Group Communication in Constrained Environments using CoAP-based Entities

Isam Ishaq, Jeroen Hoebeke, Floris Van den Abeele, Ingrid Moerman, Piet Demeester
Department of Information Technology (INTEC)
Ghent University – iMinds
Ghent, Belgium
{isam.ishaq, jeroen.hoebeke, floris.vandenabeele, ingrid.moerman, piet.demeester}@intec.ugent.be

***Abstract*— The Constrained Application Protocol (CoAP) is a new Internet protocol that is currently being standardized. CoAP allows access to the drastically increasing number of smart objects and their sensing resources from virtually anywhere. It is a light-weight protocol designed to cope with the restrictions imposed by the limited resources (CPU, memory, power,…) of many smart objects. Depending on the application, information from individual objects might not be sufficient, reliable, or useful. An application may need to aggregate and/or compare data from a group of objects in order to obtain accurate results. Although multicast may be used to transmit the same request to several objects, multicast communication with smart objects has some disadvantages. Programming individual requests is another solution but lacks flexibility and opportunities for reusability. In this paper we propose a novel CoAP-based approach for communication with a group of resources across multiple smart objects. This approach organizes the group of resources that should be accessed into a new CoAP resource, called an entity, and nicely integrates several important aspects of entity management: creation, validation, usage and manipulation. In order to demonstrate the feasibility of this approach we present an implementation and experimental validation.**

Keywords*— *Internet of Things; CoAP; sensors; wireless sensor networks; group communication; entities

I. INTRODUCTION

The Do-It-Yourself (DIY) movement is spreading beyond traditional domains, such as home painting, to more modern domains, such as programming. DIY programming gets especially interesting when it involves real-time data from the growing amount of smart objects with embedded sensors and when actuators can be triggered to perform real-world actions accordingly. It gets even more interesting and appealing when access to these smart objects can be obtained over the ubiquitous Internet – leading to what is now mostly known as the Internet-of-Things (IoT). However, these smart objects are typically optimized for low-power consumption and low-cost, are constrained in their resources (CPU, RAM, ROM…) and thus unable to run standard Internet protocols. The networks that connect these objects together are often referred to as low power and lossy networks (LLNs).

Recently a lot of effort has been made to develop open standards that cover many aspects of communication and access to smart objects. At the networking layer 6LoWPAN allows IPv6 communication with these objects through an adaptation layer. At the application layer standards are being prepared to allow access to these objects in a RESTful way, similar to how most information on the Internet is accessed. To this end the IETF established the Constrained RESTful Environments (CoRE) working group with the aim at realizing the REST architecture in a suitable form for the most constrained nodes and networks. CoRE is aimed at machine-to-machine (M2M) applications such as smart energy and building automation [1].

Typically, each of the constrained servers has at least one CoAP resource that may be queried by clients to obtain information about the smart objects themselves (e.g. battery level), about the environment that they monitor (e.g. temperature of the room), or to trigger the objects to perform real-world actions (switch the light on). Examples of resources include: “/temperature”, “/humidity”, “/room_location”, “/picture”, “/light_control”, etc. The discovery of these resources is essential in M2M application environments. For HTTP Web Servers, the discovery of resources is typically called Web Linking [2]. The use of Web Linking for the description and discovery of resources hosted by constrained web servers (CoAP or HTTP) is specified by the CoRE Link Format [3]. A well-known URI "/.well-known/core" is defined as a default entry-point for requesting the list of links with the resources hosted by a server. Once the list of available resources is obtained from the server, the client can send further requests to obtain the value of a certain resource.

Depending on the application, information from individual objects might not be sufficient, reliable, or useful. An application may need to aggregate and/or compare data from several nodes in order to obtain accurate results. In the same way, a single user request might need to trigger a series of actions on multiple actuators to perform a single user request. Although multicast may be used to transmit the same request to several objects, multicast communication in LLNs has some disadvantages. For instance, it is difficult to avoid duplication of messages, and duplication is undesirable in an LLN where bandwidth is limited for these constrained objects. Furthermore, basic multicast is not reliable in an LLN, which is problematic for requests that require guaranteed delivery. Also, the creation of multicast groups, defining which objects should be addressed when using a particular multicast address, is hard to realize inside LLNs. Additionally, the use of multicast increases the footprint of the code that needs to fit on

the constrained objects, and it is to be expected that this functionality will not be available in many LLNs.

