On-Demand Attribute-Based Service Discovery for Mobile WSANs

Klaas Thoelen, Sam Michiels, Wouter Joosen
IBBT - DistriNet, Dept. of Computer Science, K.U. Leuven
Celestijnenlaan 200A,
B-3001 Heverlee, Belgium
klaas.thoelen@cs.kuleuven.be

ABSTRACT

Adequate service discovery is required to support dynamic composition as offered by recent light-weight service platforms for wireless sensor and actuator networks (WSANs). To provide flexible and precise service discovery in such dynamic environments, the operational context of services should be taken into account. This context can be inferred by reusing the information collected by the sensor nodes' environmental and resource monitoring services. In order to be usable, a light-weight abstraction of the contextual parameters is required to provide standardized context descriptions. Furthermore, the service discovery process itself needs to be adapted to handle the flexibility of this approach. In this paper, we present a service discovery mechanism that supports the expression of contextual constraints through the concepts of attributes and dictionaries. We report on the operation of such a service discovery solution and show its feasibility in a proof-of-concept implementation on SunSPOT. In this implementation, we integrate service and route discovery into a single process hereby reducing the required network overhead.

1. INTRODUCTION

Given the considerable deployment cost of a commercial wireless sensor and actuator network (WSAN), demand is growing to share and reuse sensor networks between multiple applications (a.o. to increase the return on investment). Recent research confirms this trend to approach WSANs as light weight service platforms [3, 5, 8, 10]. One of the key requirements to achieve this goal is to offer low-overhead service discovery so that applications can find the services they are interested in.

A cold chain monitoring application, for example, typically searches for (1) a temperature sensing service, (2) provided by network nodes at a certain location or region, (3) that guarantees a minimal accuracy, and (4) has available energy to provide the data at a given sampling rate. Hence, in addition to a semantic specification of the service (i.e. temperature sensing) also the operational context of the service (i.e. location, sensor accuracy and energy) is of importance in WSAN service discovery.

Traditional service discovery protocols in WSANs, however, do not consider the operational context of a service or imply considerable network overhead in doing so [2, 3, 6, 9]. In addition, operational conditions in a typical WSAN are extremely dynamic, what adds to the complexity of service discovery: sensor and actuator nodes join and leave a network (e.g. sensor equipped containers are transported over road, rail, sea and air through various warehouses), communication channels are unreliable (e.g. Farady cage effects in a ship, tunnels), and the available battery, memory, and processing resources change over time.

In light of this, we investigate to what extent service discovery in WSANs can be optimized by exploiting collected contextual sensor data and using it as attributes to enrich on-demand service discovery. Typically, these data are available, but only to the applications that are collecting them. We argue that a service discovery solution should take into account the operational context of the sensor nodes, both from a service perspective (e.g. temperature data, sampling rate) and a node perspective (e.g. location, battery level).

The contribution of this paper is a service discovery mechanism that defines an abstraction for context parameters and a dictionary mechanism that encodes service requests in a small binary format. We have implemented these mechanisms in a proof-of-concept prototype on SunSPOT and maximize efficiency by integrating our solution with the AODV routing protocol. Finally, our approach is evaluated on expressiveness, network overhead, and memory consumption.

The rest of the paper is structured as follows. Section 2 summarizes the requirements of our service discovery solution. An overview of the proposed architecture is given in Section 3, after which we present the details of our proof-of-concept implementation in Section 4. Evaluation of our approach and implementation is given in Section 5 and sections 6 and 7 report on related and future work respectively. We conclude in Section 8.

2. KEY REQUIREMENTS

In this section we identify the key requirements for attribute based service discovery in WSANs. First, an adequate *abstraction* of service types and context attributes is needed to formulate accurate descriptions of services and their context, and allow comparisons. Second, a lightweight encoded *representation* of these descriptions is required to allow expressive service requests with low network overhead. Third, to *exchange* service requests and replies, message

