Autonomous and self-controlling smart objects for the future internet

Marco E. Pérez Hernández, Stephan Reiff-Marganiec

Department of Computer Science
University of Leicester, UK
Email: {meph1,srm13}@le.ac.uk

Abstract—Development of IoT applications relies heavily on middleware architectures. Multiple solutions have been proposed using service and agent paradigms, which consequently impose a specific application development model. We observe a trend towards a data-feeder approach in which smart objects are simple data gatherers and senders. This approach reduces the autonomy of smart objects and brings privacy concerns about the data and service manipulation by remote applications and services. We propose a new approach for middleware and application development for smart objects based on web agents and services. Specifically, we integrate services as part of activities included in roles of a web agent allowing us to keep control of the service and data generated on-object, in contrast to delegating to remote applications or services. In addition, our P2P-inspired method for service discovery gives autonomy to smart objects since a central directory is not required to interact with others. We completed an initial implementation of our architecture and demonstrated its feasibility in a case study in the smart home domain.

Keywords—IoT Middleware; Web Agents; P2P Systems; Service Discovery; Smart Objects; Autonomy.

I. INTRODUCTION

Since the early stages of the IoT paradigm, middleware has been identified as key enabler. It aims to facilitate the development of IoT applications and provide useful abstractions to IoT developers. As it is natural, IoT middleware is being built on top of successful existing approaches, technologies and concepts.

Two fundamental approaches in IoT Development are Service orientation and the Agent paradigm. Service orientation has enabled the creation of IoT applications assembling data gathered from the physical environment while leveraging on the, almost unrestricted, cloud computing power. In the same way, agents have been useful to endow IoT applications with autonomy, reasoning and social abilities. Robust agent platforms have grown in controlled environments imposing particular models of integration with open environments such as the Internet. Consequently, several Agent-based IoT middleware solutions have inherited the same approaches.

In regards to proposed IoT middleware, a common approach we observe is that they are useful in developing applications when the data control and “smartness” of the objects resides outside it. This of course, ensures intensive use of powerful infrastructures but compromises on the ideal IoT requirements for privacy and autonomy. We think that another approach is possible. We seek for a balanced development of IoT applications, driven by a middleware that allows using Smart Object (SO) capabilities to keep control of data and services offered, while exploiting remote services under the direction of the SO.

To achieve this, we see the need of using an open approach for agent and service integration for the design of IoT middleware architectures. It should allow transparent use of services and the Internet infrastructure while taking advantage of abstractions provided by the agent paradigm. It implies also the definition of new approaches for search and selection of the IoT and SO services.

The main contributions of this paper are: (1) a new approach to use web agents and services to develop IoT SO applications. (2) A new approach to middleware architecture that keeps control of the data generated and the services offered, in the SO. (3) A new method for IoT service search and selection based in the Gnutella P2P protocol and using a map-reduce ranking-based selection.

The remaining is organized as follows: Firstly, we provide some background to understand our contributions. Secondly, we introduce our IoT middleware architecture and its components. Then, we explain the novel method for search and selection of services in the SO context. Later, we give details about our implementation. Next, we describe the case study which led our initial implementation effort. We present then some of the related work, and finally we offer relevant conclusions based on our findings.

II. BACKGROUND

A. Beyond Data Feeders

The Smart Objects are central entities of the IoT [1]. They are able to overcome the digital and physical frontiers by not only gathering data from their surroundings but also altering the existing situations and working together and autonomously with other Smart Objects.

Despite potential capabilities of Smart Objects, popular IoT middleware and platforms still use them as mere data feeders. They systematically send raw data to powerful nodes, which are in charge of running the logic and use the data in multiple applications or services. These objects usually enclose platform-specific functionality for interacting with the attached devices, which is offered to third parties via pre-programmed interfaces.
One problem of this approach is the displacement of the data control away from the Smart Object. This raises concerns about privacy, since the practices for handling the data and the final receiver are unknown to the Smart Object (and its owner). Besides, the coupling of the Smart Object to a specific application or remote platform, constraints the true potential of connecting to others autonomously.

The lack of resources in the SO push design decisions towards a data-feeder model. Under the idea of leveraging in cloud resources and services, things become “dumb” nodes. However, hardware platforms for SO are increasing resources every day, and now there are devices with processing power even better than some multi-purpose computers. This makes possible, an “intelligent leverage” in which core and minimal processes are held in the SO, while the use of cloud resources is led by the SO. Accordingly, IoT middleware can drive adaptation of SO processes to its resources and seek for a proper formula for cloud leveraging. For this, endowing SO with agent characteristics is a suitable technique that can coexist with service-orientation.