As an alternative, unicast-based solutions should be considered. Some unicast-based features have been introduced to alleviate some of the problems above, but these features are insufficient. The current CoRE drafts do not foresee any unicast-based way to manipulate resources that are located on multiple smart objects with a single client request. To overcome this shortcoming and be able to perform such composite requests, intelligence is typically added to the client application to make it communicate with the smart objects individually. This leads to more complex user applications, and the added intelligence and programming cannot be shared with other applications easily. Furthermore, complex user applications may be unmanageable. Any modifications to those complex user applications may require significant testing time, thus limiting the flexibility of the user applications. Additionally a large overhead of communication between the client machine and the smart objects is generated, especially when many smart objects are involved in these actions. When the communication between the client and the smart objects is done across the Internet, delays are unpredictable and a sequence of actuator commands might arrive out of order and possibly have unwanted results. Furthermore, if the communication occurs over costly links, communication between the client and the smart objects might get unnecessarily expensive.

The discussed approaches are able to realize communication with a group of resources, but each exhibit some limitations. Therefore, in this paper we propose a novel CoAP-based approach for communication with a group of resources across multiple smart objects. First, we will briefly discuss some related work. Next, in section III we describe our approach in detail and evaluate it in section VI. Section V concludes this work.

II. Related Work

The group communication draft [4] discusses fundamentals and use cases for group communication patterns with CoAP. Building upon IPv6 multicast capabilities, the draft describes how CoAP should be used to interact with multiple smart objects. This approach exhibits the limitations as discussed above. Also multicasts are not useful when a single user action needs to trigger different sensor requests, since one multicast request delivers the same message to all group members.

Simple unicast solutions are defined in the CoRE Interfaces draft [5]. Among other interface types, this draft defines the Batch interface type and its extension, the Linked Batch interface type. Batch interfaces are used to manipulate a collection of sub-resources at the same time. Contrary to the basic Batch, which is a collection statically defined by the web server, a Linked Batch is dynamically controlled by a web client. A Linked Batch resource has no sub-resources. Instead the resources forming the batch are referenced using Web Linking [2] and the CoRE Link Format [3]. The draft does not foresee any way to manipulate resources that are located on multiple smart objects with a single client request.

An approach somewhat more similar to ours, also using the notion of an entity, has been presented in [6]. The aim here is to annotate real-world objects by using entities that are automatically created based on semantic information, which resides on the constrained devices. One problem of using semantics on constrained devices is that semantics can easily require a lot of memory that might not be available on the constrained devices. Further, in our approach users can create entities as required and we address important aspects related to entity validation and entity behavior.

To our knowledge, this is the only work that explores communication solutions for interacting with a group of CoAP-enabled constrained devices. Next to this, there exist other solutions to realize or improve multicast communication in Wireless Sensor Networks, such as [7], [8]. These solutions can alleviate some of the problems related to (reliable) multicasting, but their scope is different from the work presented here.

III. Approach

We aim to create an intermediately level of aggregation to be able to easily manipulate a group of resources across multiple smart objects. To avoid increasing the footprint of the constrained devices, we use the same technology as used to manipulate individual resources, i.e. CoAP, and extend it accordingly. Such a group of resources is called an entity and the entity can be used or manipulated through a single CoAP request. Similarly, the creation of an entity by a client is realized via a single CoAP request and includes a complete validation of the entity. Furthermore we propose the creation of profiles for the created entities. The use of entity profiles allows the client to specify in more detail how the entity should behave (e.g. if it should use confirmable or non-confirmable CoAP messages), and, through updating the profile, allows manipulation of this behavior. As such, we strive to combine ease of creation, ease of usage and flexibility in behavior into a complete solution for interacting with CoAP resources from different objects inside a LLN. By building upon being standardized concepts, the impact on the constrained devices is limited. In the following subsections we discuss the details of our approach.

A. System Overview

We call the component that manages the entities, the Entity Manager (EM). This component, which can reside e.g. on the Border Gateway of the LLN, is responsible for maintaining entities that are created from groups of CoAP servers (i.e. sensors and actuators) inside the LLN. Clients on the Internet can interact with an EM to create new entities and/or customize how these entities should behave. Optionally the client can elect to contact a resource directory [9] in order to discover which resources are available in the network. Fig. 1 shows an overview of the involved components.

