Chapter 16

Quantitative Analysis

Danilo Bonardi, Luís Botelho, Matthias Klusch, António L. Lopes, Thorsten Möller, Alexandre de Oliveira e Sousa, and Matteo Vasirani

16.1 Introduction

The different software agents and technologies that were introduced in earlier chapters of this book were also subject to a quantitative evaluation. The main objective of these tests was to verify that they can be effectively used in real world settings. Consequently, the measurements taken are mainly targeted to assess performance and scalability of CASCOM's meta services. Qualitative measures on the application level are not covered in this chapter and can be found in Chapter 15.

All tests that are subsequently described must be considered in the context of their network characteristics. Where possible, controlled environments and local services have been used, but in some cases it is opportune to use and rely on the standard Internet infrastructure respectively public services.

16.2 Service Matchmaker Agent

16.2.1 Test Environment

We have done four kinds of tests with the SMA. All the service descriptors were located on a local server at URJC. This was done so that the performance evaluation could not be affected by external factors like network delays or bandwidth.

Along all the tests, two different SMA configurations were evaluated. In the first configuration (in what follows, we will refer to it as cfg. 1), all the three matchmakers were invoked sequentially (first the role-based matchmaker, then OwlsMX and finally the precondiftion-effect matchmaker). This is the most fine-grained configuration of the SMA, but also the most time consuming, due to the presence of the logic reasoning matchmaker. In the second configuration (cfg.2), only the role-based matchmaker and OwlsMX were invoked sequentially.

$\mathbf{n}$	$\mathbf{t_r^1}$	$\mathbf{t_r^2}$
1	44735	8116
2	84023	11727
3	117200	14400
4	150816	14100
5	198588	17411
10	373209	25113
15	488944	39685
20	580619	44734
$\mathbf{n}$ concurrent requests $\mathbf{t_r^1}$ response time cfg.1 $\mathbf{t_r^2}$ response time cfg.2		

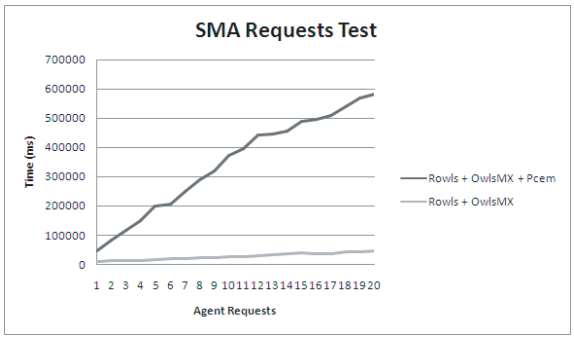

Figure 16.1: SMA response time as a function of concurrent requests

16.2.2 Test 1

The objective of the first test was to understand how the SMA reacts to an increasing number of requests, sent by several agents, with the same services and query combination. The services to match were nine services from the medical domain, with a query that match with all the descriptors.

The results are depicted in Figure 16.1. The plot shows that the response time grows linearly with the number of requests in both the two configurations, although in the second case the response time of SMA is considerably lower.

16.2.3 Test 2

The second test aimed at evaluating how the SMA manages an increasing number of services that match with the given query. The services and the query were the same of the previous test. In this test we used only one agent request.

The results are depicted in Figure 16.2. Also in this case the response time of SMA, with a growing number of matching services, increases almost linearly.

16.2.4 Test 3

The objective of the third test was to understand how the presence of matching and non-matching services affects the computation of the SMA. The request was composed by fifty non-matching services from the medical transportation domain, and an increasing number of matching services.

