow \{\forall r.c\} \qquad (2)$$

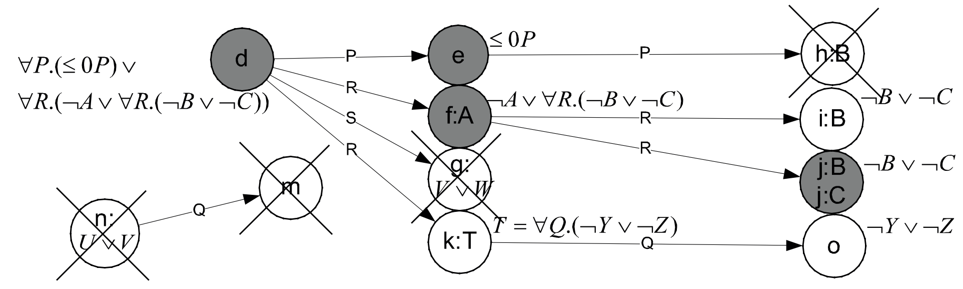
After reasoning has begun, new universal quantifiers may be added to an individual *a* which is in the set *SC*. If the new quantifier restrictions role *R* which is not yet restricted by another quantifier added to *a*, and a has *R*-neighbours, these neighbours need to be added to *SC*. Therefore, whenever a universal quantifier av_{new} is added an individual *a* in *SC*, R-neighbours are added to *SC* by a call to *getInds(e, AV)* such that $\{a.AV = a.AV + av_{new}\} \wedge \{SC = SC + addInds(a, \{av_{new}\})\}$ where $A \in SC$.

For example, for the inference check in section 4.1, $d \in RQ$, a call to *populnds*({*d*}) returns only {*d*} because *d* does not yet contain any universal quantifies. Application of the first element of the disjunction *RQ* asserts *d*: $\forall P.(\leq 0P)$. A call to *getInds*(*d, d.AV*) returns {*e*}, because *e* is a *P*-neighbour of *d* and *P* was restricted in $\forall P.(\leq 0P)$, thus *SC* = {*d, e*} therefore expansion rules for *e* can now be applied. Application of the second element in *RQ* asserts *d*: $\forall R.(\neg A \vee \forall R.(\neg B \vee \neg C))$ and a call to *getInds*(*d*, $d.AV_{new}$) returns {*f*} because *f* is an *R*-neighbour of *d* and *R* was restricted in $\forall R.(\neg A \vee \forall R.(\neg B \vee \neg C))$. Figure 8 illustrates that SC = {*d, e, f, i, j, k, o*}, therefore any expansion rules relating to all other individuals *n*, *m*, *g* or *h* were not applied (shown as crossed out in Figure 8).

Disjunction Skipping

When a disjunction is encountered during the reasoning process, the disjunction skipping strategy determines whether this disjunction is applied to create a new branch or skipped. Let *D* denote a disjunction, of the form $D = \{d_1 \vee d_2 \vee \ldots \vee d_m\}$, where d_i is a disjunction element. Let *nn*(*e*) denote *e* in non-negated form. Non-negated form implies that a negated term is made positive such that *nn*(*e*) = *x* if $e = \neg x$, or

Figure 8. Selective consistency



$\forall P.(\leq 0P) \vee$
$\forall R.(\neg A \vee \forall R.(\neg B \vee \neg C))$
d
P
R
S
R
e
$\leq 0P$
f:A
$\neg A \vee \forall R.(\neg B \vee \neg C)$
g: $V \vee W$
k:T
$T = \forall Q.(\neg Y \vee \neg Z)$
P
R
R
Q
h:B
i:B
$\neg B \vee \neg C$
j:B j:C
$\neg B \vee \neg C$
o
$\neg Y \vee \neg Z$
m
n: $U \vee V$
Q

nn(e) = *x* if *e* = *x,* where *x* is a class type name or logical expression. *D* is applied if at least one of its non-negated elements $nn(d_i)$ is contained within the set *DS*, such that $\exists_{d_i \in D}(d_i) \in DS$. Let *DS* denote a set of class type names and logical expressions defined in the ontology.

For the membership inference check $I \in RQ$, *DS* is populated using the *popu(E)* function such that $DS = popu(\neg RQ)$, where $\neg RQ$ is the negated request type definition. We assume *RQ* was a conjunction, $\neg RQ$ is a disjunction *D. popu(E),* given in expression 3, recursively collects terms which can be derived from elements in the set *E* of class terms or expressions.

$$popu(E) = \bigcup_{e \in E} nn(e) + pop(decomp(e)) \tag{3}$$

E may be a conjunction of the form $E = \{e_1 \wedge ... \wedge e_m\}$, a disjunction of the form $E = \{e_1 \vee ... \vee e_m\}$, or generic set $E = \{e_1 ,..., e_m\}$. Let *decomp*(*e*) denote the function which returns a empty or non-empty set, of terms and expressions which can be derived from *e. decomp*(*e*) is given in expression 4. Derived implies that where *e* is a universal or existential quantifier then *decomp*(*e*) returns a set containing the role filler for *e* or where *e* is a unary atomic term an empty or non-empty set is returned containing its expanded expressions, retrieved, using the *unfold(e)* function.

$$decomp(e) = \begin{Bmatrix} \{C\} & if & e = \forall R.C \vee e = \exists R.C, \\ unfold(e) & if & e^1 & \end{Bmatrix} \tag{4}$$