COMSWARE2011 July 01 - July 03 2011, Verona, Italy

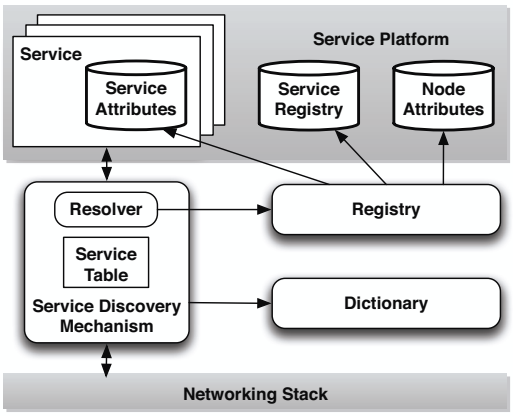

Figure 1: The composition diagram.

structures need to be defined that support the flexibility of attribute-based service discovery. And finally, leveraging on the provided service type and context abstraction, a *matchmaking* process is required to evaluate service requests against offered services and their operational context.

3. SYSTEM ARCHITECTURE

The following sections discuss how our attribute-based service discovery solution handles the requirements identified in Section 2.

3.1 Architectural Overview

The architecture of our attribute-based service discovery mechanism is shown in Figure 1. To limit the network overhead of our service discovery solution, we favor an on-demand and request-reply type of interaction to discover services. We therefore eliminate the central service registry typically used by traditional service discovery systems. Instead we provide a distributed registry, in which each node maintains an overview of at least its own services and operational context. The architecture as shown in Figure 1, is thus deployed on each node in the network.

We abstract from a particular service platform but require that it exposes the following contextual information: (i) *Node Attributes* retrieved from a central context repository on the respective sensor node, (ii) a *Service Registry* provided by the service platform containing semantic information of the registered services and (iii) *Service Attributes* exposed by the services themselves. We refer to [5, 11] for an example of such a service platform.

To provide a single point-of-access, the *Registry* collects all attributes and stores them in a concise abstract representation, called *attribute specifications* (see Section 3.2). Together with the semantic service identifiers found in the Service Registry, this information is provided to the *Resolver*. The Resolver is part of the *Service Discovery Mechanism* and performs a matchmaking process (see Section 3.5) on received local and remote service requests. Besides the contextual information in the Registry, it also consults the *Service Table*, in which all known local and remote services are temporarily cached. The Service Discovery Mechanism exchanges service request and reply messages with neighbouring nodes to perform its discovery task. Just before or after transmission, their contents is encoded into or decoded from a lightweight representation using the *Dictionary* component.

Table 1: An example dictionary.

Attribute	Data Type	Encoded
TemperatureAccuracy	float	0x00
Battery	integer	0x01
LocationX	float	0x02
LocationY	float	0x03

Operator	Encoded
=	0x00
>	0x01
<	0x02
>=	0x03
<=	0x04
!=	0x05

Service Type	Encoded
PeriodicTemperatureSensor	0x00
MonitoringService	0x01
CoolingService	0x02

3.2 Attributes and Service Descriptions

To describe the context, in which a service is required to operate, we provide an abstraction called *attribute specifications*. These contain attribute-operator-value tuples, in which (i) an *attribute* is a quantity (e.g. temperature, space coordinate), (ii) an *operator* expresses the relation between the attribute and value (e.g. equal to, less than), and (iii) a *value* indicates a certain limit on the attribute. An example attribute specification is `{battery, >, 50}`, which expresses that the battery level of the respective node should be higher than 50%.

Together with a semantic service identifier, a list of attribute specifications constitutes a service description. As an example, consider a process in search of a data processing service close-by and on a node with at least 50KB of memory left to store intermediate results. In such a service request, semantic information is provided by an identifier for the processing service, while a set of attribute specifications specify contextual location and memory constraints.

3.3 Service and Attribute Dictionary

We propose application specific dictionaries to provide a manageable standardization of service types, attributes and operators (see Table 1). Each dictionary contains a set of service types, attributes and operators, commonly used and meaningful in a specific application domain like e.g. wildlife monitoring, logistics, building automation. This restriction reflects the practical use of WSANs, which are typically deployed to serve a certain single application domain.

To contribute to a lightweight discovery operation, the dictionaries contain a simple byte coding scheme used to encode string literal service types, attributes and operators into byte representations. This decreases the network overhead or allows a larger set of constraints to be specified in a single service request. Service types and attributes are encoded into a single byte and can take up to 256 values. Operators are encoded in only 4 bits, which is more than enough to represent all relational and logical operators needed for service discovery. Attributes further specify the data types of their respective values to allow for correct interpretation.