The result of the last test depicted in Figure 16.3 is aligned with the previous tests. It's worth noting that for the first configuration (cfg.1), even if the total number of services in the request is greater respect to the second test, the response time is similar. For example, in the second test, the response time of a request with 20 matching services was 71399ms, while in the third test, the response time of a

n	t_r^1	t_r^2
1	21514	4475
2	30438	7261
3	26773	5941
4	28662	7674
5	35828	6192
10	50039	8859
15	56260	12837
20	71399	12224
n matching services t_r^1 response time cfg.1 t_r^2 response time cfg.2		

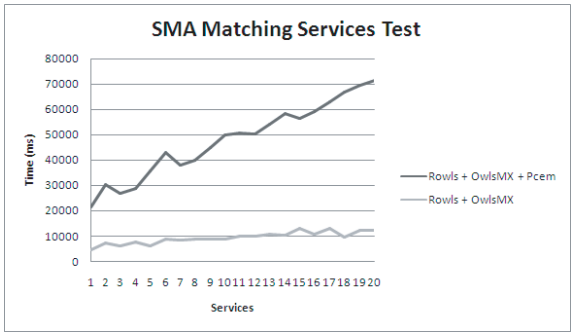

Figure 16.2: SMA response time as a function of matching services

n	t_r^1	t_r^2
1	34401	23889
2	38468	21961
3	45175	20380
4	48004	20099
5	46704	25124
10	73168	28135
15	77410	26077
20	88509	26221
n matching services t_r^1 response time cfg.1 t_r^2 response time cfg.2		

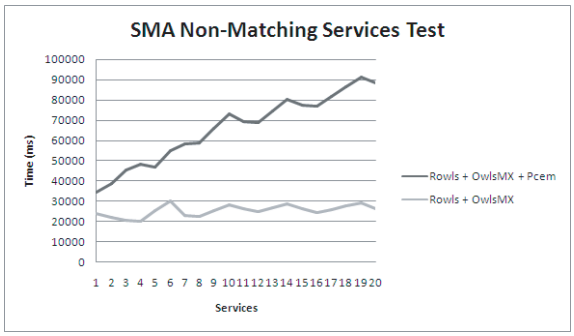

Figure 16.3: SMA response time with non-matching services

request with 20 matching services and 50 non-matching services was 88509ms. This shows that the SMA response time is more sensitive to the number of matching services rather then to the total number of services. This is because if one of the three matchmakers detects a non-matching service, this service is filtered out and not passed to the other matchmakers in the sequence.

16.2.5 Test 4

The objective of the last test is to check the retrieval performance of the matchmaker as a whole, that is, to determine the quality of the answer set of the five variants of the hybrid OWLS-MX matchmaker in terms of recall and precision (R/P). For details about the hybrid matching filters and text similarity measure from information retrieval (IR) used by each of the OWLS-MX variants and the

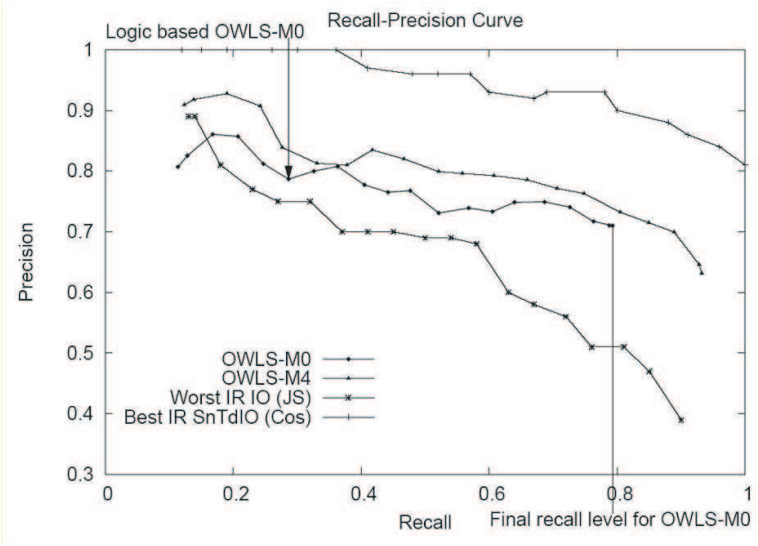

Figure 16.4: Retrieval performance of the matchmaker as a whole

R/P experiments conducted we refer to [2]. The R/P performance measurements were done against the OWL-S service retrieval test collection OWLS-TC2[1] that also includes the CASCOM services and adopted the evaluation strategy of micro-averaging the individual recall/precision (R/P) curves. The micro-averaged R/P curves of the top and worse performing IR similarity metric together with those for the OWLS-MX variants are shown in Figure 16.4.