The EM functionality does not have to be put on a dedicated device. Theoretically any CoAP server can be extended to become an EM. The choice of the most appropriate location to put the EM functionality depends on the size and topology of the network. For example, it can reside on a smart object in the

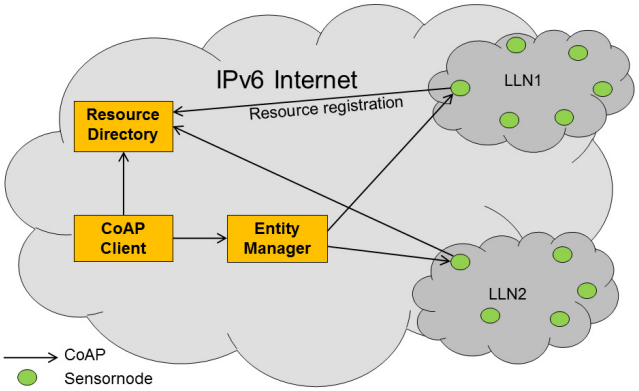

Fig. 1. Clients create entities consisting of several smart object resources on the Entity Manager. Clients can optionaly query a resource directory to discover the existence of the resources.

constrained network with enough resources, in the Cloud, on the client device itself, or on a gateway on the edge of the LLN. The latter case has the added benefit that security can be centrally managed besides offloading the processing from constrained devices.

Regardless of the location of the EM, it will serve as a "proxy" between the client and the constrained devices. Client requests will be sent to the EM, which will analyze and verify the requests and then issue the appropriate requests to the constrained devices using CoAP. Once the EM receives responses from the constrained devices, it will combine them according to the client needs and will send back an aggregated response to the client.

When a client tries to create a new entity consisting of a group of resources inside LLNs, the EM performs a sanity check on the request in order to make sure that the resulting entity would make sense. For example it verifies that the resources inside the entity are valid, if they support a certain content format or if their data can be aggregated. Customization of the entity behavior is accomplished by creating profiles for the entities. A profile of an entity can specify for example whether to return the values of all resources in the entity, only the computed average of all values or a subset of all values. Fig. 2 shows a high level structure of the Entity Manager. It shows that the EM contains two databases:

- Entity Database: In this database all entities are stored along with their profiles as defined by the user.
- Capabilities database: This optional database provides rules that can be used to match user requests with sensor capabilities. This can be as simple as translating a request for temperature in degrees Celsius while obtaining the data from as sensor that only supports Fahrenheit. It can also be more complex, e.g. converting from one content format into the other.

B. *Entity Creation*

To facilitate the creation and manipulation of entities, the Entity Manager offers a CoAP resource "/e". We call this resource the Entity Management Resource. This interface supports only the CoAP POST request method. As payload of

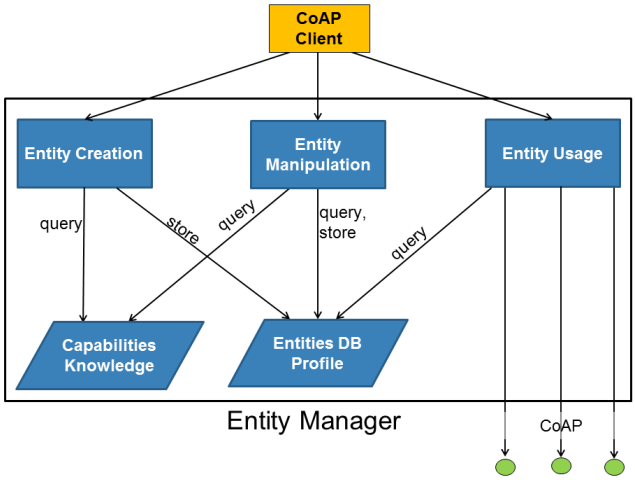

Fig. 2. Entity Manager high level structure.

the request, it expects a collection of resources in link CoRE format [3], which together should form the entity. In the response, the Location-Path CoAP option is used to specify the name of the newly created resource. In the current design, the payload of the response is in plain text and describes the results of the validation tests performed by the entity manager on the collection of resources.

When a client wants to create an entity consisting of several sub-resources, it has to compose a CoAP POST request and send it to the Entity Management resource on the Entity Manager. The EM creates the entity, assigns it a unique URI, and stores the entity in the entity database for future usage. Then the EM starts the entity validation process (explained in the next subsection). The client is informed about the URI to use in order to access or further customize the newly created entity and about the results of the validation of the entity. If the entity did not pass the validation process the client should fix any errors and resubmit the entity for validation again before the client can use the entity.