3.4 Service Requests and Replies

To discover a service, a service request message is disseminated in the WSAN, which is replied to by a service reply message when a suitable service is detected. The structure of these messages determines the flexibility that is available to describe a service and to influence the service discovery process in terms of strictness and redundancy. In this section we describe how these message constructs look like in our system.

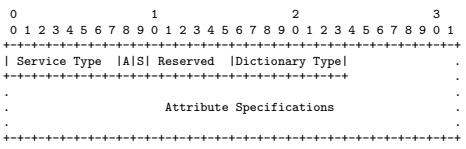

Figure 2: The service request message structure.

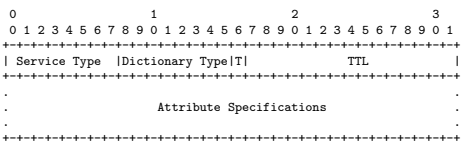

Figure 4: The service reply message structure.

Figure 2 depicts the structure of a *service request*. The *Service Type* field is a single byte representation of the service's semantical meaning. It is followed by a number of flags that influence the behavior of the service discovery process. Only the *All-* and *Strict-flags* are currently in use. The All-flag indicates whether only one or all services in the network satisfying the request are searched for. The Strict-flag controls how strict the request needs to be complied to. It is set if all constraints in the request have to be met in order to positively reply to the service requester. If not set, the service matchmaking process ignores attributes related to contextual data that is not provided by the processing node. The *Dictionary Type* field indicates which dictionary the request uses and hence determines the application domain. By checking the message's dictionary type against its own, a node determines whether or not it can meaningfully interpret the contents of a message. Finally, a set of attribute specifications is included, to which the service's operational context needs to comply.

Attribute specifications are constructed by a series of message constructs as shown in Figure 3. The *Attribute* field defines the attribute that is considered. The *Opr* and *Value* fields in turn, contain the operator and a parameter respectively, hereby completing the attribute specification tuple. The *M*-flag indicates whether *more* constraints on this attribute follow and allows for daisy chaining to express constraints like larger than ... and smaller than ... . The *Value length* field, in turn, indicates the number of bytes the value field occupies. This is particularly useful in the case a string literal needs to be described. The Attribute and Operator fields are encoded using a byte representation as described in Section 3.3.

A *service reply*, as shown in Figure 4, is used by the resolving node as a notification that a matching service is found. Since we allow for partial compliance through the Strict-flag in the service request, we include the complied to attribute specifications in the service reply. These differ from the structure in Figure 3 in that the operator field is omitted and that the values reflect the actual context at the provider. This way the requester knows to what extent a non-strict service request was complied with. The *Service type* and *Dictionary type* fields are analogous to the service request. The *TTL* field indicates the time-to-live of the service related information in this service reply either in seconds or hours, according to the *T*-flag or TTL-modus-flag.

3.5 Service Query Matchmaking

In this section we describe the matchmaking process that evaluates received service requests against the set of offered services and their operational context.

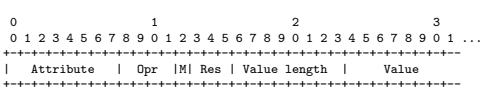

Figure 3: The attribute specification structure.

Upon reception of a service request, the included service type and attributes are extracted and decoded according to the local dictionary. Decoding is performed by a simple look-up action of the service type, attribute or operator that corresponds with the received encoded version. After decoding, first the service type is evaluated; if a service of the specified type is known, either locally or on a remote node, evaluation of the attributes follows. If such a service is not known, the evaluation terminates. The evaluation of the specified attributes takes place by recreating the original attribute-value-operation tuples and evaluating them against the collected contextual data in the Registry.

The result of the matchmaking process depends on the values of the Strict- and All-flags in the service request. Not setting the Strict-flag opens up the possibility to formulate elaborate service requests and receive replies for only partially complying services, hereby introducing some tolerance into the service discovery process. In cooperation with the All-flag this allows for an array of possible discovery results ranging from all services in the network that only comply partially to the request, up to a single instance of the service as described exactly in the service request.