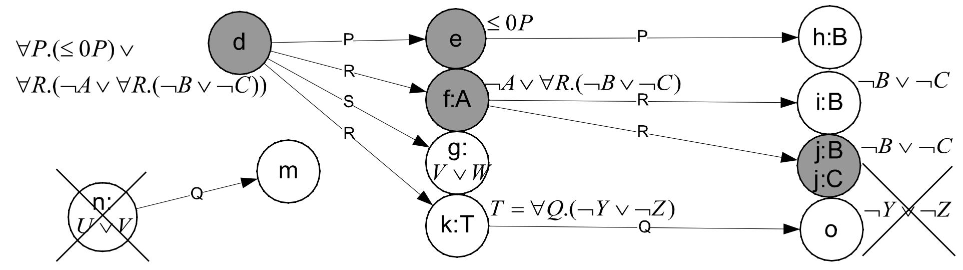
For example for the type check in section 4.1, $d \in RQ$, $\neg RQ$ unfolds to $\forall P.(\leq 0P) \vee \forall R.(\neg A \vee \forall R.(\neg B \vee \neg C))$. Therefore, $DS = popu(\neg RQ) = \{RQ, \forall P.(\leq 0P) \vee \forall R.(\neg A \vee \forall R.(\neg B \vee \neg C)), \forall P.(\leq 0P), \leq 0P, \forall R.(\neg A \vee \forall R.(\neg B \vee \neg C)), \neg A \vee \forall R.(\neg B \vee \neg C), A, \forall R.(\neg B \vee \neg C), \neg B \vee \neg C, B, C\}$. As a result, the disjunctions $\{U \vee Y\}$ and $\{\neg Y \vee \neg Z\}$ are skipped because none of their non-negated elements are contained in *DS*, while all other disjunctions are applied, as illustrated in Figure 9.

Weighted Disjunctions and Terms

This strategy seeks to manage the order in which completion rules for disjunctions, existential quantifiers and maximum cardinality in the knowledge base are applied, such that the expressions which are most likely to contribute to a clash, are applied first. The order of application for all other expressions remains arbitrary. This strategy does not compromise completeness.

A weighted queue *Q* is used in two instances. A weighted disjunction queue Q^{disj} maintains the order in which disjunctions will be applied for a particular individual *A*. The order of existential quantifier and maximum cardinality rule

Figure 9. Selective consistency



$\forall P.(\leq 0P) \vee$
$\forall R.(\neg A \vee \forall R.(\neg B \vee \neg C))$
d
P
R
S
R
e
$\leq 0P$
P
h:B
f:A
$\neg A \vee \forall R.(\neg B \vee \neg C)$
R
i:B
$\neg B \vee \neg C$
R
j:B
j:C
$\neg B \vee \neg C$
g:
$V \vee W$
m
Q
n:
$U \vee V$
k:T
$T = \forall Q.(\neg Y \vee \neg Z)$
Q
o
$\neg Y \vee \neg Z$

application is maintained by the role restriction queue Q^{rest}. A queue Q contains pairs <*object*(x), *weight*(x)> such that *object*(x) is an object and *weight*(x) is a positive integer representing the weight of *object*(x) and multiple *object*(x) can have the same *weight*(x). *nweight*(x) is a double value representing a normalised weight for *object*(x) *such that* $0 \leq$ *normalised*(x) ≤ 1. Normalised values are calculated by dividing the current weight by the highest weight in the queue, given by *nweight*(x) = *weight*(x)/$\max_{x \in Q^{ind}}$(*weight*(x)). Queue objects *object*(x) are given by the queue iterator in descending *nweight*(x) order [1..0].

This strategy employs two different approaches: disjunction weighting and term weighting. Both approaches utilise the *ClashDetect*(*C, I, CP*) function which attempts to find a pathway from term *C* (asserted to individual *I*) to a potential clash and returns a set *CP* containing terms (disjunctions, existential quantifiers and maximum cardinality expressions) if a clash pathway was found, or an empty set if no clash was found. All weight values *weight*(x) of expressions x in the clash pathway are incremented, such that $increment_{x \in ClashDetect(C,\ I,\ CP)}$(*weight*($x$)) and *increment*($v$) = v++. Note, if a term forms a clash path, but is not yet asserted to the individual, its weight is maintained by the queue and used in the event that it is added as a type for the individual.

ClashDetect(*I, C, CP*) calls the function which handles each kind of expression passed to it. For instance, if *C* is a maximum cardinality restriction it calls *CheckMaxRestriction*(*I, mx, CP*). *ClashDetect*(*I, C, CP*) pseudo code is given in Box 2. Each of the functions referred to in the above pseudo code, are described in Appendix A.

For example for the type check in section 4.1, $d \in RQ$, $\neg RQ$ unfolds to $\forall P.(\leq 0P) \vee \forall R.(\neg A \vee \forall R.(\neg B \vee \neg C))$. A clash pathway exists which includes: {d:¬RQ, e:$\leq 0P$, f: $\forall R.(\neg B \vee \neg C)$, j: $\neg B \vee \neg C$}. Therefore all the disjunctions and expressions involved in this path are incremented. The individuals involved are shaded in Figure 7, section 4.1. The queues are illustrated in Figure 10.

Now that we have detailed our optimisation strategies, we discuss our work in implementing the strategies in the next section. We also provide a performance evaluation comprising a comparison with current reasoners and performance on a resource-constrained device.

IMPLEMENTATION AND PERFORMANCE EVALUATION

In this section we provide two case studies in order to evaluate our mTableaux algorithm to answer the following two main questions:

1. How does mTableaux perform when compared to other reasoners?
    a. Since mTableaux does not guarantee completeness for all strategies, how much does mTableaux impact on result accuracy reduced, as measured using recall and precision?

Box 2.

```
ClashDetect:
Inputs: Let I be an individual, Let C be a type, Let CP
be a set of individuals and logic expressions involved in
a clash.
Outputs: CP
Switch(C)
Case C is primitive, negation, nominal or literal value:
    Return CheckPrimitive(I, C, CP).
Case C is a disjunction:
    Return CheckDisjunction(I, C, CP).
Case C is a conjunction:
    Return CheckConjunction(I, C, CP).
Case C is a universal quantifier logic expression:
    Return CheckUniversalQuantifier(I, C, CP).
Case C is an existential quantifier logic expression:
    Return CheckExistentialQuantifier(I, C, CP).
Case C is a maximum role restriction logic expression:
    Return CheckMaxRestriction(I, C, CP).
```

Figure 10. Example disjunction and role restriction queue

Q_{disj} *object(x)*	Q_{disj} *nweight(x)*
d: $\neg A \vee \forall R.(\neg B \vee \neg C)$.	1.0
j: $\neg B \vee \neg C$	1.0
i: $\neg B \vee \neg C$.	0
n: $U \vee V$.	0
o: $\neg Y \vee \neg Z$.	0

Q_{rest} *object(x)*	Q_{rest} *nweight(x)*
e: $\leq 0P$.	1.0

2. How does mTableaux scale in terms of meeting the twin constraints of processing time and memory usage on a mobile device?
    a. Does mTableaux enable successful completion of a reasoning task such that a result can be obtained on a resource constrained device (i. e., available memory was not exceeded)?
    b. Does mTableaux significantly improve performance compared to normal execution of Tableaux with no optimisation strategies enabled?
    c. Which mTableaux strategies or combination of strategies work best?
    d. Do different strategies work better for different scenarios / reasoning tasks?
    e. Do the optimisation strategies improve performance for positive as well as negative type checks?