An example of the entity creation process is shown in Fig. 3. In this simple example the client requests the creation of an entity consisting of two sub-resources: coap://[Sen5]/tmp and coap://[Sen8]/tmp. The entity manger creates the new entity, assigns it the URI "/1" and informs the client about the newly created entity. From now on, any client can access the newly created entity by accessing the "/1" resource on the EM. Please note the validation process is not shown in Fig. 3 for simplicity.

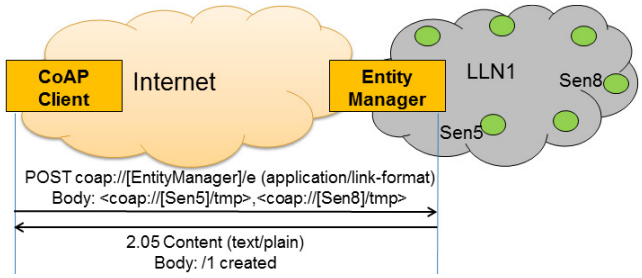

Fig. 3. A CoAP client requesting from an Entity Manger to create a new entity that contains two resources.

C. Validation Process

Whenever a client requests to create a new entity or to modify an existing entity, the EM performs a validation process. The purpose of this validation process is twofold: 1) Make sure that the sub-resources in the entity exist and can be used. 2) Derive the properties of the entity based on the properties of the sub-resources it contains. If the entity passes validation the EM marks the entity as a valid entity and stores the entity along with its calculated properties in the entity database for future usage. If the entity fails validation it is still created, but marked as invalid. The entity validation is based on EM's knowledge of the individual sub-resources and their profiles and based on the knowledge in the capabilities database as will be discussed in the next paragraphs.

Resource profiles can be used to express capabilities of a CoAP server and its resources [10]. Profiles are usually expressed in JSON format [11]. To briefly illustrate resource profiles let's assume that in Fig. 3 the temperature sensor at "coap://sen5.example/tmp" supports the "Uri-Host" (3), "ETag" (4), "Observe" (6), "Uri-Port" (7), "Uri-Path" (11) and "Content-Format" (12) CoAP options (op). This sensor further supports the "application/senml+json" (55) content format (cf) and the allowed method is GET (1). This will result in sen5 having the following profile:

```
Res: 2.05 Content (application/json)
{
  "profile":[
    {
      "path":"tmp",
      "op":[3,4,6,7,11,12],
      "cf":[55],
      "m":[1]
    }
  ]
}
```

1) Can the resources be used?

If the Entity Manager does not know any of the sub-resources in an entity (e.g. based on knowledge in a resource directory) or does not know the sub-resource capabilities, it tries to obtain this information according to a fallback mechanism as follows. First the EM tries to contact the object containing the resource in order to obtain the resource profile, since this would provide the most complete information about the resource. If the resource profile does not exist, the EM tries to obtain any information about this resource from /.well-known/core of the respective object. If this fails as well, the EM tries to query the resource directly to discover, as a minimum, if the resource exists or not. The validation process that the entity manger performs on entities is shown in a simplified form in Fig. 4. Basically the process ensures the following:

- The individual sub-resources contained in the entity are valid (e.g. the resources exist on the respective nodes).
- The requested operations can be performed on the individual sub-resources (e.g. which CoAP options are supported, which RESTful methods are allowed?).
- The individual sub-resources do not conflict. A sample conflict can occur when an entity creation request contains two sub-resources on the same actuator asking it to do contradictory actions, e.g. open and close at the same time.
- The responses sent by the individual sub-resources can be combined together using a common dominator or knowledge from the capabilities database

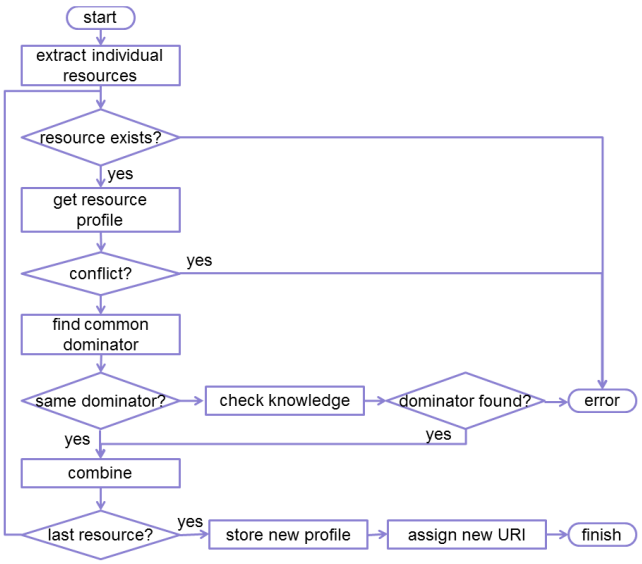

Fig. 4. Entity validation process flow.