3.6 Service Invocation

Successfully discovered services are most likely to be used by the originator of the service request. Discovering the interfaces of the service is however outside the scope of this paper as we focus on service platforms in which services are interacted with using a generic interface [11].

4. PROOF OF CONCEPT

To show the feasibility of our approach we made a proof of concept implementation on the SunSPOT platform [1]; sensor nodes developed by Sun Microsystems, Inc. (now Oracle America, Inc.), running the Java Squawk virtual machine on a 180MHz ARM920T processor with 512KB of RAM and 4MB of flash memory. We used the red-090706 version of the provided SDK.

To limit the network overhead of our service discovery solution, we integrated it with the Ad-hoc On-Demand Distance Vector (AODV) routing protocol [7] provided by the SunSPOT platform. Such an integration has previously been proposed in [2, 6, 9]. In the following sections we shortly introduce AODV and discuss the integration together with its advantages.

4.1 AODV operation

AODV is a MANET routing protocol mostly applied in wireless networks with volatile and short-lived routes due to node mobility and intermittent wireless connections. When a route to a certain destination is needed, a route request indicating that destination's address is broadcasted over the network. Any node that either has this address configured

as his own or that has an active route to such a node replies to this request. It creates a route reply packet and unicasts it back hop-by-hop to the originator of the request. As a result, all intermediate nodes know via which neighboring node they can reach the required destination, herebye creating a route to it.

4.2 Service and Route Discovery Integration

As discussed in Section 3.1, we favor an on-demand type of interaction for service discovery similar to what is performed in the AODV route discovery process. Instead of looking for a route to a node however, we are interested in which node provides a certain service. Since the location of the providing node is not known a-priori, a network wide broadcast of the service request is required to locate it, followed by a service reply to inform the requester of the providing node's address. Because of the similarity of the route and service discovery mechanisms and the fact that most searched for services will be used once found, hereby requiring a route to its provider, both mechanisms can be integrated resulting into a single discovery action. This is achieved as follows.

A source node requiring a service creates a service request as mentioned in Section 3.4 and piggybacks it on a AODV route request. In this resulting request packet, the destination address (to which to find a route) must and cannot be set, since at this stage the service provider is still unknown. Once this packet is broadcasted, receiving nodes identify it as a AODV route request with a concatenated service request and execute the service and route resolution decision tree as shown in Figure 5. This decision tree is an extension of the one described in [2].

Since at this stage the destination address is not set, the left part of the decision tree is evaluated, which essentially performs *service discovery*. The Resolver checks if a suitable service is known at this node. If this is not the case, the service request is rebroadcasted, hereby ending the decision tree evaluation. In case a service is known, the Resolver checks if it also has a route to the service provider. This can be the case either because the service is provided by the node itself or because it has a route stored in the AODV routing table due to previous service or route discovery activities. If not, the service provider's address is entered in the destination field of the request and the latter is rebroadcasted in search for a route. If a route is available however, the Resolver checks the All-flag in the service request. If set, two actions take place. First, for all services known by the processing node that comply with the service request, a reply is sent to the originator of the service request. Second, the request is rebroadcasted so that additional services can be found. If the All-flag is not set, only a single reply is sent to the originator of the request.

Rebroadcasted requests with a filled-in destination field, trigger an additional *route discovery* process. This involves

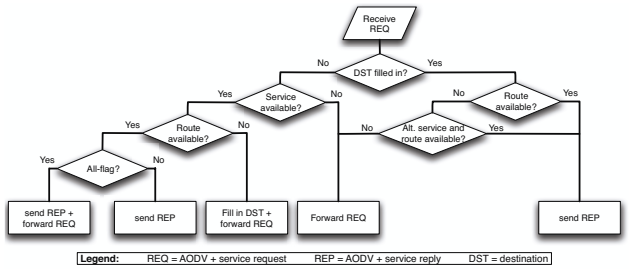

Figure 5: The service and route resolution decision tree.

the evaluation of the right side of the decision tree at subsequent nodes. If these nodes have a route available to the specified address, a reply is transmitted to the originator of the request. If this is not the case, but the processing node is aware of another suitable service, possibly on another node, a reply is transmitted pointing at this alternative service. Otherwise, the request is rebroadcasted as it was.