B. Agent and Services Integration

Service-Orientation and Agent paradigms have proven to be powerful and useful abstractions in multiple contexts. In the IoT middleware arena there has been a variety of works using these models. We consider pertinent work in section VII.

A broad distinction of the connection between Agents and Web Services was presented in [2]. From our review, we offer a distinction highlighting the role of the agent environment and the agent facilities e.g. life cycle, directory, etc. Web services are usually seen as external to the agent environments. As a result, integration efforts have addressed the way of using/publishing services through interfaces provided by the agent platform, as shown in figure 1(a) enclosed agent interface approach. Alternatively, agent platforms, MAS —Multi Agent Systems— or agent processes are wrapped as services that can be composed or orchestrated into more complex applications, as shown in figure 1(b) wrapped agent service approach. Since agent environments are usually provided by heavy platforms, a problem of these approaches is the dependency on a powerful node for hosting the agent platform facilities.

A more service-oriented approach, yet less exploited, is to expose the agents as a collection of services and so they use the Internet as the agent environment, as shown in figure 1(c). We call this open integration approach. Services define the behaviour of the agents, making them to benefit from the Internet resources and infrastructure. It gives agents more independence to consume services since they are no centrally controlled by a closed platform.

C. Challenges for Smart Object Middleware

We consider the open integration, the most suitable model when designing middleware for Smart Objects. The closer integration between agent and service oriented paradigms and technologies, boosts the autonomy of the SO and releases them from platform constraints. However, there are few solutions following this approach. Possibly, due to the fact that most popular and robust agent platforms drive naturally towards an enclosed agent interface approach and there is also a lack of robust web agent platforms.

Given the advantages of the mentioned approach, our first challenge is to define a new approach to an IoT middleware architecture, based on web agents using the Internet as environment and platform. IoT middleware requirements have been widely identified in previous works such as [3] and [4]. With the approach chosen, multiple requirements are covered transparently by the agent and service paradigms; however there are also multiple derived challenges. In this paper, we particularly aim to provide both: a Smart Object application development approach, based on web-agents and services; and a method for discovery and selection of SO services, using a P2P protocol and agents over the Internet.

III. PROPOSED APPROACH

A. Middleware Architecture

The main driver of the architecture is to boost the SO autonomy. Therefore, we exploit local capabilities and keep on-object control of services and data. To do so, the middleware provides a thin layer of agent facilities and gives access to delegable services hosted either in the cloud or the SO, according to local resources. The SO Controller keeps control on-object of the use of local or cloud services. The architecture is depicted in figure 2 and described below.
1) Communication Facilities (CF): It is based on web protocols (e.g. HTTP or COAP) for the Agent/Agent and Agent/Service communication.

2) Service Assembler (SA): A novel component that allocates the services required by the SO and assembles them according to either the role activities or the SO Controller (for delegable architectural services). It is the basis for adaptation, allowing for different configurations of on-object and cloud services, based on the SO resources. SA includes methods to search, select and use the needed services as explained in section IV.

3) Lightweight Agent Platform (LAP): It offers minimally: agent life cycle management, a communication protocol and abstractions. Adaptation and use of additional elements is possible in upper layers according to particular scenarios and available resources. The agent communication protocol relies on the CF, avoiding overloading by sharing the same infrastructure for both agent and service communication. It allows for transparent communication regardless of agent location. Instead of a centralized directory we use the SA and its mentioned search method.

4) Device Management Facilities (DMF): It is also proposed in other solutions. Here, it provides methods for adding, removing, restarting, reading, activating and deactivating sensors and actuators. These interfaces are standard, hiding specific functions provided by hardware manufacturers and when possible they rely on Operating System services.

Algorithm 1 High-level SO Controller Loop

```
while pendingGoals do
    if setUpRequired then
        checkDeviceChanges(devices);
        checkCapabilityChanges(localCapabilities);
        updateKnowledgeBase();
        performKBMaintenance();
    end if
    observe(propertiesOfInterest);
    updateKnowledgeBase();
    activities = reason(roles, observations);
    for all activity in activities do
        serviceContract = 
            searchServiceHosts(actionToExecute);
        callService(serviceContract, serviceArgs);
    end for
end while
```

5) Smart Object Controller (SOC): The main component, responsible for keeping the SO up and running according to goals specified and stored in the Knowledge base. It uses LAP and CF to send/receive messages to/from other SO components and to/from local or remote agents. It performs the cycle depicted in algorithm 1, which is an adaptation of the abstract agent loop proposed in [5].