2) Entity Profile

Once the EM knows all information about the sub-resources that should become part of the entity and once all necessary checks have passed, the EM creates a profile for the entity based on this information and the EM's capabilities database. To illustrate this let's further assume that the second temperature sensor in Fig. 3. "coap://sen8.example.org/tmp" supports the same options as sen5 except for the observe option. Only the GET method is allowed and the supported content formats on this sensor are "text/plain" (0) and "application/senml+json" (55). Thus sen8 will have the following profile:

```
Res: 2.05 Content (application/json)
{
  "profile":[
    {
      "path":"tmp",
      "op":[3,4,7,11,12],
      "cf":[0,55],
      "m":[1]
    }
  ]
}
```

Based on these two profiles the EM constructs a profile for the newly created entity. This profile contains information related to the resource itself, as described in [10]. In this example, this includes the options that are supported, the supported methods (only GET) and the content format "application/senml+json" (55). In addition, the profile is extended with an entity specific part, providing more information about the entity itself. The resulting profile of the entity looks as follows:

```
Res: 2.05 Content (application/json)
{
  "profile":[
    {
      "path":"1",
      "op":[3,4,7,11,12],
      "cf":[55],
      "m":[1]
    }
  ],
  "entity":[
    {
     "r":"coap://[Sen5]/tmp,coap://[Sen8]/tmp"
    }
  ]
}
```

This simple example illustrates how an entity profile is constructed; either based on information from individual resource profiles or based on information retrieved via other means such as resources attributes known from /.well-known/core. Much more information than shown here can be included and, by using a flexible representation format, the profile concept can be easily extended with new information.

D. Entity Usage

Once an entity is created the response contains the URI of the dynamically created resource name. The client can now interact with the entity by issuing a single CoAP request to the resource representing the entity. When a request for an entity arrives, the process flow as shown in Fig. 5 is executed. The EM breaks down the request into its components and sends the individual requests to the respective objects using unicast CoAP messages. It can either do that in parallel or sequentially. Once all needed answers are received, the EM creates a response to the client based on the individual responses and sends it to the client. Note that how many sub-resources should respond, how the response is composed and how it should look like can be changed be customizing the entity profile as will be explained later on. Fig. 6 shows an example of using the entity that was created previously in Fig. 3. The client issues a GET request on the entity's resource "/1". This results in the EM issuing two GET requests to the individual sub-resources, waiting for replies from both of them and then sending back both results in one combined response back to the client.

E. Entity Modification and Behaviour Manipulation

It is possible that a client wants to modify an entity after its creation. For example, a client could want to add new sub-resources to the collection of sub-resources in the entity or remove a number of sub-resources. Alternatively, the client could like to customize the behavior of an existing entity. The latter can include aspects such as the number or percentage of sub-resources that should respond before the entity manager replies to the client, the content format of the response, the operation (e.g. average, max, min, etc.) that should be performed on the results before sending them to the client, etc. Modifications to the entity or to its behavior can be made by updating the entity's profile and posting the updated information (PUT or POST) to coap://[EM]/.well-known/profile?path="[ENTITY_URI]", in which /.well-known/profile is a resource for accessing the profile of a

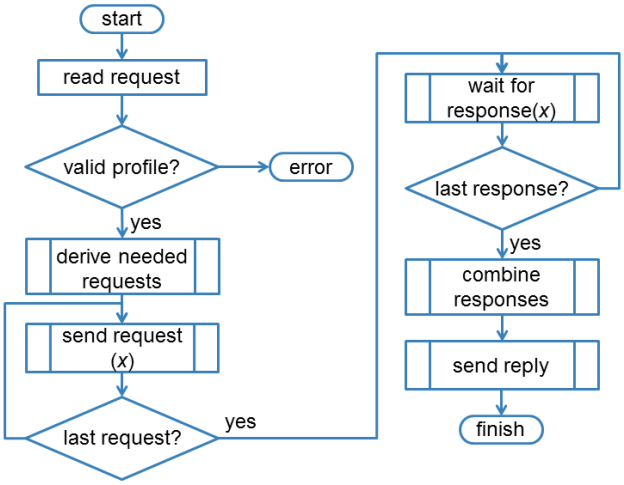

Fig. 5. Simplified entity usage process flow.