Every node receiving a reply strips the service reply from the route reply. The service-related information in the service reply is stored in the Service Table and the routing information in the AODV routing table. This information is cached during the specified TTLs in the replies and is used to resolve future service and routing requests. If the receiving node is the service reply's addressee, the Resolver takes act of the discovered service. If in the respective request the All-flag was set, the Resolver waits until a timer elapses to allow other replies to arrive before notifying the service discovery initiator. Once the timer elapses, all service replies are reported to the initiator. In case the All-flag wasn't set, the service reply is immediately reported to the request initiator and the associating timer is cancelled. It is possible however that no replies are received in which case the service discovery initiator is notified hereof after the timer elapses.

5. EVALUATION

We evaluate our architecture and proof-of-concept implementation on three characteristics; (i) expressiveness of our attribute based service queries, (ii) network overhead of the integrated service and route discovery mechanism, and (iii) the memory and code overhead of our implementation. We start however by introducing a scenario that functions as a concrete use case throughout the evaluation.

5.1 Evaluation Scenario

Imagine a cooled warehouse, in which temperature sensitive products, e.g. fruit, continuously enter and leave as part of their supply chain. A back-end control application controls the local temperature and each crate is equipped with a sensor node providing a temperature sensing service.

In such a scenario, assume that one of those sensor nodes detects a too high temperature. To adjust the cooling in the warehouse, it needs to report the high measurement to the control application. However, because it is new in the local network, it does not know where this application can be found. Consequentially, it initiates a service discovery round that resolves the gateway as the application's provider. Once found, the high temperature reading, along with additional information like the node's location, is forwarded to the control application. A single high reading however is not sufficient to adjust the cooling of the warehouse. Additional readings are needed to confirm the rise in temperature. As a result, the control application sends out a new service request for all temperature sensing nodes in close proximity to the original temperature sensor, with a high enough accuracy and more than 50% of remaining battery lifetime. Once such nodes are discovered, the control application queries them for additional temperature readings. According to their values, the cooling in the warehouse can be adjusted in a sensible manner.

5.2 Expressiveness of the Service Queries

The expressiveness of the service queries is determined by the amount of attributes that can be specified in a service request. Figure 3 shows that an attribute specification consumes 3 bytes plus the amount of bytes required for the

Table 2: Overview of the attributes specified in the second service request of the evaluation scenario.

Attribute	Operator	Value
LocationX	>=	0
LocationX	<=	2
LocationY	>=	0
LocationY	<=	2
TempAccuracy	<	0.5
Battery	>	0.5

attribute's value. The latter largely depends on its data type and can be anything from one byte (i.e. small number) up to tens of bytes (i.e. a string literal). The 4 bytes Java uses to represent the integers and floats of the second service request of the evaluation scenario can thus be regarded as a good average size of an attribute's value (see Tables 1 and 2). Hence, the average attribute can be described using 7 bytes.

The network layer of the SunSPOT platform, which is situated above a traditional IEEE802.15.4 layer, allows for payloads of up to 85 bytes. AODV headers and the service request's header respectively consume 24 and 3 bytes, which leaves us with 58 bytes to describe additional attributes of a service. This allows us to specify up to 8 attributes of the previously defined average size of 7 bytes. Given that the rather extensive second service request of the evaluation scenario contains 6 attributes (see Table 2), a maximum of 8 attributes can be considered to provide us with a fair level of expressiveness. Of course a service request can be fragmented over multiple packets as well, although this is considered to be less desirable.

5.3 Network Overhead

In this section we evaluate the influence of the service and route discovery integration on the network overhead. We consider a small network topology that reflects the evaluation scenario as described in Section 5.1 and shown in Figure 6. Sensor node 1 detects a too high temperature and initiates the service discovery for the monitoring service provided by a back-end infrastructure accessible via the gateway. Consequentially, the monitoring service initiates a service discovery for additional temperature sensing services in the proximity of sensor node 1, which will be replied to by sensor nodes 1, 2 and 5. We assume empty caches on all intermediate nodes.