We do this using two case studies as well as the Galen[3] ontology. Our two case studies are detailed in the next two subsections.

Case Study 1: Searching for a Printer

Bob is walking around at his university campus and wishes to locate laser printer-fax machine (to print some documents and send a fax). He issues a service request from his PDA for a listing of black and white, laser printers which support a wireless network protocol such as Bluetooth, WiFi or IrDA, a fax protocol and which have a dialup modem with a phone number. Equations 5-8 show Bob's request in Description Logic (DL) (Baader et al., 2003) form, while equation 9 presents a possible printer.

$$PrinterRequest \equiv PhModem \wedge \exists hasColour.\{Black\} \wedge hasComm.\{Fax\} \wedge LaserPrinterOperational \cap WNet \quad (5)$$

$$PhModem \equiv \exists hasComm.(Modem \wedge \geq 1\ phNumber) \quad (6)$$

$$LaserPrinterOperational \equiv Printer \wedge \exists hasCartridge.\ \{Toner\} \wedge \geq 1\ hasOperationalContext \quad (7)$$

$$WNet \equiv \exists hasComm.\{BT\} \wedge \exists hasComm.\{WiFi\} \wedge \exists hasComm.\{IrDA\} \quad (8)$$

Printer(LaserPrinter1), hasColour(LaserPrinter1, Black), hasCartridge(LaserPrinter1, Toner), hasComm(LaserPrinter1, BT), hasComm(LaserPrinter1, Fax), hasOperationalContext(LaserPrinter1, Ready), Modem(Modem1), hasComm(LaserPrinter1, Modem1), phNumber (Modem1, "9903 9999")

(9)

Note, these equations are simplified for illustrative purposes, the actual ontology used for this case study comprises 141 classes, 337 individuals and 126 roles. Equation 5 defines five attributes in the request, the first is unfolded into equation 6, specifying the printer must have a modem which has a phone number. The second attribute specifies a black and white requirement. The third attribute requires support for the fax protocol, and the fourth unfolds into equation 7, specifying a printer which has a toner cartridge and at least one operational context. The fifth unfolds into equation 8, which specified that one of the wireless protocols (Bluetooth, WiFi or IrDA) are supported. Equation 9 shows a DL fragment defining the LaserPrinter1 individual as meeting the service request. We also define an individual LaserPrinter2 as the same as equation 9, but without a phone number.

Case Study 2: Searching for a Movie Cinema

Bob is in a foreign city centre and has walked past several shops, short range ontology download points, and other people carrying devices with accumulated ontologies of their own. As such Bob collects a range of ontologically described service advertisements. He sits down in a park out of network range, and decides to find a movie cinema with a café attached which has a public phone and WiFi public Internet. He issues a request for a retail outlet which has at least 5 cinemas that each screen movies, has a section which sells coffee and tea, sells an Internet service which supports access using the WiFi protocol and sells a fixed phone service. We specify that an individual VillageCinemas matches the service request and GreaterUnionCinemas is the same as VillageCinemas except it provides Bluetooth Internet access rather than by WiFi, and therefore fails to match the request. The request specifies universal and existential quantifier and cardinality restrictions. The ontologies for this scenario contain 204 classes, 241 individuals and 93 roles.

Implementation

Our mTableaux strategies have been implemented as an extension to the Pellet 1.5 reasoner which supports OWL-DL with SHOIN expressivity. That is, mTableaux is implemented into the Pellet source tree. (Sirin, Parsia, Grau, Kalyanpur & Katz, 2007) discusses the implementation and design of Pellet. We chose Pellet because it 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: selective consistency, skip disjunctions, and rank by disjunctions and terms, and we evaluate the impact these have on performance in the next sections. We intend to make the source code for the system available for download on completion of the project.

Comparison of mTableaux with Other Reasoners

In order to show how mTableaux compares to other widely used OWL semantic reasoners, we provide a performance comparison with FaCT++ 1.1.11, RacerPro 1.9.2 beta and Pellet 1.5 without our optimisations. As stated in section 4.2, these reasoners perform an ontology "realisation" in which consistency checks are used to determine all the inferred class types for every individual in the ontology, $I_{[1, 2, .., n]} \in RQ_{[1, 2, .., m]}$, where n denotes the number of individuals in the ontology and m denotes the number of classes, resulting in $n.m$ possible individual and class combinations. Subsequent queries to the reasoner then draw from this pre-inferred data. Since an ontology realisation is unnecessary for service discovery in which specific service candidates are compared against single request class types, mTableaux does not perform an ontology realisation. Therefore, our performance evaluation presents two results for mTableaux one with full realisation and one where a subset of individuals are compared against a single user request class type such that $I_{[1, 2, .., n]} \in RQ$. The individuals represent discoverable services.

The evaluation was conducted on a Pentium Centrino 1.82GHz computer with 2GB memory with Java 1.5 (J2SE) allocated maximum of 500MB for each experiment. All times are presented are computed as the average of 10 independent runs. We performed our evaluation using both of the case studies described in section 6.1 and 6.2, as well as several publically available ontologies, including: Galen[iii], Tambis[4], Koala[5] and Teams[6]. Galen is a large ontology of medical terms with 2748 classes and 844 roles. Tambis, Koala and Teams ontologies have 183, 20 and 9 classes respectively. For each of our Printer and Product ontologies we checked 20 service candidates against the request printer and product user request, respectively. The Galen, Tambis, Koala and Teams ontologies did not contain individuals so we created a matching (positive) and non-matching (negative) individual for request each class type that we checked. The expected results for each ontology are illustrated in table 1.

