Contract-Based Discovery in Sensor Web [*]

Javier Cubo
Dept. Computer Science
Univ. of Málaga, Spain
cubo@lcc.uma.es

Nadia Gámez
Dept. Computer Science
Univ. of Málaga, Spain
nadia@lcc.uma.es

José Antonio Martín
Dept. Computer Science
Univ. of Málaga, Spain
jamartin@lcc.uma.es

Lidia Fuentes
Dept. Computer Science
Univ. of Málaga, Spain
lff@lcc.uma.es

The evolution from traditional wireless sensor networks towards the Internet of Things has originated the need of interconnecting heterogeneous devices. Sensor Web represents a web of sensor networks accessible by means of service interfaces which enable an interoperable usage of sensor resources. Hence, a shift in the sensor-as-a-service paradigm is required. Dynamic, scalable and adaptive service discovery is crucial in this new scenario of diverse and scarce resources. Contracts present a versatile solution to reason, negotiate and discover services based on their characteristics and capabilities, as well as on their current states. Therefore, we propose a novel approach to address discovery of heterogeneous sensor nodes based on contracts. We use ontologies to capture and support sensor descriptions with semantics concepts and their relationships, increasing the interoperability and allowing complex reasoning in the discovery. We have implemented the discovery process as a service provided by FamiWare, a middleware family for Ambient Intelligence.

1 Introduction

The Future Internet has emerged as a new initiative to pave a novel infrastructure linked to objects and things of the real world, ranging from mobile devices and sensors to cars and electrical appliances[1], in order to meet the changing global needs of business and society. Future Internet applications will have to support the interoperability between many diverse stakeholders or objects by governing the convergence and lifecycle of Internet of Contents (IoC), Services (IoS), Things (IoT), and Networks (IoN). These applications should handle dynamic and continuous changes, for instance, in the provisioning of services, availability of things and contents, connectivity of networks, mobility of wireless connected objects and so on. These objects will sometimes have their own Internet Protocol addresses, be embedded in complex networks and use sensors or tags to obtain information from their environment (e.g., recording temperature or humidity), and/or use actuators to interact with it (e.g., air conditioning valves that react to the presence of people) [3].

Specifically, sensor networks are increasingly being used for monitoring different domains, such as environmental, agriculture, industrial or traffic. The Open Geospatial Consortium (OGC) proposes the Sensor Web Enablement (SWE) framework[2] in order to integrate heterogeneous sensor. SWE promotes Service-Oriented Architectures (SOAs) [2] to specify Web service interfaces used for accessing sensor data, controlling sensors and alerting based on measures. In OGC, a Sensor Web represents a web of sensor networks accessible by means of service interfaces which enable an interoperable usage of sensor resources [1]. Hence, a shift in the sensor-as-a-service paradigm is required. Dynamic, scalable and adaptive service discovery is crucial in this new scenario of diverse and scarce resources. Contracts

[*]This work is partially supported by the projects TIN2008-05932 (ReSCUE) and TIN2008-01942 funded by the Spanish Ministry of Science and Innovation (MICINN) and FEDER; and the projects P07-TIC-03131 and P09-TIC-05231 (FamWare) funded by the Andalusian local Government.

[1]In the sequel, we use the terms devices and sensors indistinctly to refer to whatever node in a network.

[2]`http://www.opengeospatial.org/projects/groups/sensorweb`. Accessed on 20 April 2011.