resource as described in [10] and ENTITY_URI the URI of the entity, e.g. "/1" in our example. When a client wants to modify the profile of an entity, this information is passed to the EM, which will validate the request and change the profile if the validation was successful. Finally, removing the entity can be realized by deleting the profile of an entity or, alternatively, by performing a DELETE request on the entity resource itself.

IV. Evaluation

In order to evaluate our approach we have implemented it using the CoAP++ framework [12]. The framework itself and the entity implementation on top of it have been realized in Click Router, a C++ based modular framework that can be used to realize any network packet processing functionality [13]. The framework's interoperability with other CoAP implementations has been formally tested by the European Telecommunications Standards Institute (ETSI), a non-profit standards organization, in two events called CoAP Plugtests [14]. At the moment, the functionality for creating, validating and using entities has been implemented as described above. Fig. 7 shows a screenshot demonstrating the result of sending a CoAP POST request to the Entity Manager to create an entity with five heterogeneous sub-resources. This request resulted in the creation of the entity with the URI "/2" and in the

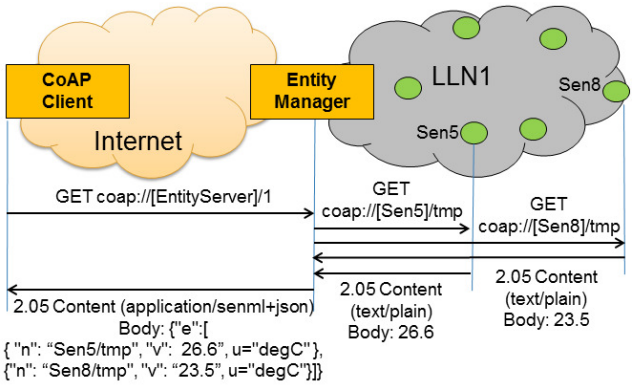

Fig. 6. A CoAP client requesting from an Entity Manger to obtain the values for the enity that was previosly created in Fig. 3.

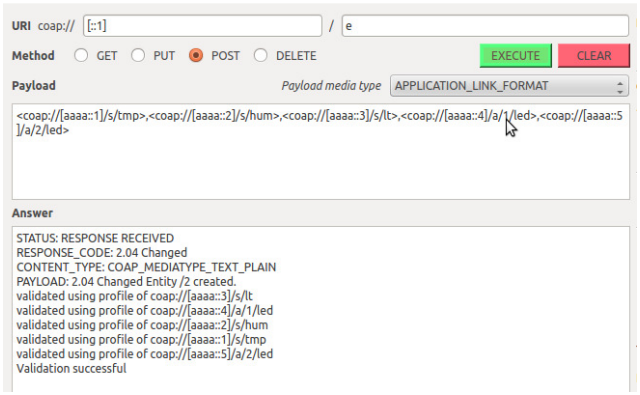

Fig. 7. Sending a CoAP POST request to the Entity Manager to create an entity with five sub-resource, resulted in the creation of the entity with the URI "/2" and in the validation of the entity.

validation of this entity by querying all sub-resources profiles. All complexity related to the creation and validation of the entity is hidden for the client and managed transparently by the EM. At this moment, the entity has been created and the client can use the newly created entity and interact with it by sending a single CoAP request to the entity resource. Fig. 8 shows such a client issuing a CoAP GET request to the newly created entity on the Entity Manager. The request ultimately results in a single reply from the EM, which combines the results of querying all five sub-resources of the entity. The client does not have to bother executing all individual requests, and processing the corresponding results.

