Ontology-driven Dynamic Discovery and Distributed Coordination of a Robot Swarm

Niels Bouten, Anna Hristoskova,
Femke Ongenae, Jelle Nelis, and Filip De Turck

Ghent University - Department of Information Technology - IBBT
Gaston Crommenlaan 8/201, B-9050 Ghent, Belgium
{niels.bouten,anna.hristoskova,femke.ongenae,
jelle.nelis,filip.deturck}@intec.ugent.be
http://ibcn.intec.ugent.be

Abstract. Swarm robotic systems rely heavily on dynamic interactions to provide interoperability between the different autonomous robots. In current systems, interactions between robots are programmed into the applications controlling them. Incorporating service discovery into these applications allows the robots to dynamically discover other devices. However, since most of these mechanisms use syntax-based matching, the robots cannot reason about the offered functionality. Moreover, as contextual information is often not included in the matching process, it is impossible for robots to select the most suitable device under the current context. This paper aims to tackle these issues by proposing a framework for semantic service discovery in a dynamically changing environment. A semantic layer was added to an existing discovery protocol, offering a semantic interface. Using this framework, services can be searched based on what they offer, with services best suiting the current context yielding the highest matching scores.

Keywords: Service Discovery, Semantics, Context-awareness, Distributed Planning, Swarm Robotics

1 Introduction

Heterogeneous system designers have to cope with the lack of standardisation that exists between different devices. Programmers are therefore often obliged to incorporate statically programmed interactions, deteriorating the overall flexibility. Service-Oriented Architectures (SOA) [5] are a popular approach towards attaining higher versatility and flexibility in networked environments. Devices offer their services over the network, allowing their functionality to be easily discovered using a service discovery protocol.

Most discovery mechanisms however, perform service matching solely based on the syntax resemblance between requested and offered descriptions [15]. This often leads to poor results, since the requested description can be semantically similar but syntactically different from the offered descriptions (e.g., *go* and *move* *to* which are synonyms with different syntax), or syntactically similar but with a different meaning (e.g., *object* meaning a goal and *object* meaning a thing). Another drawback is of course that syntax matching does not consider relations between the different concepts in the descriptions. Semantic descriptions overcome these shortcomings by using ontologies to capture semantics of and relations between the different concepts.

Applying conventional discovery mechanisms in a robotic setting requires to overcome some challenges imposed by the specific properties of mobile robots. Using a central repository like most discovery mechanisms do, is not viable in a swarm robotics application since there is no guarantee that the repository will remain available due to the mobility properties. Moreover, since robots only have limited resources and the matchmaking of semantic services scales exponential with the number of instances, a robot is not able to process all incoming matching requests in reasonable time when serving as a central repository [9].

Another shortcoming of existing mechanisms, is the absence of context evaluation during the matching process, i.e. current location, consumed resources and current tasks. A robot requesting help on a certain location, will prefer a robot which current location is closer to the goal to execute the task. Robots acting as service providers also need to deal with their limited resources when offering services. Based on the current status, a robot could decide to no longer offer a specific service since it would overload the system. Taking into account the current context of both the requesting robot and the service provider during the matching process would enable more accurate and resource efficient service discovery.

This paper suggests a different approach by letting the provider of the services match the incoming requests to their offered services. In this way, each device only has to match a limited number of services with low delay. By offering matching as a service, the number of semantic descriptions exchanged between devices is considerably smaller than when each device collects all descriptions of every other device in the swarm. Furthermore, context information, such as the current location, is easily injected into the matching process, without the need to disseminate all context information of each service provider.

The outline of this paper is as follows: first, a brief overview of related work is presented in Section 2. In Section 3, the proposed framework for ontology-driven discovery and coordination of a robot swarm is discussed. Section 4 describes the details concerning the implementation of the aforementioned framework. The evaluation results are presented in Section 5. Section 6 concludes this paper.