6) Reasoning Engine (RE): As part of the decision-making process, it provides the intelligence to choose activities, services and providers (location). It queries the set of rules defined per each role. Then, it evaluates them according to observations; and finally, it obtains the activities to invoke. This component works mainly as a proxy, delegating the most of the intensive processing. It enables switching of local/remote services.

7) Capability Loader (CL): It loads the instances required to run the services demanded (either locally or remotely). Execution instances contain the information (service contracts, parameters and capability location) to dynamically—at runtime— instantiate or reuse service implementations.

8) Knowledge Base (KB): A distributed repository, based on semi-structured data storage. The domain model is inspired by the agent domain model [5] and OGC’s SensorThings data model\(^1\). It also stores the data resulting from SO observations. Due to the heterogeneous nature of the SOs and the amount of data generated, it follows a document-oriented approach that offers flexibility and scalability. Control over data and services is ensured through access policies, initially, defined at three levels: private, public and specific—a parametrized set of URIs.

\(^1\)http://ogc-iot.github.io/ogc-iot-api/datamodel.html
9) **KB and Storage Management (KBSM):** It provides centralized access to the KB and includes methods altering the KB’s contents and structure. It ensures the periodic replication and cleanup of the KB.

### B. Programming model

Definition of goals, roles, activities and rules is declarative while internal service programming does follow a control flow. Figure 3 shows the main concepts and relations of the model. Each role includes a set of conditions and activities to execute. Each activity has at least one action that specifies both the service description and the execution parameters. Every SO has a set of known capabilities, which are offered through a set of services. So, the service is the link between the capability and the action. Multiple agents and services can be added on top of the architecture, according to the SO resources.

### IV. Search, Selection and Use of SO Services

Activities are categorized according to the actions included. These categories are shared by the services. E.g. an “air-related” category, is common to activities and services related to ‘sense chemical properties’ or ‘measure pollution level’. When the SOC needs a particular service it invokes the SA, which aims to locate the most suitable service for the SO, based on the known service providers (defined in the KB) and the role activities.

#### A. Selection based in Map-Reduce

In an IoT scenario, multiple Smart Objects can provide common capabilities. Additionally, the service location is stored in distributed repositories —SO Knowledge bases— that must be analyzed in order to obtain the demanded service. Performing an efficient search and selection over huge data sets becomes a key aspect. The MapReduce programming model [6], is a simple but efficient way to search in data repositories with these characteristics. It requires the definition of the map and reduce functions that we approached as described below:

```json
{  
  "id": "service/192.168.1.5:8093/saf/safag/notifyBySMS",
  "rev": "16-54b871b965ea9904e4a39b84df4a78",
  "type": "service",
  "categories": ["alert-management"]
}
"name": "notifyBySMS",
"args": {
  "message": "java.lang.String",
  "target": "java.lang.String"
},
"result": "java.lang.String",
"capability": "capabilities.AlertManagement",
"host": {
  "url": "http://192.168.1.9:8093/shc/shcag/",
  "lastSuccess": "2015-01-07T22:32:42.943",
  "capability": "capabilities.AlertManagement",
  "ranking": "0.5",
  "local": "false"
}
```

![Figure 4. JSON Service Contract Example](image)

1) **Map phase** The search is based in the service contract stored in the KB (see figure 4). When the SO is executing an activity, it searches for services to carry out activity’s actions. If the service is known, it performs a simple search with the contract as filter. However, more extensive models, for example including soft and hard requirements on non-functional properties as proposed in [7], are also possible. The map function should emit one intermediate value for all the hosts (URI) where the demanded contract is present — according to the KB.

2) **Reduce phase** From the possible service implementations, the selection of the right one is based on a ranking calculated from further service attributes; e.g. the last date of usage or a combination of QoS parameters. The reduce function uses this ranking and returns the location of the demanded service.