Figure 11 presents the total time required to perform the 8 inference checks for the Galen ontology and Figures 12 and 13 present the total time to check all 20 service individuals against the user request class for the product and printer case studies, respectively. The 4 inference checks for each of the Tambis, Koala and Teams ontologies are not graphed because they completed in under 1 second.

As illustrated in Figure 11, mTableaux significantly outperformed the other reasoners for the Galen ontology, requiring only 0.67 seconds to perform the 8 inference checks. mTableaux with realisation almost performed as well as FaCT++ and outperforms RacerPro. Pellet with no optimisations performed poorly, requiring more than 40 seconds to complete. Figure 12 and 13 show that RacerPro performed worst, followed by Pellet, for the Product and Printer ontologies. mTableaux is slower when a full realisation is performed, because this compares irrelevant individuals against the user request. FaCT++ performed slightly better than mTableaux for the Product ontology, which we attribute to its implementation in C++. We note that mTableaux with realisation and FaCT++ could not complete the printer ontology and did not provide a result.

These results show, that our optimisation strategies significantly improve the performance of Pellet. We also observed that for all evalu-

Table 1. Expected results for each ontology

Ontology	Request Class	Positive	Negative	Total
Printer	PrinterRequest	3	17	20
Product	ProductRequest	3	17	20
Galen	BacterialGramPositiveStainResult	1	1	2
	FailureOfCellUptakeOfBloodGlu-coseDue-ToCellInsulinResistance	1	1	2
	AcutePulmonaryHeartDisease	1	1	2
	LocalAnaesthetic	1	1	2
Tambis	small-nuclear-rna	1	1	2
	peptidase	1	1	2
Koala	MaleStudentWith3Daughters	1	1	2
	KoalaWithPhD	1	1	2
Teams	MarriedPerson	1	1	2
	MixedTeam	1	1	2
Total		16	44	60

Figure 11. Reasoner comparison using galen ontology

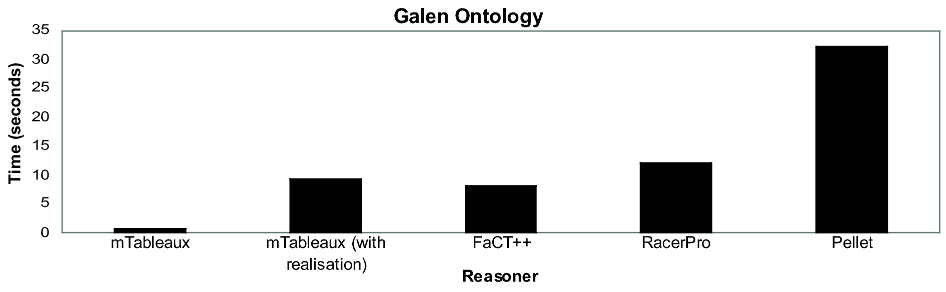
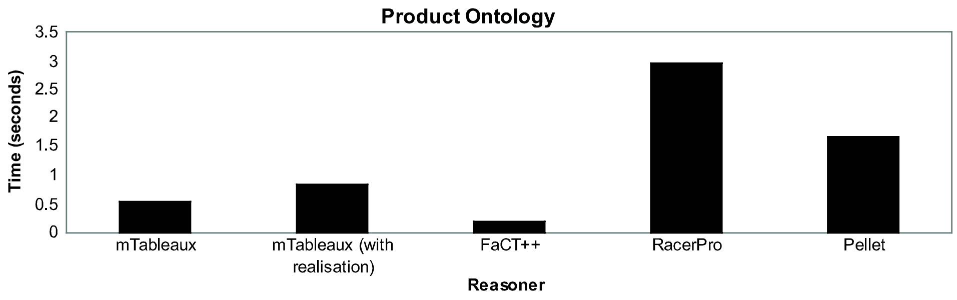
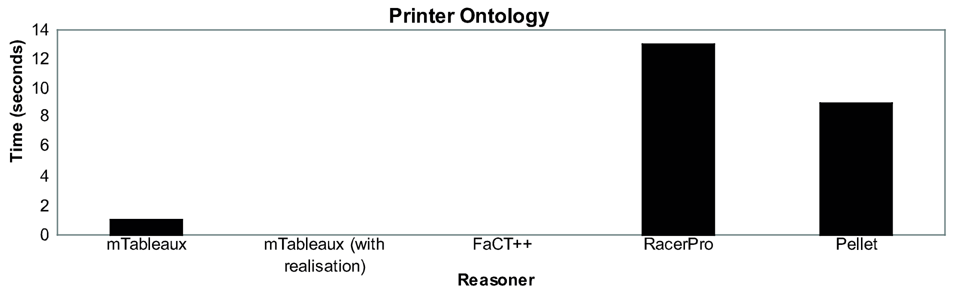


Figure 12. Product ontology reasoner comparison



ations the number of branches applied when using mTableaux was less than half that of Pellet. We conclude that when the amount of available memory available is constrained as on a small device, the performance improvements resulting from mTableaux will be significantly enlarged.