2 Related Work

Ontologies are used to incorporate semantics in service descriptions, by modelling the domain knowledge in terms of concepts and relationships between them [7]. Semantic Web services are often used to overcome the interoperability issues between different robot platforms. Several frameworks exist for defining, matching, invoking and monitoring of services [11–13]. However, these platforms are focussed on Web services offered by Web servers and consumed by desktop and laptop computers over the Internet. To apply these frameworks in a robotic setting several issues need to be overcome, such as the local context of both the service requestor and provider, the limited resources available on mobile devices and the inherent distributed nature of swarm robotic systems.

To overcome the high delays caused by the computationally intensive semantic matching for central repositories, the use of semantic caches was proposed by Stollberg et al. [20]. This is a feasible solution when dealing with static environments where little context information has to be processed during the matching of services. In a robotic setting however, where rapid context changes are common, the caching of semantic requests and their respective matching results would lead to less accurate and even unusable results. VOLARE [18] proposes a Service-Oriented Architecture for mobile devices, taking into account the current context. This solution is mainly focussed on making services located on a Web server available to mobile devices. EASY [16] tries to reconcile the computationally expensive semantic matching and the limited resources of robots and embedded systems by encoding semantic descriptions and organising them into service caches. This allows faster semantic matching without overloading the devices. As mentioned before, the use of caching causes difficulties to incorporate the current context into the matching process.

Other projects focus more on the semantic discovery aspect in swarm robotic applications. In Geminga [1], a robot periodically announces its available services which are stored in a local repository located on each robot. Every robot has to match his service request locally with the services in the repository. When context information needs to be included in the matching process, the current context of each robot has to be obtained, causing delays proportional to the number of robots in the swarm in both the service matching and the context retrieval process. ROBOSWARM [23] makes use of a central mediator [8], responsible for maintaining the service repository and serving matching requests. S-Ariadne [16] uses a set of repositories located in a P2P overlay network. This speeds up the matching process, but still requires context dissemination throughout the swarm.

Other technologies focus more on semantic service composition [6, 22], composing new complex services out of existing services. In most of these projects, the emphasis lies on the generation of the semantic descriptions for these services by matching the outputs of a service to the offered inputs of the services. In this paper, existing composite service descriptions are used, but the binding of a service type to a specific service is done dynamically, taking into account the current context.

3 Framework Design Details

The proposed framework, presented in Figure 1, consists out of a *Service Manager*, a *Context Manager* and a *Robot Control* component. The services offered by the robot are accessible through the *Service Manager*, which is responsible for matching incoming requests with the local repository and monitoring service invocations. The *Robot Control* component executes the invoked services and is able to find, match and execute remote services through the *Service Manager*. The *Robot Control* component periodically reports contextual information to the *Context Manager*. This component keeps track of the current context and semantically annotates the received data, which can then be used by the *Service Manager* to evaluate matching results under the current context.

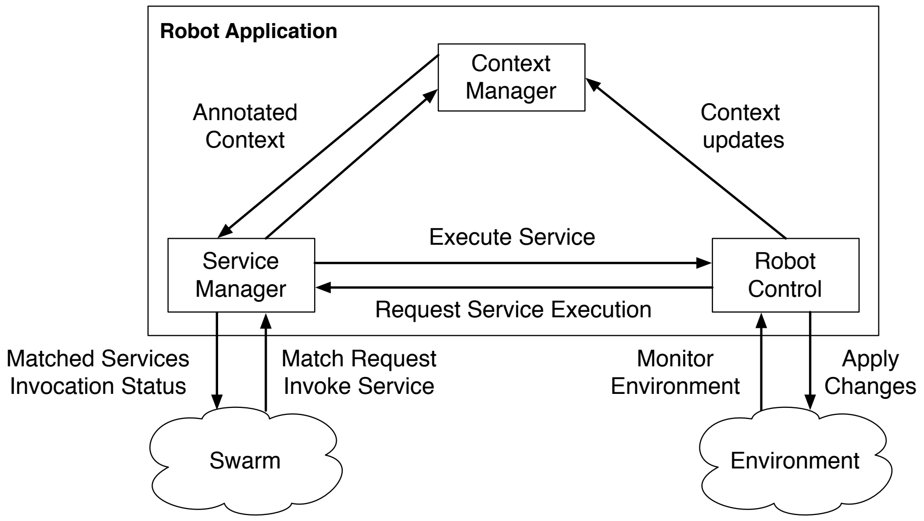

Fig. 1. Framework overview presenting the mechanism for offering the robot functionality (*Robot Control*) as services to the other robots in the swarm through the *Service Manager* and the monitoring of the current environment state by the *Context Manger*.