This preliminary result provides strong evidence in favor of the proposition that building Semantic Web Service matchmakers purely on crisp logic based reasoning may be insufficient. A preliminary quantitative analysis of these results showed that even the best IR similarity metric (Cosine/TFIDF) alone performed close to the pure logic based OWLS-M0 which can be significantly outperformed by hybrid semantic matching with OWLS-M1 to OWLS-M4 in terms of both recall and precision. Second, the hybrid matchmakers OWLS-MX, in turn, can be outperformed by each of the selected syntactic IR similarity metrics to the extent additional parameters with natural language text content are considered.

[1]The OWLS-TC2 is available at `http://projects.semwebcentral.org/projects/owls-tc`

n	t_r^f	t_r^{nf}
1	44250	46125
5	33141	30078
10	44593	37203
15	76984	39844
20	81281	42864
25	89203	43453
30	84360	45123
35	85157	46983
40	94672	48223
45	111453	50241
50	126047	52123

n concurrent requests
t_r^f service found [ms]
t_r^{nf} service not found [ms]

Figure 16.5: SDA response time as a function of non-matching services

16.3 Service Discovery Agent

16.3.1 Test Environment

SDA has a lot of functionality, we decide to use only the find service feature because is related to OWL-S and it includes some algorithm complexity. We have done two kinds of test, that both comprise a grown number of requests to the SDA agent. In the first test, all the requests are a success; in the second, all the agents request a not found service.

16.3.2 Test Results and Discussion

The results of the test depicted in Figure 16.5 evidence a consistent difference between the two kinds of request. The results also show that the find service feature of the SDA is quick, also consider that in our test we bypass the SDA cache.

If we analyze the detail of each request done to the SDA, we can determine what the most time-consuming tasks are in the overall process. Figure 16.6 shows a pie chart with the sub-tasks that have to be performed by each search requestq done to the SDA.

The results depicted in Figure 16.6 were obtained by the average result of 200 search requests done to the SDA, which had, at the time, 200 services registered in

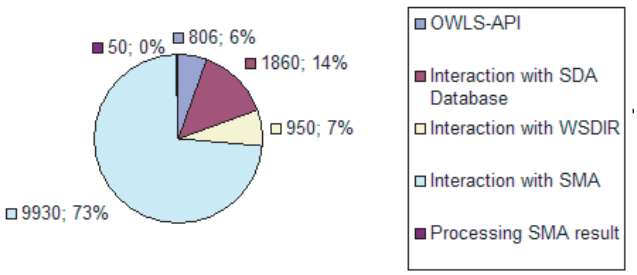

Figure 16.6: SDA search time [ms] percentages

the service directories. As can be seen from the picture, the most time-consuming task (around 73%) is in fact the matching process that is carried out together with the SMA. The remaining time (around 27%) is consumed by the operations related to the acquisition of services from the service directories and processing the acquired service descriptions using the OWL-S API.

16.4 Service Composition Planner Agent

As mentioned in Chapter 11, two different Service Composition Agents (SCPA) have been developed in CASCOM that differ in the planner engine used: one is based on XPlan [4] while the other relies on SAPA [1]. In any case, the chosen CASCOM SCPA takes a set of OWL-S services, a description of the initial state and the goal state to be achieved as input, and returns a plan that corresponds to a composite service that gets invoked using the FIPA-Request interaction protocol.

16.4.1 OWLS-XPlan

We evaluated the performance of XPlan, using the publicly available benchmark of the international planning competition IPC3, and compared the results with that of the four top performing IPC3 participants, i.e. FF planner, Sim-Planner, and the HTN planners TLPlan, and Shop2. XPlan was tested without task specific methods. Planning performance was measured in terms of (a) the planning completeness, i.e. the total percentage of solved problems, (b) the average plan length, and (c) the average plan quality, i.e. the average distance of individual plans from the optimal plan length in relation to the complexity of the given problems2. In summary, the experimental performance results of both versions of OWLSXPlan show that service composition planning can be done reasonably efficient and effective for low to medium complex planning domains (up to 15k objects), in particular