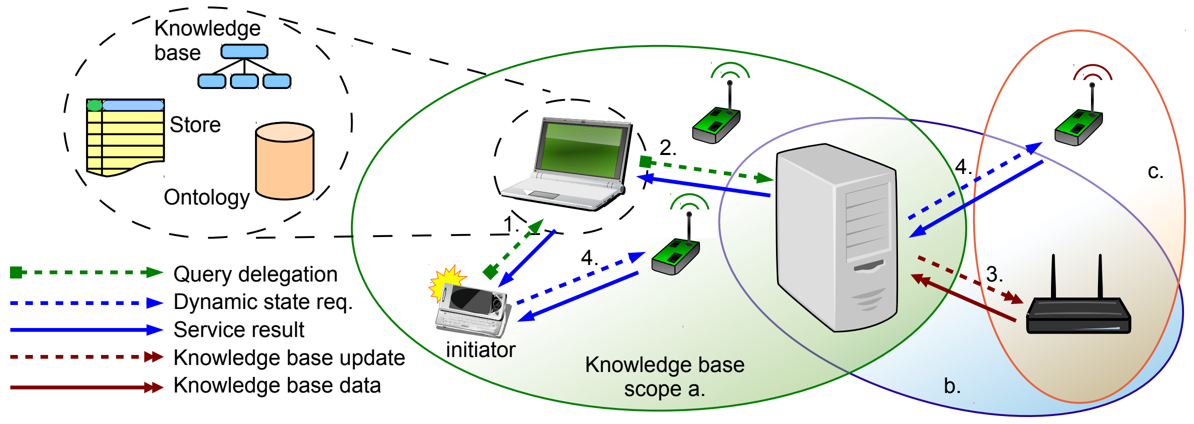

Figure 1: Overview of our Discovery Approach

present a versatile solution to reason, negotiate and discover services based on their characteristics and capabilities, as well as on their current states.

In this paper, we propose a novel approach to address discovery of heterogeneous sensor nodes based on contracts. Specifically, we use contracts in order to specify a query and find out the most suitable sensor nodes for that query. In our approach, the contracts could be negotiated in case there not exist any node fulfilling the initial query. Moreover, the system could be reconfigured dynamically in order to satisfy the contract. In addition, we take advantage of ontologies to capture and support sensor descriptions with semantics concepts and their relationships, increasing the interoperability and allowing complex reasoning in the discovery.

An overview of our approach is presented in Figure 1, which will be detailed throughout the paper. This figure represents all the possible interactions supported by our approach. Our proposal supports a wide range of devices with different capabilities and requirements and hence it comprises a range of different messages.

The basic message is *query delegation*, which is a discovery request that must be replied either synchronously or in an eventual *service result* message. Query delegation can entail a subsequent chain of delegations to broaden the scope of the discovery process. Additionally we have *dynamic state requests* which demand the current dynamic state of the receiving device. This request is also replied with the service result message. Unlike query delegation messages, dynamic state requests are never delegated.

Nodes with larger capabilities in memory and energy may (optionally) serve as a registry of services or, more precisely, a cache of known services. This cache, known as *knowledge base* of the node and represents the scope of services known to it. For example, in Figure 1, the laptop is aware of the services in scope a, the desktop computer knows the services in both the scopes a and b whereas the router is contains in its knowledge base the single mote of scope c.

The knowledge base is populated in two ways: i) via the chains of query delegation that the node participate in, and hence intermediates in the service result messages; and ii) via *knowledge base update* messages that can be sent to other nodes with knowledge base. The latter kind of messages is replied with a *knowledge base data* which contains the requested knowledge base.

We build our approach on two assumptions: to try to minimise the number of communications required for a distributed discovery process and large messages consume more resources (mainly battery) than several small control messages. In this sense, those communications which entail service information (i.e., service result and knowledge base data) are the only large messages in our approach, and they should be used sparringly.

Our discovery process, which has implemented as a service provided by FamiWare, a middleware family for Ambient Intelligence, is conceived to address several challenges occurring in Wireless Sensor and Actuators Networks (WSAN).

Hierarchical. Certain nodes have a hierarchy (e.g., based on resources or spanning tree) which will be used to delegate queries. For instance in Figure 1, the laptop might be the head or sink of the motes in scope a whereas the mote in scope c depends on the router nerby.

Indirect. It should be able to discover services not directly known by the head of the hierarchy. For instance, the initiator must be able to discover the mote in scope c.

Dynamic scope. New services which fulfill the query can be discovered even after the query has been initially answered.

Dynamic characteristics. The query can be composed of static features (such as functionality, ID and hardware capabilities) and dynamic features (such as QoS, remaining energy and location, for instance). The former can be cached and stored in the knowledge base to quicken future discoveries but the later have to be consulted when the services are needed.

Aggregated results. We aim at supporting queries that must be collaboratively answered by several services working together. Therefore, the possible results of the discovery are not just a set of candidate services but a set of sets of services that fulfill the query altogether.

Robustness. A service can always be discovered providing that it is possible to (indirectly) communicate with it. Nonetheless, we assume that it is provided a transport layer in charge of message communication and routing. Therefore, the transport layer is considered out of scope.