In order to attain the required degree of flexibility and interoperability, each robot offers its functionality as a set of services which are discoverable by its peers. Each of these services is stored in a local repository, accompanied by its corresponding semantic description containing the semantic representation of the services' *Inputs*, *Outputs*, *Preconditions* and *Effects*. Figure 2 gives an overview of the interactions between the different components of the *Service Manager*. The *Service Repository*, keeps track of the availability of each atomic service as well as the composite services. Additionally to these robot-specific services, each of the devices also needs to offer a common *MatchMaker* service, which is responsible for matching incoming service requests and local context injection (2). A conventional *Service Discovery* mechanism is used to discover the *MatchMaker* services of other devices, which can then be invoked to perform semantic matching with the services in the local repository of that device (3). Each of the discovered robots will match the requested description with the semantic descriptions available in the local repository taking into account the current context (4) and return the results, if any (5). The matching algorithm is explained in more detail in Section 4.3. The requesting robot can then choose the best suited service based on the returned matching results. The inputs are then transformed in order to match the ones stated in the semantic description

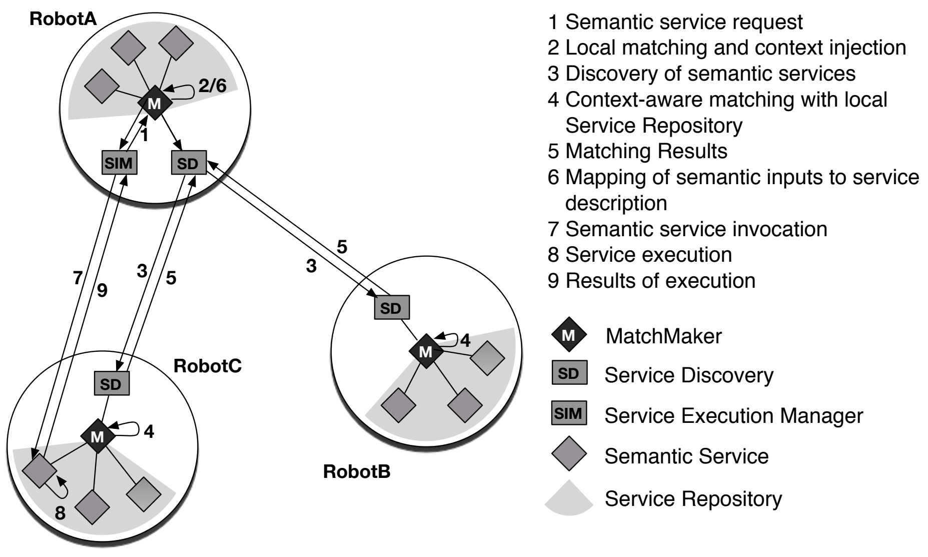
RobotA
RobotB
RobotC
M
SIM
SD
1
2/6
3
4
5
7
8
9
1 Semantic service request
2 Local matching and context injection
3 Discovery of semantic services
4 Context-aware matching with local Service Repository
5 Matching Results
6 Mapping of semantic inputs to service description
7 Semantic service invocation
8 Service execution
9 Results of execution
MatchMaker
Service Discovery
Service Execution Manager
Semantic Service
Service Repository

Fig. 2. Interactions during the semantic service discovery process

of the offered service (6). After this, the service is invoked (7) and monitored (9) by the *Execution Manager*.