Figure 13. Printer ontology reasoner comparison



Since some strategies to not guarantee completeness, we measure the accuracy of mTableaux compared to other reasoners using recall and precision metrics, as illustrated in equations 10 and 11, where *x* denotes the number of service individuals which were expected to match but also actually found to match by the reasoner to match, *n* denotes the total number of service individuals which were expected to match (including any not returned by the reasoner) and *N* denotes the total number of service individuals which the reasoner claims do indeed match. Note that an expected match implies that a true match can be deduced by a reasoner in which completeness holds.

$$\text{Recall} = x / n \qquad (10)$$

$$\text{Precision} = x / N \qquad (11)$$

The recall and precision results obtained by completing the matching detailed in table 1, are provided in table 2. For instance mTableaux returned all 16 of the service individuals which were expected to match. The results show that the actual results were as expected for all reasoners except that FaCT++ did not match the positive individual with the class type MaleStudentWith3Daughters in the Koala ontology, because FaCT++ does not match Boolean literal values which were present in the request class type. Therefore, although mTableaux does not guarantee completeness for the selective consistency (SC) and skip disjunction (SD) strategies, there was no degradation in result accuracy on the ontologies tests in our evaluation. We conclude in data sets representing realistic scenarios such as the ones we used, mTableaux does not compromise result completeness as measured by recall and precision. In our tests, we checked to see whether ontology consistency was compromised by applying the negation of a specific class expression $\neg RQ$ to an individual *I*, in order to check whether the individual holds inferred membership to this expression $I \in RQ$. All applied expansion rules and disjunctions which led to clashes (causing an inconsistent ontology for all models) were the result of the negated expression $\neg RQ$ having been asserted. Since CS and SD strategies include or exclude individuals and disjunctions based on universal quantifies and expressions which result from the individual *I* and expression $\neg RQ$, respectively, there was no breach of completeness. Completeness may be compromised when the application of disjunctions, or expressions resulting from these disjunctions, do not relate to the expression *RQ*, which would result in a failure of mTableaux to prove a positive inference. In models of the knowledge base, parts of the ontology which do not relate to the class type *RQ* involved in the inference check may interact with each other to create clashes. It is in these cases where completeness is not guaranteed.

Since mTableaux outperformed all reasoners except for FaCT++ in some case while preserving completeness in our case studies, we now provide a performance evaluation to show how mTableaux performs on a small resource constrained device, in the next section. We also show which strategies work best together and the level of overhead incurred by using each optimisation.

mTableaux Performance on a Mobile Device

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) (Mysaifu, 2009), allocated 15MB of memory. We executed the four type check combinations shown in table 1, to evaluate both case study requests against a matching/positive and non-matching/negative service individual, defined as individual A and B, respectively. We executed each of the 4 consistency checks outlined in table 3 with every combination of the 4 optimisation strategies enabled (16 times). Table 4 indicates which strategies were enabled for each of the 16 tests (organised in bitwise order). Pellet with SHOIN expressivity was used for all tests. Test 16 represents normal execution of the Tableaux algorithm, with none of our optimisations strategies enabled. Successfully executed tests returned the expected result shown in table 3.

Figure 14 shows two graphs, which each show the consistency time to perform a type check for individual A and B against the request for the tests in table 3, using Pellet with SHOIN expressivity. The left and right graph present results for the printer and product case studies, respectively. Tests which did not complete due to insufficient available memory or which required more than 800 seconds to execute, omitted from the graph. In addition to consistency checking, an additional 35-40

Table 2. Total actual results for each reasoner

Reasoner	Actual Positive	Actual Negative	Recall	Precision
mTableaux	16	44	16/16 = 1.0	16/16 = 1.0
Pellet	16	44	16/16 = 1.0	16/16 = 1.0
RacerPro	16	44	16/16 = 1.0	16/16 = 1.0
FaCT++	15	45	15/16 = 0.937	15/15 = 1.0

Table 3. Type membership checks

Case Study	Request	Individual		Expected Result
Case Study 1	Fax Laser Printer	A	#LaserPrinter1 (with phone number)	Match
		B	#LaserPrinter2 (no phone number)	No Match
Case Study 2	Movie Cinema	A	#MovieCinema2 (WiFi Internet)	Match
		B	#MovieCinema2(Bluetooth Internet)	No Match

Table 4. Optimisation tests

Test	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
Selective Consistency		×		×		×		×		×		×		×		×
Skip Disjunctions			×	×			×	×			×	×			×	×
Rank by Disjunction					×	×	×	×					×	×	×	×
Rank by Term									×	×	×	×	×	×	×	×

Figure 14. processing time required to perform each test, for Selective Consistency (SC), Skip Disjunction (SD), Rank by Disjunction (RD) and Rank by Term (RT) strategies, showing total consistency time to perform an inferred membership check for matching individual A and non-matching individual B, for the Printer ontology (left) and Product ontology (right).

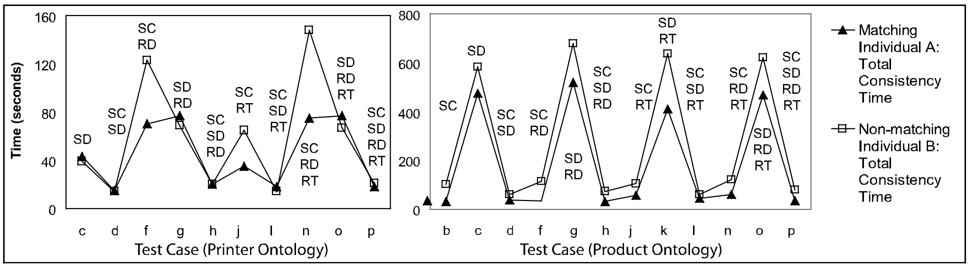


Table 5. Re-ordered Optimisation tests

Test #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Selective Consistency	×	×	×	×	×	×		×	×							
Skip Disjunctions	×	×	×	×			×			×	×	×				
Rank by Disjunction			×	×				×	×	×	×		×	×		
Rank by Term		×	×			×			×	×		×	×		×	

seconds was required load the ontology into the reasoner (not shown on graph).

Test a, with no optimisations (standard Tableaux algorithm) failed to complete due to insufficient memory. The same occurred for many of the tests which are not shown on the graph. This demonstrates that our strategies reduce memory consumption, making reasoning feasible on resource constrained devices. We note that in all tests, the Java virtual machine (JVM) used all of the memory allocated to it. Since the graphs in Figure 14 are difficult to interpret, we re-ordered (see table 5) the tests in an attempt to arrange the fastest processing times at the front of the graph. We show the re-ordered results in the graph in Figure 15.