In Figure 7 we compare the practically determined total amount of transmissions used to perform both discovery actions with 2 theoretical cases. The transmissions that are taken into account are those used to forward service and route requests, service and route replies and acknowledgements to the replies. On the one hand we see that the difference between the practical and theoretical integrated

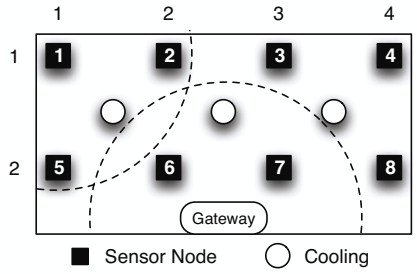

Figure 6: The topology used during network overhead evaluation. The dashed lines indicate the used radio range.

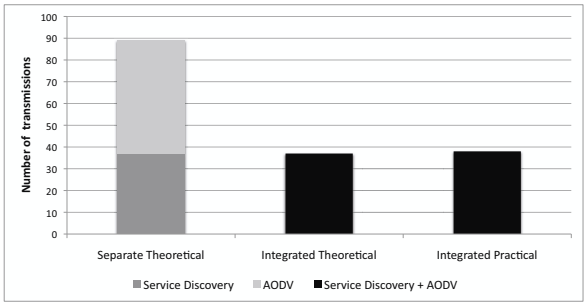

Figure 7: Barchart showing the amount of transmissions required and measured during the network overhead experiments.

approaches is minimal, i.e. in average a single transmission. The slightly higher number of transmissions measured was due to unacknowledged service replies, which consequentially were retransmitted. The small difference demonstrates the correct operation of our implementation. On the other hand, we can deduce from the figure that integration of service and route discovery reduces the amount of transmissions by more than 50%, i.e. from 89 transmissions to 37 (theoretical) or 38 (practical). This reduction equals the amount of transmissions needed for separate route discovery of the service providing nodes. In the integrated approach, a single network wide broadcast is needed to discover both services and routes. In the separate approach however, the worst case scenario of the temperature service discovery needs three broadcasts only for route discovery; this because the service discovery process results in the offering of three temperature service providing nodes, for which separate route discoveries need to be performed. As expected, the integration of service and route discovery largely reduces network overhead while performing service discovery.

5.4 Code and Memory Overhead

The AODV implementation on SunSPOT uses about 3001 lines of code. To implement the additional service discovery and dictionary functionality, an additional 2761 lines of code were needed.

To evaluate the memory overhead, we measured the additional memory use introduced by the service discovery mechanism on top of the original SunSPOT AODV routing layer. The average memory use increased from 8924 bytes without service discovery up to 10412 bytes with service discovery. This low increase of 1488 bytes or 17% can be attributed to the fact that the AODV implementation was reused for a large part and only code for creating and interpreting a number of additional headers and maintaining the service registry cache was added. Finally, the dictionary functionality consumes an additional 1860 bytes. This increase is largely due to the object-overhead of String and HashTable objects needed to store the contents of the dictionary.

We consider the extra code and memory overhead to be a small price for the added functionality of attribute-based service discovery. If needed however, a considerable reduction in memory overhead can be expected by implementing in C instead of Java.

6. RELATED WORK

While service discovery and context-awareness are well-investigated topics in WSAN research, their combination is

not. In this section, we discuss relevant related work.

In [4], attributes are used to describe or name data sources, which are discovered through Directed Diffusion. The integration of naming and routing in the application level minimizes the amount of naming indirections, hereby eliminating the communication costs of maintaining central information of provided services in the network. Directed Diffusion however enforces a strict matchmaking process that disallows partial compliance to the service request. Furthermore, while important to us to allow reasoning over a set of service providers, the identification of the service provider, or data source, is not supported.

In [3], the authors describe a process and a system architecture to query, select and use services in the SOA based Internet-of-Things. They decouple service type discovery from service instance discovery. While, the latter is initially based on service advertisements and a central service repository, they include on-demand service discovery as a last measure, yet do not provide a detailed description how to do so. Additionally, the operational context that is included during instance discovery, is rather limited to location only.