4 Implementation Details

4.1 Robot Ontology

Since the proof of concept scenario, discussed in more detail in Section 5.1, takes place in an environment with a heterogeneous robot swarm and a multiplicity of networked devices, a specialised ontology is constructed. Bearing in mind the rapid innovation in the field of robotic devices, the ontology is designed to be easily expandable while taking into account the limited resources robots have to reason about semantics. Figure 3 shows how a distinction is made between physical entities and general properties. The physical entities are then split up into *Components* and *Devices*. This classification is based on the fact that *Devices* can execute certain tasks, while *Components* can not do this without being part of a *Device*. The class *Component* has two subclasses: *Actuators* grouping components able to make changes to the environment and *Sensors* who are able to measure changes or properties of the environment. The *Properties* describe physical attributes such as *Location* and *Memory*. In order to model the current context, each individual will have some properties containing information about its state, such as a *MobileRobot* having a *Location*-property.

4.2 Service Description, Discovery and Invocation

UPnP [21] is chosen as the service discovery protocol, specifically the Cling protocol stack [2] is used. UPnP offers automatic service discovery, allowing for

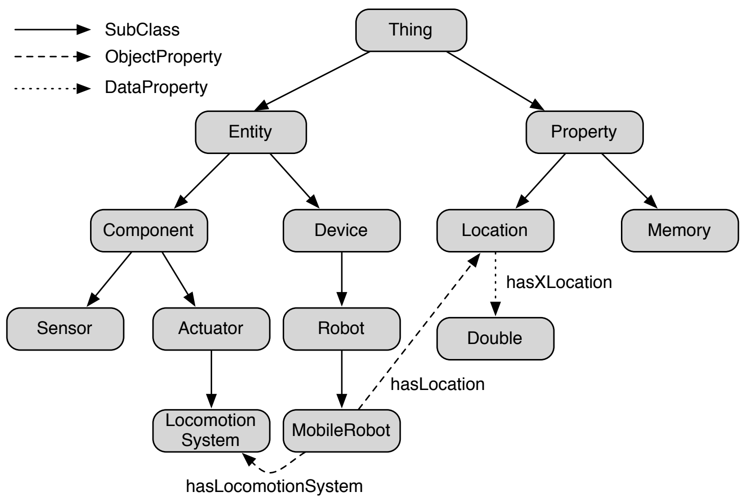

Fig. 3. Fragment of the robot ontology displaying a *MobileRobot* containing a *LocomotionSystem* modelled by a *ObjectProperty*-relation and the instantiation of a *X-Coordinate* of the *Location-Property* by a *DataProperty*-relation.

devices to dynamically join and leave the swarm. Moreover it supports monitoring of the execution process of a service via an eventing mechanism. A common way to describe semantic information is by using an ontology, representing objects and their relations. For the construction of the swarm ontology, OWL [17] is chosen as semantic language. This decision is made after reviewing the offered functionality of OWL-S [14], a service ontology based on OWL. OWL-S can be used to semantically describe the inputs, outputs, preconditions and effects of services and reason about them. Originally OWL-S is aimed at Web Services only, but little effort is necessary to extend the grounding (the link between the semantic description and the WSDL description) to be interoperable with UPnP and other discovery protocols.

4.3 Service Matchmaking Algorithm

The matchmaking algorithm is used to link a semantic service request to the each service in the *Repository*, by mapping their respective Inputs, Outputs, Preconditions and Effects (IOPE's) and calculating a score representing their degree of resemblance. The implemented matching algorithm uses bipartite graph matching [3] to determine the degree of interoperability between the service request and the offered services in the local repository. Some modifications had to be made to the original algorithm to include the current context into the matchmaking process. Preconditions and effects are split up into context-dependent (i.e. when the evaluation of the precondition is affected by the current context) and context-independent preconditions, this is done by the creator of a service. The parameters of context-dependent preconditions and effects are matched to the semantic representations of the current context using the same bipartite graph matching algorithm as for inputs and outputs. Those matched parameters are then substituted by their respective current context-values and the obtained expression is evaluated. Depending on the condition, this leads either to a boolean or a numeric value, which is included in the final matching report of the service. For example the evaluation of the distance to a given target (i.e. expressed as a *Location*) returns a numeric value, while the condition that a certain locomotion system (e.g., a *LocomotionSystem*) has to be present returns a boolean value.