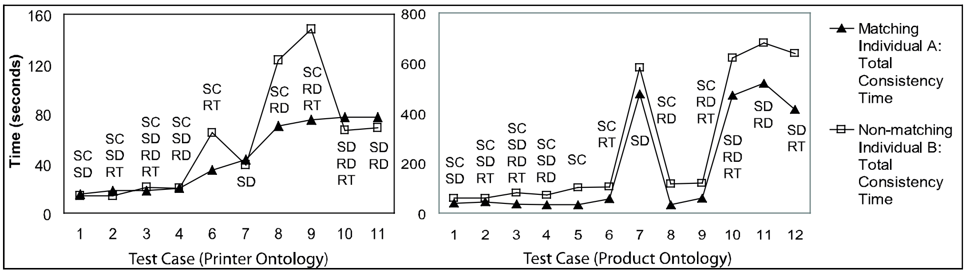
With optimisations enabled the best result for case study 1 and 2 was 18 and 35-70 seconds, respectively. This illustrates significant performance improvements in both scenarios.

When used in isolation, the selective consistency strategy proved to be the most effective in case study 2, while skip disjunctions was more effective in case study 1. Utilising both of these strategies together provided even better results, which suggests there is no advantage in selecting different strategies for different scenarios.

We found that the weighted strategies (rank by disjunctions and terms) did reduce the number of disjunction branches applied, by up to half in some cases, but this failed to significantly reduce the number of consistency rules applied overall. In addition, the ranking strategies did not improve performance when used in combination with the selective consistency and skip disjunction strategies. However, we observed that tests 13, 14, and 15, when matching individual A, in case study two, completed in 972, 982 and 983 seconds (not shown on graph), respectively, compared to 2139 seconds in test 16. This suggests that the rank disjunction and individual strategies improve performance but are far less effective than selective consistency or skip disjunction strategies. These ranking algorithms need to be improved in future work.

Due to the fact that our selective consistency and disjunction skipping strategies reduce

Figure 15. Re-ordered processing time required to perform each test, for Selective Consistency (SC), Skip Disjunction (SD), Rank by Disjunction (RD) and Rank by Term (RT) strategies, showing total consistency time to perform an inferred membership check for matching individual A and non-matching individual , for the Printer ontology (left) and Product ontology (right).



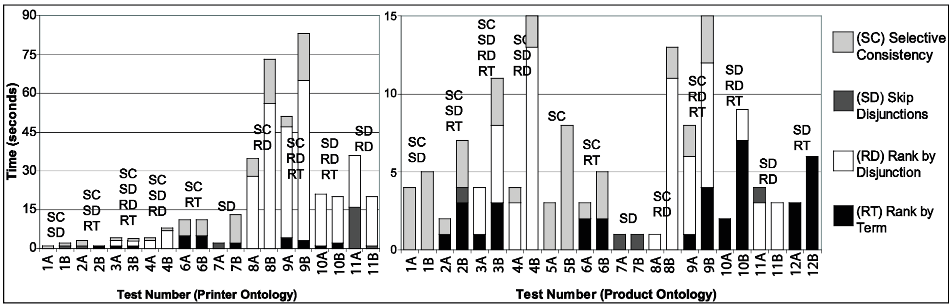
the number of potential rules and disjunctions to be applied, they improve performance in all cases. However, the results also showed that the optimisations can be less effective in improving performance for non-matching individuals B than with matching individuals A, as shown in every test in case study 2 and some in case study 1. This is because the Tableaux algorithm continues applying branches and consistency rules until a clash is found. This will inherently result in more rules to apply for non-matching individuals which do not clash for all branches. This finding also motivates the need for a resource-aware strategy, in which branches below a certain threshold are not applied, where resources are low, to assume no-match with some uncertainty rating.

Figure 16 illustrates the overhead cost incurred in executing the optimisation strategies for each test in from table 5, and shows the level to which each strategy contributes to the total overhead for the test. Each test is completed twice, for both matching individual A and non-matching individual B. We observed that skip disjunctions resulted in little to no overhead in all cases. Overhead costs for selective consistency was similar for both case studies, usually remaining under 5 seconds and peaking to 18 in tests 8B and 9B (test 8 and 9 for individual B) in case study 1, indicating a greater number of individuals to add to the weighted queue. Case study 1 recorded higher rank disjunction overhead than case study 2, suggesting there were fewer disjunctions and clash paths in the ontologies of case study 2, to evaluate. Rank disjunction overhead was also significantly higher for tests 8 and 9 for both case studies due to the skip disjunction strategy being disabled. It was also higher when type checking individual B compared to A, due to the reasoner exhaustively branching on disjunctions where a clash is never found.

In summary, we have demonstrated that:

1. mTableaux outperforms reasoners such as RacerPro and Pellet, performs comparatively with FaCT++ when full realisation is performed and faster than FaCT++ when it is not,
2. mTableaux does not compromise completeness as measured by recall and precision when all clashes are the direct consequence of the inference check rather than other unrelated concepts in the ontology as in realistic data sets such as those in our evaluation,
3. mTableaux minimises memory consumption such that successful completion of reasoning tasks on resource limited devices is possible,
4. mTableaux significantly reduces processing time compared with normal Tableaux with no optimisations,

Figure 16. Optimisation overhead breakdown. Each test was conducted twice, once for matching individual A and once for the non-matching individual B, for each case study (left graph: Printer, right graph: Product). EG 1A indicates test 1, individual A (see table 3).



5. selective consistency and skip disjunction strategies work best together while rank by disjunction and term strategies provided no added performance benefit,
6. the selective consistency strategy was more effective in case study 2 while skip disjunctions was more effective in case study 1, and provided the best results for both scenarios when used together, and
7. mTableaux strategies improved performance for both positive and negative type checks, however overall performance for negative type checks in case study 2 was poorer, leaving scope for resource-aware reasoning in future work.