V. CONCLUSION

In this paper we have presented a novel solution for interacting with a group of CoAP resources across multiple smart objects. It provides an interesting alternative to multicast-based solutions, which are challenging to realize in a constrained environment, and to application-based solutions, which simply program the required functionality. An Entity Manager, which can reside anywhere, turns groups of resources into entities. A strong point of our approach is that it nicely integrates the important aspects of entity management: creation, validation, usage and manipulation. At the side of the constrained devices, it requires no additional complexity, except support for profiles in order to realize more powerful validation. The introduction of entity profiles introduces a lot of flexibility and opportunities for further extensions regarding how entities should behave. We have implemented our proposal and demonstrated and validated its feasibility.

As future work, we foresee a detailed evaluation of our solution. We will explore several aspects (overhead, timing, scalability, etc.) related to the creation, validation and usage in realistic sensor networks and, if possible, compare it with existing multicast-based solutions. Also, additional ways to interact with entities, by extending the profiles, will be investigated. As such, we hope to realize a powerful enabler for group communication in LLNs and an interesting building block for IoT applications.

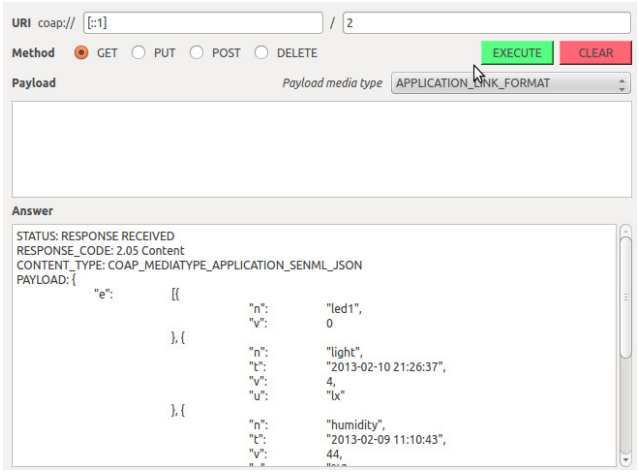

Fig. 8. Sending a CoAP GET request to the entity results in reply that combines the results of querying all sub-resources in the entity.

ACKNOWLEDGMENT

The CoAP++ framework was realized through funding by the European Union's Seventh Framework Programme under grant agreement n°258885 (SPITFIRE project).

REFERENCES

[1] "Constrained RESTful environments (core)," [Online], Available: http://datatracker.ietf.org/wg/core/ [Accessed: 28-Dec-2012].

[2] M. Nottingham, "Web linking," IETF, 2010.

[3] Z. Shelby, "Constrained RESTful environments (CoRE) link format," IETF, 2012.

[4] A. Rahman and E. Dijk, "Group communication for CoAP," draft-ietf-core-groupcomm-05, work in progress, IETF, 2013.

[5] Z. Shelby and M. Vial, "CoRE interfaces," draft-shelby-core-interfaces-03, work in progress, IETF, 2012.

[6] H. Hasemann, O. Kleine, A. Kroeller, and M. Leggieri, "Annotating real-world objects using semantic entities," in *EWSN 2013, the European Conference on Wireless Sensor Networks*, 2013, pp. 67–82.

[7] J. Hui and R. Kelsey, "Multicast protocol for low power and lossy networks (MPL)." IETF, 2013.

[8] S. Eswaran, A. Misra, F. Bergamaschi, and T. La Porta, "Utility-based bandwidth adaptation in mission-oriented wireless sensor networks," *ACM Transactions on Sensor Networks*, vol. 8, no. 2, pp. 1–26, 2012.

[9] Z. Shelby, S. Krco, and C. Bormann, "CoRE resource directory," draft-shelby-core-resource-directory-04, work in progress, IETF, 2012.

[10] B. Greevenbosch, J. Hoebeke, and I. Ishaq, "CoAP profile description Format," draft-greevenbosch-core-profile-description-01, work in progress, IETF, 2012.

[11] "JSON." [Online]. Available: http://www.json.org/. [Accessed: 11-Mar-2013].

[12] I. Ishaq, J. Hoebeke, J. Rossey, E. De Poorter, I. Moerman, and P. Demeester, "Facilitating sensor deployment, discovery and resource access using embedded web services," in *6th Int. Conf. on Innovative Mobile and Internet Services in Ubiquitous Computing*, 2012, pp. 717–724.

[13] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, Aug. 2000.

[14] C. Lerche, K. Hartke, and M. Kovatsch, "Industry adoption of the Internet of Things: A constrained application protocol survey," in *7th Int. Workshop on Service Oriented Architectures in Converging Networked Environments (SOCNE 2012)*, 2012.