The bipartite graph matching algorithm matches each concept of the IOPE's of the requested description to those of the most suited concept of the offered service description. For example, for the output matching, this is done by creating a bipartite graph $G = (R, O, E)$, where R is the set of requested outputs, O the set of offered outputs and E the set of edges such that each $e \in E$ has one vertex in R and one in O. A matching of a graph G is a subgraph $G' = (R, O, E')$ such that no two edges $e_1, e_2 \in E'$ share the same vertex. A matching G' is complete if each vertex $r \in R$ is adjacent to exactly one $o \in O$. The corresponding edge e' connecting r to o is allocated a weight w' describing their degree of match:

Exact If r is an equivalent concept to o.
Plugin If r is a superclass of o, or r subsumes o.
Subsume If o subsumes r.
Fail If none of the above conditions hold.

Exact matches yield the highest scores, while for *Plugin* and *Subsume* matches the score is dependent on the number of levels between both semantic concepts by turning the parameter levelscoring on. Each graph $G' = (R, O, E')$ can then be evaluated based on the weights of the different edges. The edge with the minimum weight $min(w') : e' \in E'$ determines the overall degree of match x for each graph G'. The graph having the highest value for x is the best match, which is returned. Each of the matching scores for the IOPE's, along with the scores for the context evaluation are returned to the requesting application.

5 Experimental Results

5.1 Application of Swarm Robotics for Search and Rescue

A search and rescue scenario is adopted during the functional testing where a swarm of robots is responsible for searching survivors in a burning building. The robots differ in functionality: some are able to detect human beings, others are capable of shooting a video on a specific location. Additionally to the robot swarm, a collection of networked devices such as cameras, TV's and laptops are at hand. The goal of the swarm is to cooperatively locate the survivors and create an impression of their situation. The scenario is demonstrated using iRobot Roombas [10] controlled by a ALIX2D3 computer, connected via the serial interface of the Roombas and equipped with a wireless antenna. A camera is mounted on top of some of these robots.

The functional tests are then split up into several scenarios. The semantic context-aware matching and selection is demonstrated by letting three devices offer somewhat similar services. Two devices offer a service that matches the request but one of them is closer to the goal location, yielding a higher score for the context evaluation. The selection algorithm evaluates these matches and chooses the robot closest to the goal to execute the service. A second scenario tests the execution monitoring and the selection of a backup service. The robot closest to the goal location is selected to execute the service, but during execution some problem occurs, causing the *Execution Manager* to detect a failed execution. In reaction to that, the second service is selected and executed, showing the desired backup behaviour. The third scenario emulates the search and rescue use case where a swarm of robots is ordered to scan and report a burning building. This composite service (i.e. consisting out of 3 services: locating the survivors, shooting a video on these locations and streaming to the firemen outside) is offered by the respective robots. The selected robot searches for suitable robots or devices to fulfil each of these subtasks and monitors their executions. For each of the services, robots and devices are ranked based on their current context and the best suited robot is chosen (i.e. the camera robot closest to a certain survivor is selected to go and monitor the situation).