CONCLUSION AND FUTURE WORK

We have presented a novel strategy for improving the scalability of the Tableaux algorithm for mobile semantic reasoning. mTableaux was shown to significantly reduce processing time and minimize memory consumption of pervasive discovery reasoning tasks in two case studies, so that they can be completed on small resource constrained devices. It was also shown to outperform RacerPro and Pellet without reducing the quality of results returned in realistic datasets such as in our scenarios. It also performed comparatively with FaCT++ when a full realisation was undertaken and outperformed FaCT++ when a realisation was not. The mTableaux strategies achieve this by limiting the number of branches and expansion rules applied and by applying the most important branches first to avoid the need for full branch saturation.

However, despite these significant optimisations, it is still possible that large ontologies may still exhaust all available memory before completing the task or require excessive amounts of time. In order to cater for time and memory constraints in situations where ontology or request size is too large even with the optimisation strategies enabled we are implementing the adaptive strategies briefly mentioned in section 4.2 which take available memory and time into consideration:

- The adaptive request condition matching strategy has the goal of matching first, the most important conditions in the request as deemed by the user, at the request level. The user is asked to specify weights of importance to each request condition. The most important conditions are matched first. In the event that important conditions do not match the reasoner will not continue to attempt to match less important conditions, if a threshold is exceeded. The threshold is determined based on the amount of time and memory available, under the assumption that limited processing power is better spent attempting to match another potential service.

- Our adaptive expansion rule application strategy utilises the weighted expansion rules from the weighted disjunctions and terms strategy in section 5.3. Similar to the strategy above, its goal is to stop the application of expansion rules which have a weight that falls below a certain threshold, except this occurs at the reasoner level. The threshold is increased when remaining time or memory becomes low.
- On-demand ontology loading has a goal of only loading of portions of the total ontology into the reasoner's memory. Reasoners such as Pellet, currently utilise an ontology parser and loader such as Jena (Jena, 2009) or OWL-API (WonderWeb, 2008) to load ontology files into memory. This data is then supplied in its entirety to the reasoner which creates classes, roles and individuals to represent all of this information as objects. Loading all of these parsed triples into the reasoner incurs significant initialisation costs and requires more processing time for lookup and retrieval during reasoning. In addition, if there is insufficient memory available to complete the reasoning task, the task fails even if most of the ontology data was irrelevant to the inference check. Unfder this on-demand loading strategy, rather than iterating all triples in the ontology to create objects in the reasoner, the reasoner instead queries the triples in order to create only the specific classes, roles or individuals which it requires during the reasoning process. That is if a URI of an individual is encountered by the Tableaux algorithm and no individual object is found within the reasoner to match the URI, it asks that the individual and the data associated with it is, be loaded into its knowledge base.

Our current work focuses on implementation and evaluation of these adaptive strategies to enhance the operation of mTableaux.

REFERENCES

Pellet. (2003). Retrieved from http://www.mindswap.org/2003/pellet/.

FaCT++. (2008). Retrieved May 1, 2007, from http://owl.man.ac.uk/factplusplus/.

KAON2. (2008). Retrieved June 21, 2007, from http://kaon2.semanticweb.org.

RacerPro. (2008). Retrieved May 23, 2007, from http://www.racer-systems.com.

Almeida, D. R. d., Bapista, C. d. S., Silva, E. R. d., Campelo, C. E. C., Figueiredo, H. F. d., & Lacerda, Y. A. (2006). A Context-Aware System Based on Service-Oriented Architecture. In *20th International Conference on Advanced Information Networking and Applications(AINA'06)* (pp. 205-210). IEEE Computer Society.

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

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D. & Patel-Schneider, P. F. (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., Domingue, J., Galizia, S., Gugliotta, A., Tanasescu, V., Pedrinaci, C. et al. (2006). IRS-III: A Broker for Semantic Web Services based Applications. In *5th International Semantic Web Conference (ISWC 2006)*, Athens, GA, USA.

Chakraborty, D., Joshi, A., Yesha, Y., & Finin, T. (2004). Towards Distributed Service Discovery in Pervasive Computing Environments. *IEEE Transactions on Mobile Computing*.

Chakraborty, D., Perich, F., Avancha, S. & Joshi, A. (2001). DReggie: Semantic Service Discovery for M-Commerce Applications. In *Workshop on Reliable and Secure Applications in Mobile Environment,* *In Conjunction with 20th Symposium on Reliable Distributed Systems (SRDS)*.

Chatti, M. A., Srirama, S., Kensche, D., & Cao, Y. (2006). Mobile Web Services for Collaborative Learning. In *4th International Workshop on Wireless, Mobile and Ubiquitous Technology in Education* (pp. 129-133). IEEE.

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

Guttman, E. (1999). Service Location Protocol : Automatic Discovery of IP Network Services. *IEEE Internet Computing*, *3*(4), 71-80.

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

Horrocks, I., & Sattler, U. (2005). A Tableaux Decision Procedure for SHOIQ. *19th International Conference on Artificial Intelligence (IJCAI 2005)*.

Howes, T. A., & Smith, M. C. (1995). *A Scalable, Deployable Directory Service Framework for the Internet*. Technical report, Center for Information Technology Integration, Univerity of Michigan.

Issarny, V., & Sailhan, F. (2005). Scalable Service Discovery for MANET. *Third IEEE International Conference on Pervasive Computing and Communications (PerCom)*, Kauai Island, Hawaii.

Jena - HP Semantic Framework. (2009). from http://www.hpl.hp.com/semweb/.

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

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

Lee, C., Helal, A., Desai, N., Verma, V., & Arslan, B. (2003). Konark: A System and Protocols for Device Independent, Peer-to-Peer Discovery and Delivery of Mobile Services. *IEEE Transactions on Systems, Man and Cybernetics*, *33*(6).

Miller, B. A., & Pascoe, R. A. (2000). Salutation Service Discovery in Pervasive Computing Environments. *IBM Pervasive Computing White Paper*.

Mysaifu, J.V.M. (2009). Retrieved from http://www2s.biglobe.ne.jp/~dat/java/project/jvm/index_en.html.

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

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., & Katz, Y. (2007). Pellet: A Practical OWL-DL Reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web, 5*(2).

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

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

Universal Description Discovery and Integration (UDDI). (2009). Retrieved from http://uddi.xml.org/.