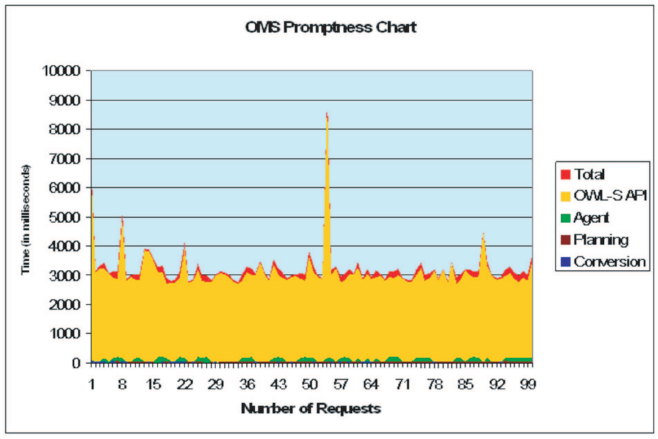

Figure 16.7: Service Composition in the OMS Domain for 100 requests.

the limited e-health planning domain considered in the project CASCOM during the field trial.

For more information and preliminary evaluation of XPlan 2.0, we refer to [4]. Evaluation results for the static version OWLS-XPlan 1.0 are provided in [3].

16.4.2 MetaComp

The alternative composition planner that can be used by the CASCOM SCPA, that is SAPA as part of the module MetaComp (cf. Chapter 11), was evaluated with respect to domain independence, performance and scalability. The evaluation tests were carried out using an Intel(R) Pentium(R) M processor 1.60 GHz, equipped with 512 MB of RAM memory. Since the designed service selection methods have not been integrated in MetaComp, the evaluation tests address only the agent remaining features.

The domain independence hypothesis was evaluated through testing the agent in different application domains namely the medical records translation domain (Medical Records), the online medicine selling domain, the box depots transport (Depots) domain, the package delivery domain (DriverLog), and the aero travel domain (ZenoTravel). The agent successfully generated compound services in all of these domains. Although this does not prove that MetaComp is completely domain independent, it constitutes significant evidence supporting the domain independence hypothesis.

MetaComp performance and scalability were assessed in the Online Medicine Selling Domain (OMS). Of all the domains we have used in our tests, the OMS domain was chosen because it is the only one for which OWL-S descriptions of its services exist, allowing us to estimate the complete execution time, including the

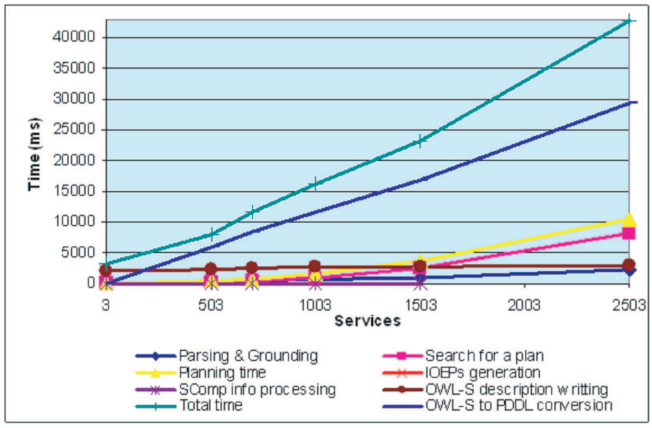

Figure 16.8: Average service composition times vs. number of available services

OWL-S to PDDL conversion. Figure 16.7 shows the performance results in the OMS domain, with 100 requests and no additional services besides those required to achieve the composition goal.

The total execution time per request was about 3.4 seconds. The average planning was 26.95 ms per request, which is practically imperceptible for the end user. Another component that is also hardly noticeable by the user (in this test) is the OWLS2PDDL converter, due to the reduced number of services to convert. OWL-S API is responsible for about 95 percentage of the total execution time.