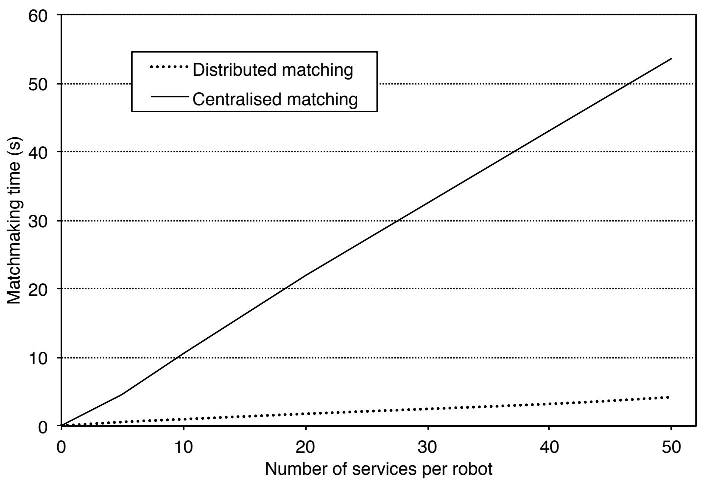

Fig. 4. Performance comparison of centralised versus distributed matchmaking

5.2 Centralised Versus Distributed Matching

Several tests are conducted measuring the corresponding matchmaking time relative to the number of services in the repository. As shown in Figure 4, the matchmaking time shows a linear relationship with the repository size. The graph also displays the difference in matchmaking time between the centralised and the distributed approach. For the centralised matchmaker, all services of all robots are stored on one single robot. This leads to a matchmaking delay proportional to the number of robots in the swarm. With 10 robots, each offering 10 services, the matchmaking in the centralised approach takes 10 times as much time as when each robot does local matchmaking on its own services. These tests are executed under several assumptions, where in the centralised approach, the robot already has the complete semantic repository and the context information of each robot. In a realistic scenario the dissemination of all context information of each robot to the central robot would cause extra delay, and even increase the total delay incurred in the centralised approach. These high delays can lead to inaccurate matching since the context could already have changed when the matching results become available.

5.3 Impact of Included Features on Matchmaking Performance and Accuracy

The OWLS TC benchmark [19] is used to measure the influence of the features (i.e. levelscoring, profilechecking, precondition and effects checking) of the *Matchmaker* on the matchmaking time and the accuracy of the corresponding results. This benchmark only includes inputs and outputs in the matching process. For this reason, only the levelscoring can be evaluated for both the accuracy and performance measurements. For evaluating the influence of the respective features on the performance, the average matching time for a set of 27 requests on a repository of 1000 services is measured.

The accuracy of the *Matchmaker* is evaluated by counting the number of *true positives* (tp) and *true negatives* (tn), summing up to the number of correctly classified matches, and the number of *false positives* (fp) and *false negatives* (fn), totalling the number of misclassified matches. The precision and recall values were then calculated as $\frac{tp}{tp+fp}$ and $\frac{tp}{tp+fn}$ respectively. A high precision value is desired, since only suited services will be useful to be executed. The F_β-score, calculated as in (1), is the harmonic mean of precision and recall. A smaller value for β puts more emphasis on the precision than on recall. The influence of each parameter on the performance is displayed in Figure 5, while the effect of scoring according to the degree of matching on the accuracy is shown in Figure 6.

$$(1+\beta^2) * \frac{precision * recall}{\beta^2 * precision + recall} \tag{1}$$

Level scoring By turning this parameter on, not only the fact that the requested parameter type is a sub or super type of the offered parameter type is taken into account, also the distance between these two types is measured to calculate the corresponding score. Turning level scoring on reduces the number of false positive matches from 24% to 8% and the number of false negative matches from 32% to 12%, increasing recall from 62% to 70% and precision from 81% to 88%. The F_1 and $F_{0.5}$ values increase from 70% to 78% and from 77% to 84% respectively. This increased accuracy comes at a cost of a 16% higher matching delay.

Profile checking It is possible to filter services based on their profile definition. A hierarchy is created to model relations between several profiles. For this test, a distinction is made between *RoboticServices* and *ComputerServices*. When searching for a mobile robot for the execution of some task, the services catalogued as *ComputerServices* are excluded. Turning the profile checking on reduces the matching time by 30%.

Precondition and effects checking Turning these parameters on, the matchmaker will evaluate preconditions and effects using the current context. This yields more precise context-based matching in exchange for matching delays that are 5% higher.