The rest of this paper is structured as follows. Section 2 presents our model to specify contracts. In Section 3, the discovery process is described. Section 4 details the discovery process incorporated into FamiWare middleware, in the scope of this work. Finally, in Section 5 some conclusions and comparison with related works are drawn, and plans for future work are outlined.

2 Contract-Based Model

In this section, we present our model based on contracts used to reason, negotiate and discover heterogeneous sensor nodes. Contracts specify queries on sensor characteristics and capabilities, as well as on their current states.

2.1 Sensor Characteristics

Resources in IoT are limited. Communication in devices with limited capabilities causes a major consumption of their resources, such as the battery which is very scarce in these devices. Hence, one of the main goals in a discovery process in Sensor Web is to reduce as much as possible the communication between devices. In order to achieve this goal, when a device needs to discover other devices or sensors satisfying certain characteristics, the discovery process should not rely on expensive message flooding. Instead, we propose to perform queries in a two-step process by using a *knowledge base* or repository that contains information about the sensor networks related to certain sensor characteristics. This repository is updated when a new sensor is incorporated to the network, by including its characteristics to such knowledge base. The first step of the process consists in a negotiation by means of the knowledge base, and in the second step the queries are directly executed on the devices obtained in the first step.

On the one hand, in the first step, the knowledge base contains the static characteristics known at design-time corresponding to the devices registered in such a base. These static characteristics are the following: (i) *Sensor Identification*, the identification of a node within a network; (ii) *Sensor Description*, the description of a node; (iii) *Sensor Type*, the type of a sensor node (e.g., static, mobile, mote-like, body sensor, etc.); (iv) *Sensor Location*, the position of a static node in a network; (v) *Sensor Capabilities*, the sensing units of the sensor node (e.g., temperature, humidity, etc.); (vi) *Sensor Data*, the data that can be sent by a node to other nodes of the network (e.g., units of measurement); and (vii) *Sensor Lifetime*, the initial lifetime of the sensor.

On the other hand, in the second step, we assume the dynamic characteristics can change at run-time. Then, these characteristics are not represented in the knowledge base, thereby it is required to ask the devices for them. Dynamics characteristics may be the following: (i) *Sensor Battery*, the level of a sensor node battery; (ii) *Sensor Node Traffic*, the amount of traffic sent and received by the node; (iii) *Sensor Location*, the position of a mobile node in a network (by using GPS or relative positioning algorithms); (iv) *Sensor Active Capabilities*, the sensing units which are active at run-time; and (v) *Sensor Node State*, the state of a node (e.g., alive, dead, idle, slept, etc.).

Next, we describe the syntax of our contract-based model, as well as how a query is specified in our contract model. In addition, we briefly mention the semantic annotation we use to relate concepts of the query with those corresponding to sensor nodes.

2.2 Contract in a Nutshell

In our proposal, a contract is specified as a query described in a XML-based language for advertising the capabilities of sensor nodes providers according to certain preferences. The knowledge base can be specified as a XML document. Technologies such as XQuery, XPath or XPointer can be used to manipulate well-formed XML documents consisting of information items and structural metadata which define the relationships between the items. In order to represent queries, we use XQuery [12], which is a query and functional language recommended by W3C for providing flexible query facilities to extract and manipulate data expressed in XML by means of expressions[3]. Then, in our proposal, XQuery allows the selection of sensor nodes by means of the different criteria or preferences established in the query, considering that all sensor nodes are registered in the knowledge base whose information is represented in XML. In XQuery, the queries can be made up of up to five different types of clauses, following the SQL-like FLWOR expression for performing joins: For, Let, Where, Order, and Return.

Example 1: We present a query specified in XQuery on the knowledge base encoded in an XML document called *"network.xml"*. This query lists all the sensors with capabilities `temperature` and `humidity`, and being `mobile` `mote-like` nodes as static characteristics; and with `battery` more than 50% as dynamic characteristic.

```
for $node in doc("network.xml")/Middleware/Device
where every $device in $node//Device satisfies ($device//SensingUnits/Temp
      and $device//SensingUnits/Humidity and $device//Type="MoteLike"
      and and $device//Type/Mobility="Mobile" $device//Battery>50%)
order by $/device//Battery
return $node/Device
```

[3]`http://www.w3.org/XML/Query/`. Accessed on 10 April 2011.