We have increased the number of services considered for composition to assess the agent scalability. Fifty compositions for 503, 703, 1503 and 2503 services were carried out in the OMS domain. The average times are presented in Figure 16.8. For 503 services, the average time spent in composition was 8 seconds. For 703 services, this time was 12 seconds. For 1503, the average total time was 24 seconds. Finally, for 2503 services, the average time spent in composition increased to 43 seconds.

These results show that MetaComp performs within a reasonable time frame while the number of services considered for composition is less than 500 services. Unfortunately, when this number increases, performance rapidly worsens to impractical values. The times spent writing the OWL-S description of the compound service, and generating its inputs, outputs, preconditions and effects do not change because the solution plan is always the same. As the number of available services for composition increases, the planning time also increases, becoming larger than the time for writing the OWL-S description. Given the enormous growth of the number of considered services, the largest increase happens in the OWL-S to PDDL conversion time.

In summary, the results show the excellent performance of SAPA as one

alternative planning component of the CASCOM SCPA. For compound services involving a small number of elemental services (which we assume to be the case for most service composition environments), the generation of the OWL-S description of the compound service from the output of the planning component consumes about 95 percentage of the total time spent to handle each composition request. This quite unexpected result is due to using the OWLS-API, which is still under considerable development. It is assumed that this may be improved in the near future.

16.5 Service Execution Agent

16.5.1 Test Environment

Probably the most interesting measure to explore the performance of the Service Execution Agent (SEA) is the time it takes to execute a given service as a function of concurrent requests (using alternating input data). Consequently, this test has been conducted but in two different cases: one where the service is correctly executed and the other where the service execution has some(random) run time error in input. In these two tests, the real existing Healt@Net Web Service *Find Patient* has been used, annotated by a locally stored OWL-S descriptor. This means that the test environment can not be taken as entirely controlled since it uses — and relies on — the standard Internet infrastructure and its properties. However, this decision was made on purpose to include the dynamics of the Internet and to get insights on how this might effect the test in terms of time variations.

16.5.2 Test Results and Discussion

As it can be seen from Figure 16.9, the SEA is initially fast for both cases. Nevertheless, the plot evidences a continuous raise of the execution time with increasing number of concurrent requests higher than what was expected. As detailed analysis revealed, there is one main reasons for this, which goes back to implementation matters: The current implementation of the FIPA interaction protocol part of the JADE platform – used as a basis by the SEA – works as a barrier that serializes incoming parallel requests, that is, incoming requests are put into a queue and each request is handled one after the other. This means that the SEA currently conducts just one execution at a time although its internal structure behind this barrier especially was designed and implemented to support concurrent (composite) service executions. Only by a extension of this behavior in the near future the SEA can demonstrate its full potential in terms of responsiveness on increasing load.

The observation that the execution in case of input failure is faster than the normal one is no surprise and a result of the fact that in this case the invoked Web Service produces a fault, that is, it does not run the functionality that it

n	t_r^w	t_r^f
1	14688	12703
5	54594	38828
10	92390	64031
15	129422	92484
20	170875	127172
25	224891	179266
30	263907	190141
35	369875	206688
40	399031	254610
45	422203	302968
50	451468	312734

n concurrent requests
$\mathbf{t_r^w}$ service working [ms]
$\mathbf{t_r^f}$ service failure [ms]

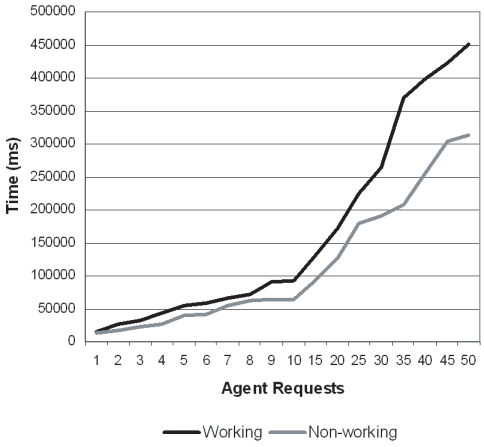

Figure 16.9: SEA response time as a function of concurrent requests and service failures

represents.