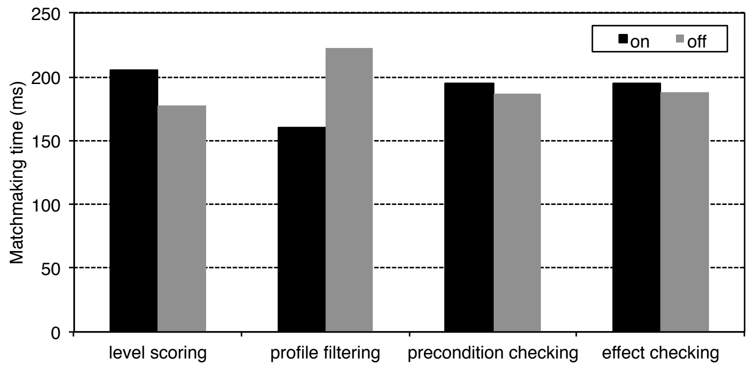

Fig. 5. Impact of selected features included during matchmaking on matching time

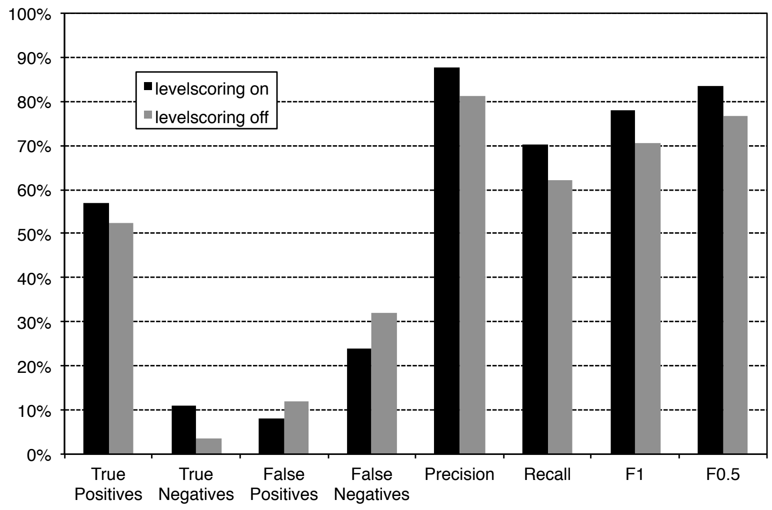

Fig. 6. Impact of selected features included during matchmaking on matching accuracy

6 Conclusions and Future Work

In this article, the viability of ontology-driven and context-aware discovery in a swarm robotics setting is studied in detail. Using semantics to annotate the service descriptions as well as the current context yields more accurate service matching, resulting in a more efficient use of the available services. Using existing technologies, a discovery framework was developed where each robot is responsible for matching incoming service requests to its local repository. This distributed approach, where each swarm member performs a part of the matching task, induces considerably lower delays than in the centralised approach. Considering a robot swarm where each robot offers a similar amount of services, the matching delay caused by the centralised approach is ten times higher than in the distributed matching approach. Moreover, since in the distributed approach, each robot is aware of its local context, no extra delays are incurred by the context dissemination process as is the case with centralised matching. The proposed framework also allows to execute composite services in a dynamic way, where the executor for each atomic service is selected based on the current context. Future work includes the dynamic generation of composed services based on the available semantic services as well as the current context. A first step was taken towards offering semantic matching as a service, but some scalability issues remain when the number of robots and offered services increases drastically. Extending the matchmaker to allow semantic grouping of services based on their functionality could overcome these issues. Furthermore, since it concerns computationally extensive matching on mobile devices, energy consumption should be taken into account. During the evaluation, it turned out that there are no benchmarks available for evaluating the accuracy of precondition and effects matching, the development of such benchmarks would be beneficial.

References

1. Baer, P. A., Weise, T., Geihs, K.: Geminga: Service Discovery for Mobile Robotics. Systems and Networks Communications, 2008. ICSNC '08. 3rd International Conference on , vol., no., pp.167-172, 26-31 Oct. 2008 doi: 10.1109/ICSNC.2008.29.
2. Bauer, C.: Cling UPnP, http://teleal.org/projects/cling/.
3. Bellur, U., Kulkarni, R.: Improved Matchmaking Algorithm for Semantic Web Services Based on Bipartite Graph Matching. Web Services, 2007. ICWS 2007. IEEE International Conference on , vol., no., pp.86-93, 9-13 Jul. 2007 doi: 10.1109/ICWS.2007.105.
4. Donoho, A., Costa-requena, J., Mcgee, T.: UPnP Device Architecture 1.1. Architecture. Oct. 2008.
5. Durvasula, S., Guttmann, M., Kumar, A., Lamb, J.: SOA Practitioners Guide, Part 2, SOA Reference Architecture. Combined Effort. 1-52 2006.
6. Fujii, K., Suda, T.: Dynamic service composition using semantic information. In Proceedings of the 2nd international conference on Service oriented computing (ICSOC '04). ACM, New York, NY, USA, 39-48. DOI=10.1145/1035167.1035174
7. Gruber, T.: Ontology definition, http://tinyurl.com/tomgruber.