As we will explain in Section 4, the knowledge base is represented by using Feature Models [8]. We also use a feature modelling tool, called Hydra [5], in order to solve the queries.

Since a network is made up of heterogeneous sensor nodes, the same sensor characteristics may be described by using different concepts (e.g., `Temp` or `temperature`). Therefore, we use ontologies to represent both static and dynamic characteristics with semantic concepts and their relationships in order to lead to the development of Semantic Sensor Web (SSW), increasing the scalability and interoperability and allowing complex reasoning in the discovery process [11].

Among the different languages that focus on Semantic Web technologies, W3C recommends OWL to capture semantic descriptions. OWL proposes a formal representation of a set of concepts within a domain by capturing the relationships between those concepts. This is called an *ontology*, and OWL is currently the *de facto* standard for constructing ontologies. Using OWL ontologies a sensor node could describe its characteristics. Therefore, using a sufficiently expressive semantic model, the queries can be specified as both qualitative and quantitative preferences.

A contract creates a negotiation, explained in the next section, that comprises both static and dynamic characteristics of the sensor nodes, and may change dynamically depending on the nodes found in our discovery process (see Figure 1) described in Section 3.

2.3 Static and Dynamic Negotiation

As aforementioned, the negotiation of our contracts is divided in two phases: firstly we solve the static characteristics, and secondly the dynamic ones. In the following steps we present the negotiation process.

1. First, the knowledge base, which contains the information of all sensor nodes registered in this base, is queried for the sensor static characteristics. This corresponds to the message number 1 in Figure 1. All those nodes fulfilling the required static characteristics are obtained and ranked according to their suitability to the contract. In our example, `mobile mote-like` nodes that are able sensing `temperature` and `humidity` are selected.
2. If no sensor node is found, then a negotiation to reduce the contract could be performed in two ways (under the user's decision). Following with our example, let us suppose there not exist `mobile mote-like` nodes with the capabilities to sense `temperature` and `humidity` at the same time.
   a) Devices not satisfying the less prioritised characteristics could be obtained. User must provide the information about the prioritisation, otherwise a default order will be used. In our scenario, `mobile` characteristic could be considered the less prioritised one.
   b) Devices satisfying the major number of characteristics could be selected, considering in this case that the characteristics no fulfilled are in between the less prioritised ones. For instance, here nodes which can sense `temperature` and `humidity`, even if they do not satisfy the other static characteristics, could be selected.
3. Once the devices fulfilling the static characteristics have been discovered, our contract establishes an order with respect to the preferences of the static characteristics of the selected devices. This ordering is taken into account for the search of the devices that could satisfy the dynamic characteristics. This search (message number 4 in Figure 1) returns as many devices as have been requested in the query, ending the search at the moment in which those devices are found. In such a way, we avoid to ask more devices than needed, by reducing once again the communication. For our example, we assume that in the previous steps corresponding to the static phase, three nodes has been returned, and we need only one of them that fulfills the dynamic characteristic, i.e., its `battery` must be more than

50%. Therefore, this dynamic phase asks one by one until find out the first one with the battery level more than 50%.

4. If after this search, no device has been discovered, then three process could be performed (under the user's decision).
   a) Initialise a negotiation to reduce the contract of the dynamic characteristics. For instance, the required battery level could be reduced to 40%.
   b) Reinitialise the initial negotiation by reducing the contract with respect to the static characteristics. We proceed in a similar way to step 2.
   c) Reconfigurate the system when it is possible, in order to obtain the enough number of nodes satisfying the contract. For instance, if our query will request for the battery lifetime instead of the battery level, then the negotiation could reconfigure a node with a similar battery lifetime than the required one by reducing some sensing tasks or the frequency of sensing.

In the next section, we present our discovery process based on contracts and the previous negotiation process, as well as current data diffusion mechanisms in WSAN.