Because of the fact that the standard Internet infrastructure was part of the test the results must be considered in the context of its network and utilization characteristics. The plot in Figure 16.9 does not show strong outliers. This indicates that both the Internet infrastructure as well as the invoked service offered nearly constant runtime characteristics at the time when the test was conducted.

16.6 WSDir

16.6.1 Test Environment

The objective of the test environment was to compare the response time of the directory federation towards multiple simultaneous search requests (only abstract search) coming from an increasing number of clients and registered services descriptions. For that, we have decomposed the testing of WSDir into three scenarios. Based on the topology used in CASCOM, the number of directory services were different for each scenarios. Actually, each time, a set of service descriptions is stored in the federation (evenly balanced) and the set of clients accesses as much as possible different directory services in the Top Network layer. All the directory services were partially distributed (some were running on the same servers) in a secured network. The clients were all running at the same time in threads on the same computer. Each client performed three random searches to produce different results.

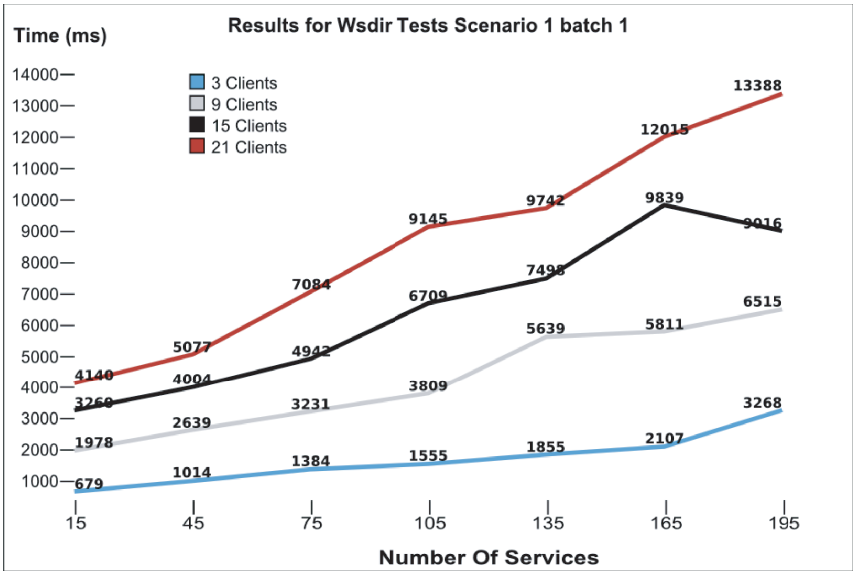

Figure 16.10: WSDIR Average search request processing time per number of services for scenario 1

16.6.2 Topology and Scenario

In CASCOM, WSDIR is distributed on three network layers : a Hidden Layer, a Top Layer and a Body Layer. Agents only access the directory services located on the Top Layer. Each scenario has a specific number of directory services in the Top Layer and the Body Layer (the Hidden Layer always contains one directory service). The three used scenarios were the following:

1. Scenario 1 : three directory services on the Top Layer and six on the Body Layer.
2. Scenario 2 : six directory services on the Top Layer and six on the body layer.
3. Scenario 3 : ten directory services on the Top Layer and twenty on the Body Layer.

16.6.3 Test Results and Discussion

The following figures show the average response time in millisecond per number of stored services. Each line correspond to a given number of simultaneous clients requesting the federation.

Note that scenario 2 (see Figure 16.11) shows the best results. This implies that Top Layer's directory services play an essential role in the scalability of the system. This observation can be explained by the following reason: not enough