8. Haseeb, A., Matskin, M., Kungas, P.: Mediator-Based Distributed Web Services Discovery and Invocation for Infrastructure-Less Mobile Dynamic Systems. Next Generation Web Services Practices, 2008. NWESP '08. 4th International Conference on , vol., no., pp.46-53, 20-22 Oct. 2008. doi: 10.1109/NWeSP.2008.23
9. Hristoskova, A., Moeyersoon, D., Van Hoecke, S., Verstichel, S., Decruyenaere, J., De Turck, F., Dynamic composition of medical support services in the ICU: Platform and algorithm design details. Computer Methods and Programs in Biomedicine, v.100 n.3, p.248-264, Dec. 2010. doi: 10.1016/j.cmpb.2010.03.019.
10. IRobot: iRobot Corporation: Home Page, http://www.irobot.com/.
11. Kapahnke, P., Klusch, M.: Semantic web service selection with SAWSDL-MX. Proceedings of the International Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web (SMR2 2008) at ISWC 2008, pp. 318 2008.
12. Kaufer, F., Klusch, M.: WSMO-MX: A Logic Programming Based Hybrid Service Matchmaker. Web Services, 2006. ECOWS '06. 4th European Conference on , vol., no., pp.161-170, Dec. 2006. doi: 10.1109/ECOWS.2006.39
13. Lopes, A.: SEA: A Semantic Web services context-aware execution agent. Proceedings of the AAAI Fall Symposium on Agents and the Semantic Web, 2005.
14. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: OWL-S - Semantic Markup for Web Services, http://www.w3.org/Submission/OWL-S/.
15. Meshkova, E., Riihijrvi, J., Petrova, M., Mhnen, P., A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks. Comput. Netw., Vol. 52, No. 11. Aug. 2008, pp. 2097-2128. doi:10.1016/j.comnet.2008.03.006
16. Mokhtar, S., Preuveneers, D., Georgantas, N., Issarny, V., Berbers, Y.: EASY: Efficient semAntic Service discoverY in pervasive computing environments with QoS and context support. Journal of Systems and Software, Volume 81, Issue 5, May 2008, Pages 785-808, ISSN 0164-1212. doi: 10.1016/j.jss.2007.07.030
17. OWL Working Group: OWL 2 Web Ontology Language Document Overview, 1-12 Oct. 2009.
18. Papakos, P., Rosenblum, D.S., Mukhija, A., Capra, L.: VOLARE: Adaptive Web Service Discovery Middleware for Mobile Systems. In Proceedings of ECEASST. 2009.
19. SemWebCentral: OWL-S Service Retrieval Test Collection: Project Info, http://www.semwebcentral.org/projects/owls-tc/.
20. Stollberg, M., Hepp, M., Hoffmann, J.: A caching mechanism for semantic web service discovery. The Semantic Web. Lecture Notes in Computer Science, Volume 4825, 480-493, Jan. 2007. doi:10.1007/978-3-540-76298-0
21. UPnP: UPnP Forum, http://www.upnp.org/.
22. Urbieta, A., Barrutieta, G., Parra, J., Uribarren, A.: A Survey of Dynamic Service Composition Approaches for Ambient Systems. Proceedings of the 2008 Ambi-Sys workshop on Software Organisation and MonITоring of Ambient Systems. 2008
23. Vallivaara, I., Kemppainen, A., Makela, T., Haverinen, J., Roning, J., Martinez, D. M.: Roboswarm, http://roboswarm.eu/.
24. Voyage: Voyage Linux — x86 Embedded Linux, http://linux.voyage.hk/.