3 Discovery Process

Our discovery approach has been inspired by data diffusion mechanisms in sensor networks [6, 7]. Depending on their capabilities, nodes might have a *knowledge base* (see Section 2) and a *store* to keep track of recent discovery requests and a cache of their results. The steps of the discovery process are enumerated as follows:

1. If the query was already known by the receiving node, return the results which were associated with it, if any. Otherwise store the query, its source node and the current time. When the device is aware of new services, these are evaluated against stored queries (i.e., *matched*) and, if they match any of these queries, a *result update* is emitted. This result update is a message analogous to the one containing the discovery results but it is sent when the query is considered already solved. The matching is performed according to the contract evaluation explained in Section 2.3.
2. Check against the local knowledge base to find possible matches. Keep track of partially matching services.
3. If it exists, delegate the query on the direct parent in the hierarchy only if the query was not received/delegated from it. Then, the parent starts the process from step 1.
4. If no enough services are found then evaluate the *next closest match*.
   a) If the current device has enough resources, request a *knowledge base update*.
   b) Otherwise delegate the query. In this case, the intermediary remembers the request and dynamically provides new results if they appear later.
5. Repeat step 4 until the request is fulfilled or no more candidates are found.
6. Once the query is solved, store the results under the query, source and time stored in step 1.
7. If it was a delegated query, rely the answer to the source node (i.e., the one stored in step 1).
8. If the node receives a control message notifying about a query expiration (i.e., a *forget-query message*), or after a certain timeout, remove the query from the store.

The search in the local knowledge base (step 2) is done following the procedure described in Section 2. Let us highlight that, if the discovery query depends on some *dynamic characteristics*, these are requested on-demand to the services which partially match the query. For instance, the message number 4 in Figure 1 is a request of the current dynamic characteristics.

For devices with limited storage or computation capabilities which lack of knowledge base, or in those cases where no match is found locally but the current node belongs to a *hierarchy*, the query is delegated on the parent (step 3). If a partial solution has been found already, send it to the parent along with the query. All subsequent computation and communications are performed by the delegate, which will only reply either with the most suitable results or a negative answer. This corresponds to message 1 in Figure 1.

At this point (step 4), no ancestor in the hierarchy is aware of any complete result for the query. Therefore, it is necessary to ask to other known devices which may be closer to the desired services. We assume that services with similar characteristics are deployed in devices which usually know each other. For this reason, we calculate a tree distance (see [**?**]) between the query and each of the services known to the current node. The device where it is deployed the closest matching service is designated as the next device to consult (i.e., the next closest match). In this way, we achieve an *indirect* discovery mechanism. This indirect querying is exhaustive in the sense that, if no result is found from the designated device, the next one in the ranking is consulted until the current node finds results or runs out of alternatives, therefore obtaining *robustness* in the discovery.

Once another device is designated to carry on with the discovery, the current node can decide to delegate the discovery (step 4b) or to increase its local knowledge base (step 4a) depending on its current resources. This delegation is analogous to the hierarchy delegation explained in step 3. Alternatively, if the node decides to increase its knowledge base, it askt the designated device for the information of its own knowledge base. Then it can proceed with this new information and continue with step 4. In Figure 1, the discovey is delegated to the server in between scopes a, b and c by means of message 2. Additionally, message 3 and its reply correspond to a knowledge base update so that the server is now aware of the services in scope c.

Some queries demand more than one service as candidate result or *aggregated results*. For this reason, delegation (both to the parent node or to the next designated device) includes the partially matching services found so far. In addition, step 4 is repeated until the query is completed.

Every node with enough resources keeps track of the processed queries and a cache of its results in its store. This serves to support queries with *dynamic scope* where previously solved discovery queries are re-answered or notified of new coming services that match them (step 1). In this way, results are kept fresh and updated with new nodes and services.

Once the device which originated the query is satisfied and it is no longer interested in matching services, then it emits a *forget-query message* to the chain of delegates so that the query can be removed from their stores.

4 Discovery in a Middleware for IoT

This section presents the implementation of the service discovery as part of the services provided by FamiWare [4], which is a family of middlewares for IoT, ambient intelligence and pervasive systems.

4.1 FamiWare Feature Model

Considering the high variety of the hardware and software that constitute the IoT and pervasive systems, instead of developing a single middleware, FamiWare is defined as a family of middleware for this kind of systems. FamiWare uses the Software Product Line (SPL) [10] approach in order to characterise the inherent variability of the IoT domain by a Feature Model (FM).