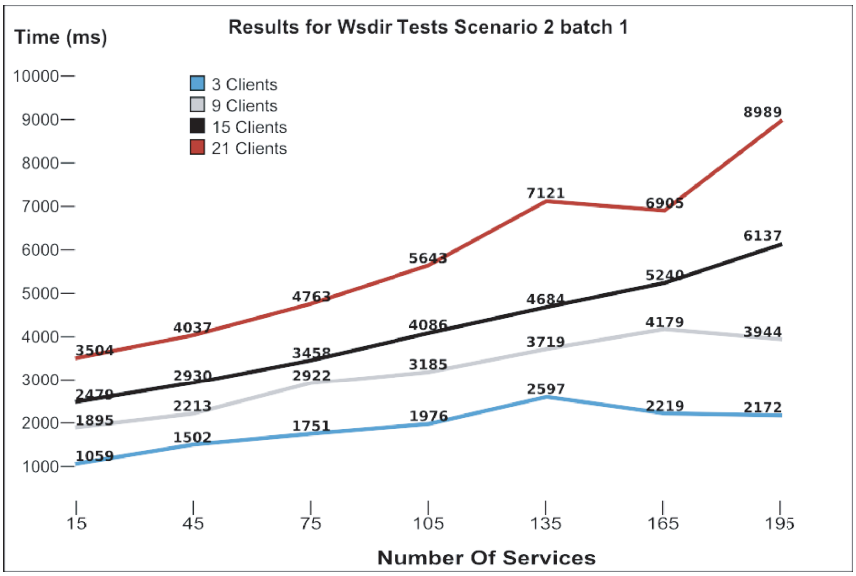

Figure 16.11: WSDIR Average search request processing time per number of services for scenario 2

directory services on the Top Layer serve badly too many clients and too many directory services on the top layer produce too many messages within the federation, loosing more time on intra-communication. For a given number of clients, we see that the time increases linearly. The abstract search algorithm is in $O(n)$ complexity. The variations on the lines are due to run time performance clean up of the directory services and random queries (a query that matches a lot of services takes more time to be processed).

References

[1] M. B. Do and S. Kambhampati. Sapa: A Scalable Multi-objective Heuristic Metric Temporal Planner. *Journal of AI Research*, 20:155–194, 2003.

[2] M. Klusch, B. Fries, and K. Sycara. Automated Semantic Web Service Discovery with OWLS-MX. In *Proceedings of the 5th Int'l Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Hakodate, Japan, 2006. ACM Press.

[3] M. Klusch, A. Gerber, and M. Schmidt. Semantic Web Service Composition Planning with OWLS-XPlan. In *Proceedings of the AAAI Fall Symposium on Semantic Web and Agents*, Arlington VA, USA, 2005. AAAI Press.

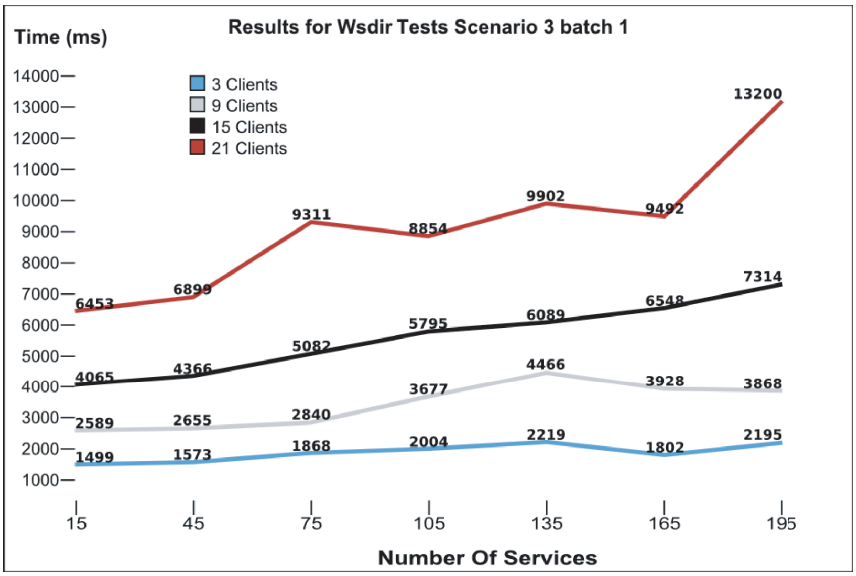

Figure 16.12: WSDIR Average search request processing time per number of services for scenario 3

[4] M. Klusch and K-U. Renner. Fast Dynamic Re-Planning of Composite OWL-S Services. In *Proceedings of 2nd IEEE Intl Workshop on Service Composition (SerComp)*, Hongkong, China, 2006. IEEE CS Press.