#### B. P2P inspired service discovery

If the service is unknown by the SO, the strategy is to find providers to ask for the demanded service. Initially, providers are searched for using the category and working...
under the assumption that providers, having capabilities in a category, are more likely to know other hosts in the same category. The category matching is syntactic; however, the RE can be extended to a semantic mapping based on defined rules. If there are no hosts under the same category, arbitrary known hosts are taken. We conceive an unstructured overlay network of SOs and we use some of the primitives from the Gnutella protocols [8], such as query and query-hit, for conducting the search within it. It allowed us to define a fully decentralized model not requiring super nodes. A query method performs the search, described in section IV-A, locally and then asks other SOs.

Having the addresses (URI) of the identified providers, SOC makes a query call to the identified SO through the LAP and CF. When service is found, the SO replies to the query call with the address of the provider. Every SO filters services based on query parameters, ensuring that results satisfy required QoS. At the end, the requester’s KB is updated to include the new provider for the service and an initial ranking is calculated for it. Rankings are updated using the RE when circumstances of the provider vary. E.g. a failed attempt to use the service brings a reduction in the availability parameter of the service, so other higher-ranked and known services can be chosen. When the ranking decreases from a defined threshold, a new service query can be started or the action can be cancelled. These are decisions to be made by the RE.

When replying to query calls, SOs also check the access policy configured for the known services. If services are public, they can be included in every query-hit, otherwise they are only included if the requester object has the access granted by the provider. Service providers are the only ones entitled to modify these restrictions.

V. IMPLEMENTATION

A prototype has been developed following an incremental and iterative approach. We concluded an initial release led by the case study explained in section VI. It was focused on the core functionality of the architecture modules marked with a star (*) in figure 2. These modules were identified as mandatory for the case study. The implementation is based on the Java Platform, the EVE Agent Platform and CouchDB.

EVE [9], is a web-based agent platform that works on top of Jetty. Which is a “small, fast, embeddable web server and servlet container”2. EVE promotes a model in which agents reside in an open environment, it does not rely in a central directory facilitator unlike most FIPA implementations. The framework is also lighter than other platforms since it offers only an essential agent model focused on communication, task-scheduling and agent memory management. These features give us the flexibility to use our own agent/service search and selection method.

We extended the EVE platform by introducing a base agent with pre-defined features and actions common to all agents (e.g. loading the KB reference). We also included the implementation of the agent domain model (roles, activities, actions, etc.).

CouchDB implements a document-oriented repository using JSON documents. It allows for the definition of map-reduce functions in javascript and offer a REST interface for operations over the structure and the contents.

Although the Eve offers a CouchDB interface, we implemented a custom client to enable more flexibility in data and model operations.

For the inter-smart object communication, we rely on EVE capabilities. It wraps agent messages as JSON-RPCs over the offered transports. We worked with HTTP since it was the most robust choice available3. The SA deals with the service-

\[2\]http://www.eclipse.org/jetty/

\[3\]Our proposal does not fundamentally depend on HTTP, so other transport protocols can be employed.
oriented communication. We assume cloud services offer a REST interface and we implement a client supporting the main HTTP primitives. Both, agent communication platform and SA use the same HTTP transport.

VI. EVALUATION

We describe now the case study that led our initial implementation. It is based in a smart home scenario, where a Smart Air Freshener (SAF) cooperates with a Cleaning Cupboard (SCC) and with the Smart Home Controller (SHC). Table I shows the detailed roles of every participant. For this case, we focused on the SAF’s behaviour and goals. Middleware allows to enhance what SAF can do —considering its hardware or knowledge restrictions— by locating and enabling the use of others’ capabilities available as services. For SAF we use a Raspberry Pi 1 Model B, for SCC a Model B+ and for SHC a Laptop with a Core i5.

1) Smart Object Programming: We installed the first build of the middleware implementation. Middleware ensures that every object runs a web agent with an URL (i.e. IP + port + web-context + agentId). SAF was programmed with two roles: airFreshener and suppliesManager, defined as JSON documents in KB. The former ensures that fragrance is sprayed when surrounding movement is detected. The latter, observes the fragrance deposit to, proactively, ask for a refill when the specified threshold is reached. Agent interactions are shown in figures 5 and 6.

To achieve the goals, SAF discovers and uses services in SCC and SHC. Movement was simulated randomly and for the deposit level observation, we assumed 1 spray spent approximately 0.1 ml of fragrance, and a refill is 250 ml. So, an environment property in the KB was updated with every spray.