An FM specifies which elements or features of the family of products are common and which are variable. Thereby, the features are classified as mandatory, optional, alternative OR and alternative XOR. In addition, it is possible to specify formal cross-tree constraints or dependencies between the features. A valid FM configuration corresponds to a set of features of the FM that satisfies the tree constraints and the cross-tree constraints.

The FamiWare feature model represents the following information: (i) the characteristics of the devices supported by the middleware, (ii) the services (with theirs corresponding versions) provided by the middleware, and (iii) the different routing protocols implemented in the middleware that can be used for the communication between devices. Figure 2 depicts a partial feature model with the most representative features considered in this paper, including the sensor static characteristics (e.g., Temp, Humidity, Mobile, MoteLike, etc.) and the discovery service.

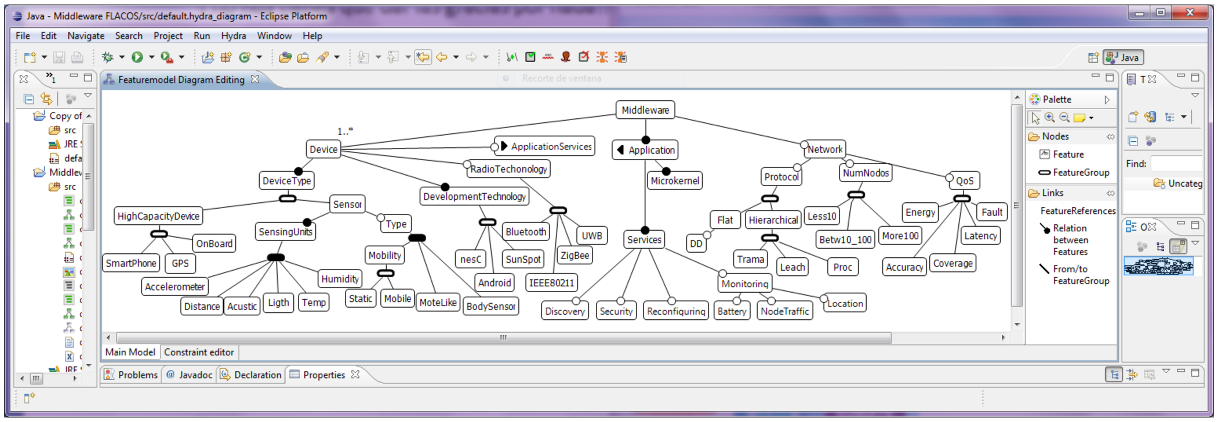

Figure 2: FamiWare Partial Feature Model

This feature model is the base of a model-driven process that derives a minimum configuration of the middleware adapted to the requirements of each network node. Then, FamiWare uses the feature model to customise the family and get middleware configurations at design-time. In the scope of this work, we propose to use this model also at run-time to discover the nodes by fulfilling the static characteristics of the contract. FamiWare makes use of Hydra tool for designing the FM and obtaining the configuration for every device automatically. In order to achieve these configurations, certain input parameters are given as initial constraints and Hydra generates the minimum valid configuration that satisfies those initial constraints.

4.2 Discovery in FamiWare

Once we have the FM configuration corresponding to all the devices of the network within the knowledge base, we can discover the nodes satisfying the query by providing to Hydra as input the query as initial constraints. In such a way, Hydra automatically returns a configuration containing those nodes that fulfill the query. For this, we have to transform the query of our contract specified in XQuery as initial constraints, i.e., regular expressions. This process corresponds to the checking of the static characteristics.

Example 2: Going back to our example, the static characteristics of the query represented in XQuery are transformed in an initial constraint related to the discovery of `mobile` `mote-like` sensors with capabilities `temperature` and `humidity`. This initial constraint is an regular expression as follows:

```
Temp and Humidity and MoteLike and Mobile;
```

Taking as input this initial constraint, Hydra returns all the nodes that contain these four features.

For the verification of dynamic characteristics, with the list of nodes provided by Hydra by fulfilling the static characteristics, we directly ask every node for its dynamic characteristics. To get this goal, monitoring services in FamiWare are executed in every node to give information about those characteristics. For instance, services monitoring the battery, the location, or the node traffic (see Figure 2).