The integration of service and route discovery protocols was proposed earlier by [6, 9, 2]. They all however target ad-hoc networks and MANETS, not WSANs. Furthermore, they do not consider the operational context of the services to be discovered.

7. FUTURE WORK

We identify three tracks for future work that all aim to increase the usability of the proposed solution.

First, the flexibility of the attribute specifications and dictionary system can be further improved concerning the relationships between attributes and the redundancy they introduce. As an example, *temperature sensor accuracy* and *humidity sensor accuracy* are two distinct attributes in the current solution, but clearly have a large overlap in their meaning. The research area of ontologies might suggest better ways of structuring our attribute sets.

An updating system of the dictionaries is a second item of future work. As deployments evolve, new services and hardware are introduced in the system. These additions cause new attributes and service types to appear, which need to be included in the dictionary to allow en- and decoding. Additional support is needed to update the dictionaries in an incremental manner, instead of redundant bulk updates.

Finally, the registry can be opened up for other purposes than service discovery. A group management protocol for instance, might select the members of a group based on their operational context. Furthermore, a queryable context registry will provide easier reuse and sharing of contextual data amongst services. An easy, yet generic and flexible interface to the context registry needs to be developed.

8. CONCLUSION

Service discovery in mobile WSANs can benefit greatly by taking into account the operational context of the services to be discovered. This increases the flexibility and precision of the service discovery process. By standardizing and encoding the contextual constraints, this can even be achieved in a light-weight manner. We presented both an architecture and a proof-of-concept implementation of such an approach and proved its feasibility through evaluation. Our dictionary system is expressive enough to formulate complex service requests and the overall networking overhead is reduced greatly by the integration of the service discovery into the AODV routing protocol.

9. ACKNOWLEDGEMENT

The authors are grateful to Pieter Jacobs and Johannes Dewitte who completed part of the presented research in the form of their master's thesis.

This research is partially funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, and the Research Fund K.U.Leuven. It is conducted in the context of the IWT-SBO-STADiUM project [1].

10. REFERENCES

[1] SunSPOT. http://www.sunspotworld.com/, 2011.

[2] C. Frank and H. Karl. Consistency challenges of service discovery in mobile ad hoc networks. In Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems, MSWiM '04, pages 105–114, 2004.

[3] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio. Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services. IEEE Transactions on Services Computing, 3:223–235, 2010.

[4] J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan. Building efficient wireless sensor networks with low-level naming. In Proceedings of the eighteenth ACM symposium on Operating systems principles, SOSP '01, pages 146–159, 2001.

[5] D. Hughes, K. Thoelen, W. Horré, N. Matthys, S. Michiels, C. Huygens, W. Joosen, and J. Ueyama. Building wireless sensor network applications with looci. International Journal of Mobile Computing and Multimedia Communications, 2(4):38–64, 2010.

[6] R. Koodli and C. E. Perkins. Service Discovery in On-Demand Ad Hoc Networks. Internet-Draft draft-koodli-manet-servicediscovery-00.txt, IETF, Oct. 2002.

[7] C. E. Perkins, E. M. Belding-Royer, and S. R. Das. Ad hoc on-demand distance vector (aodv) routing. RFC 3561, IETF, July 2003.

[8] N. B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao. Tiny web services: design and implementation of interoperable and evolvable sensor networks. In Proceedings of the 6th ACM conference on Embedded network sensor systems, SenSys '08, pages 253–266, 2008.

[9] B. Raman, P. Bhagwat, and S. Seshan. Arguments for cross-layer optimizations in bluetooth scatternets. pages 176 –184, 2001.

[10] A. Rezgui and M. Eltoweissy. Service-oriented sensor-actuator networks: Promises, challenges, and the road ahead. Computer Communications, 30:2627–2648, September 2007.

[11] K. Thoelen, N. Matthys, W. Horré, C. Huygens, W. Joosen, D. Hughes, L. Fang, and S.-U. Guan. Supporting reconfiguration and re-use through self-describing component interfaces. In Proceedings of the 5th International Workshop on Middleware Tools, Services and Run-Time Support for Sensor Networks, MidSens '10, pages 29–34, 2010.

[1] http://distrinet.cs.kuleuven.be/projects/stadium/