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

OWL-API. (2008). Retrieved from http://owlapi.sourceforge.net/.

Web Service Modelling Ontology (WSMO) Working Group. (2009). Retrieved from http://www.wsmo.org/.

ENDNOTES

1 http://www.google.com/mobile

2 http://www.yahoo.com/mobile

3 http://www.cs.man.ac.uk/~horrocks/OWL/Ontologies/galen.owl

4 http://www.mindswap.org/ontologies/debugging/miniTambis.owl

5 http://protege.stanford.edu/plugins/owl/owl-library/koala.owl

6 http://www.mindswap.org/ontologies/team.owl

APPENDIX A

This section provides pseudo code detailing the functions referred to in section 5.3. Note that hasType(*I*, *C*) returns true if individual *I* has been assigned the class type *C*, and unfold(*C*) returns a set of all logic expressions and type names which type *C* is the equivalent of.

CheckPrimitive

```
Inputs: $I$, $C$, $CP$. Outputs: $CP$.
Let $I$ denote an individual.
Let $C$ denote a primitive class name or a literal value.
Let $CP$ denote a set (clash path).
Let $S$ denote a set $S = \{\}$.
If hasType($I$, $\neg C$):
    $CP \leftarrow I + CP$.
    Return $CP$.
Else:
    $S \leftarrow$ unfold($C$).
    Foreach $y_i$ in $S$:
        $CP \leftarrow$ ClashDetect($I$, $y_i$, $CP$).
        If $CP \neq$ null: Return $CP$.
    Return null.
```

CheckDisjunction

```
Inputs: $I$, $D$, $CP$. Outputs: $CP$.
Let $I$ denote an individual.
Let $D$ denote a disjunction.
Let $CP$ denote a set (clash path).
Let $S$ denote a set $S = \{\}$.
Let $e$ denote a disjunct element in $D$ where $D = \{e_1 \vee e_2 \vee_{\dots} \vee e_n\}$.
For each $e_i$ in $D$:
    $S \leftarrow$ ClashDetect($I$, $e_i$, $CP$).
    If $S$ = null: Return null.
    Else: $CP \leftarrow S + CP$.
Return $CP$.
```

CheckConjunction

```
Inputs: $I$, $C$, $CP$. Outputs: $CP$.
Let $I$ denote an individual.
Let $C$ denote a conjunction.
Let $CP$ denote a set (clash path).
Let $S$ denote a set $S = \{\}$.
Let $e$ denote a conjunct element in $C$ where $C = \{e_1 \wedge e_2 \wedge_{\dots} \wedge e_n\}$.
For each $e_i$ in $C$:
    $S \leftarrow$ ClashDetect($I$, $e_i$, $CP$).
    If $S \neq$ null:
        $CP \leftarrow S + CP$.
        Return CP.
    Return null.
```

CheckUniversalQuantifier

```
Inputs: I, av, CP.
Outputs: CP.
Let I denote an individual.
Let CP denote a set (clash path).
Let av denote a universal restriction expression, let avR denote the role to
which av applies to, let avC denote the role filler type defined in av for avR,
such that av=∀avR.avC.
Let $o_i$ denote an avR-neighbour to I.
Let O = {$o_1$, $o_2$, $o_n$}.
Let denote a set S = {}.
For each $o_i$ in O:
    S ← ClashDetect($O_i$, avC, CS).
    If S ≠ null:
        CP ← S + CP.
        Return CP.
    Return null.
```

CheckExistentialQuantifier

```
Inputs: I, sv, CP. Outputs: CP.
Let I denote an individual.
Let CP denote a set (clash path).
Let sv denote an existential quantifier restriction, let svR denote the role to
which sv applies to and let svC denote the role filler type for svR defined in sv
such that sv = ∃svR.svC.
Let mx denote a maximum cardinality role restriction, let mxN denote the cardi-
nality value defined in mx and let mxR denote the role to which mx applies to,
such that mx=(≤ mxR mxN).
Let $o_i$ denote an svR-neighbour to I.
Let O = {$o_1$, $o_2$, $o_n$}, where $o_i \neq o_{i+1..n}$.
Let $mx_i^{SVR}$ denote an mx which applies to the role svR.
Let MX = {$mx_1^{SVR}$, $mx_2^{SVR}$, $mx_m^{SVR}$}.
For each $o_i$ in O:
    If (svR is a functional role) AND (n ≥ 1 AND hasType($o_i$, ¬SVC)):
        Return CP + I + SV.
    Else:
        For each $mx_i^{SVR}$ in MX:
            If $mxN_i$ ≤ n + 1 AND hasType($o_i$, ¬SVC):
                Return CP + I + SV + MX.
```

CheckMaxRestriction

```
Inputs: I, mx, CP. Outputs: CP.
Let I denote an individual.
Let CP denote a set (clash path).
Let mx denote a maximum cardinality role restriction, let mxN denote the cardi-
nality value defined in mx and let mxR denote the role to which mx applies to,
such that mx=(≤ mxR mxN)
Let $o_i$ denote an mxR-neighbour to I.
Let O = {$o_1$, $o_2$, $o_n$}, where $o_i \neq o_{i+1..n}$.
If mxN < n:
    Return CP + I + mx.
```

Luke Albert Steller is a PhD candidate in the Faculty of Information Technology at Monash University. Luke's research is in the area of optimised and resource-aware semantic reasoning and pervasive service discovery.

Shonali Krishnaswamy is a Senior Lecturer in the Faculty of Information Technology at Monash University. Shonali's research is broadly the area of Distributed, Mobile and Pervasive Computing Systems where her focus on developing intelligent applications that aim to service real-time information needs while having to function in highly dynamic and resource-constrained environments. Her specific expertise is in Mobile and Ubiquitous Data Stream Mining, Service Oriented Computing and Mobile Software Agents.

Mohamed Medhat Gaber is a research fellow at Monash University, Australia. He has published more than 50 refereed articles and co-edited two books. Mohamed has served in the program committees of several international conferences and workshops. He received his PhD in 2006 from Monash University.