Example 3: Specifically, in our scenario, the service monitoring the battery of every node given in the static phase should be queried in order to find out the first node fulfilling the required battery level, i.e., more than 50%, or the level closest to 50% if there is no sensor with the desired level.

As it was mentioned in Section 2.3, sometimes it will be require the reconfiguration of the sensor nodes in order to satisfy the contracts. FamiWare allows this reconfiguration process.

5 Concluding Remarks

We have presented a novel approach to tackle the discovery of devices or sensor nodes in the IoT domain by using contracts based on queries on a knowledge base. We take advantage of the feature model used in the FamiWare middleware as static information to represent the knowledge base. Our discovery process allows a negotiation at run-time by either relaxing the contracts or reconfiguring the devices of the system. In addition, discovery requests are remembered throughout the nodes involved in the process so that when new services appear, dynamic updates are sent to the interested node.

Our discovery approach has been inspired by current data diffusion and dynamic routing mechanisms in WSAN (e.g., [6, 7]) by extending and specialising them to serve for service discovery in Sensor Web. These protocols assume that the network overhead caused by control packages is neglectable in comparison to data packages. For this reason, data requests or advertisement messages are key in these approaches and actual data is only transferred when there is a real interest negotiated between a particular sink and source. Heidemann et al. [6] propose that, when a node requires certain information, it sends a query using message-flooding. Every node which receives that request, evaluates it against its provided information and, if there is a match, it floods that information as result for the request. The TinyDiffusion protocol [9] improves the idea in the following way. When the initial request is flooded, every node stores the first source where it receive the request from. In this way, whenever a node replies with the data, these heavy data messages are forwarded following a direct son-to-parent path, avoiding the data flooding used in the previous approach. In addition, requests are stored (with an expiration date) in every node so that new information that matches the request can be sent even after the request was issued. Compared to these works, we obtain a dynamic approach which is robust with regard to the ever-changing structure of the network; message-wise efficient, as it avoids any kind of broadcast or flooding; and hierarchy-aware, as it exploits both the hierarchy and the static self-information stored in the nodes.

As regards future work, we plan to further develop the ideas presented in this ongoing paper. In addition, we will evaluate the complexity of our approach, to formally demonstrate desirable properties (such as robustness) and illustrate the approach with real-wold examples.

References

[1] M. Botts, G. Percivall, C. Reed & J. Davidson (2007): *Sensor Web Enablement: Overview And High Level Architecture*. OGC White Paper, Open Geospatial Consortium.

[2] T. Erl (2005): *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. Prentice Hall.

[3] Commission of the European Communities (2009): *Internet of Things - An Action Plan for Europe*. Technical Report COM(2009) 278 Final.

[4] L. Fuentes & N. Gámez (2011): *FamiWare: A Family of Event-based Middleware for Ambient Intelligence*. *Personal and Ubiquitous Computing* 15(4), pp. 329 – 339.

[5] CAOSD Group (2010): *Hydra*. Available at `http://caosd.lcc.uma.es/spl/hydra/`.

[6] J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin & D. Ganesan (2001): *Building Efficient Wireless Sensor Networks with Low-Level Naming*. In: *Proc. of SOSP'01*, ACM, pp. 146–159.

[7] W.R. Heinzelman, J. Kulik & H. Balakrishnan (1999): *Adaptive Protocols for Information Dissemination in Wireless Sensor Networks*. In: *Proc. of MobiCom'99*, ACM, pp. 174–185.

[8] K. Lee, K. Kang & J. Lee (2002): *Concepts and Guidelines of Feature Modeling for Product Line Software Engineering*. In: *ICSR'02*, *Lecture Notes in Computer Science* 2319, Springer, pp. 62 – 77.

[9] M. Mysore, M. Golan, E. Osterweil, D. Estrin & M. Rahimi (2003): *TinyDiffusion in the Extensible Sensing System*. Available at `http://www.cens.ucla.edu/~mmysore/Design/OPP/`.

[10] K. Pohl, G. Böckle & F. Linden (2005): *Software Product Line Engineering - Foundations, Principles, and Technique*. Springer.

[11] A. Sheth, C. Henson & S.S. Sahoo (2008): *Semantic Sensor Web*. *IEEE Internet Computing* 12(4), pp. 78 – 83.

[12] P. Walmsley (2007): *XQuery*. O'Reilly Media.