2) Service Search, Selection and Use: Known services by each participant at the beginning and end of the scenario are presented in table II. When SAF joins the network ($S_0$), it just knows its private services and the pre-defined service notifyByEmail provided by SHC. SAF uses it to find the other services it requires. The scenario starts with SAF spraying until reaching the deposit threshold when the suppliesManager role is triggered. Then, the agent sends a query request to SHC asking for the checkHomeAvailability service. SHC receives the query request and looks for the services locally. Note that some of the services are unavailable to SAF’s. SHC replies with the execution details of the checkHomeAvailability service. It considers the access policy, excluding private services from the reply.

SAF stores the execution details of the received service in its KB and call directly the provider. Service are consumed asynchronously. If provider is available, SAF calls the SHC’s notifyBySMS service to notify the house keeper in order to replace SAF’s refill. After it, SAF goes back to the airFreshener role until the refill is fully consumed. Then, it asks again to SHC, but this time it replies with 0 available refills. At this point, SAF decides how to deal with this situation, based on the suppliesManager role rules. It now searches for the service lookUpSupplyInMarket, sending a query message to known providers. SAF searches for new providers only when known ones are unavailable. Status of KB of each participant at the end of the scenario is shown in column $S_1$ of table II.

The implemented middleware allowed us to program the SO in a high-level declarative way. Once capabilities are implemented, these can be easy shared by other objects and included as actions in multiple role definitions. Both actuating and sensing capabilities are exposed as services, while the access policies provide an intuitive and simple way of control what to share with others. Regarding the service search method, the map-reduce query performs well when there are various services available, however our method had to be adapted to deal with cases when just few service are found. JSON-RPC over HTTP worked effectively, however other choices will be explored since the message becomes heavy for a complex service contract searching.
VII. RELATED WORK

IoT Middleware has been very active lately, we consider here the works more closely related to ours. Where possible we identify our understanding of the approach used, from those depicted in figure 1 and described in section II-B.

[10] was one of the first that envisioned a middleware where agents manage IoT resources. UBIWARE (a) aims offering interoperability between multiple enclosed platforms. It elaborates on the concepts of repository of roles and reusable behaviours to allow flexibility and adaptation in runtime. They propose the use of a P2P-based discovery facility as backup to the central directory of agents. In contrast, we rely completely in the P2P-based search, we use the SO as a central entity and agents are embedded in every SO.

[11] and [4] examine several middleware solutions. The former in the WoT field and the latter compares works for Smart Environments and Objects. Only ACOSO (a) [12], [13] and [14] exhibits an agent-based programming model. It is a JADE-based architecture and a discovery service for integration with the agent middleware. The architecture follows a master/slave model with a powerful node hosting the agent platform. It offers flexibility by the use of adapters for communication, device and KB management. Agent communication relies in ACL and the central Directory Facilitator. We use decentralized directory facilities, with different decoupled components but keeping control in the SO, additionally, KB and agent messages are formatted as JSON documents.

[15] proposes a lightweight approach (b). They include agent-based application-specific proxies to trigger services. They distribute the processing of data among distributed IoT devices. Localization of services relies on a resource directory while communication on HTTP and COAP. A gateway enables interaction with “external” services.

The concept of avatar —an agent representing the SO in the virtual world—, has been recently used in iotSilo [16] and ASAWoO (c) [17], [18]. The former is an Agent Service Platform, addressing the management of heterogeneity in devices and offering communication through a message bus. The latter is ongoing project expected to finish by the end of 2017, it aims “to enhance appliance integration into the web”4. A wide scope includes requirements such as interoperability, disconnection tolerance, safety, etc.

Both iotSilo and ASAWoO offer adaptation. In iotSilo it is possible using Portable-Service abstractions and dependency-injection patterns; whereas in the latter it is based on a deployment manager and the OGSi framework. In [16], agents are located away from the object and depend on an agent manager. [18] envisions an avatar architecture that can be either hosted by powerful objects or distributed along constrained-objects and the cloud. In theory, any module can be located out of the object and there is no difference on it, in regards its location. In contrast, our architecture is conceived to retain on-object the components ensuring control of data and processing. We favoured a simple agent platform and the identification of components that can be delegated to the cloud. The SO requires some minimum features. Our work is targeted to SOs from level 2 (Networked) according to levels reported in [19]. Indeed, the middleware is intended to enable development of capabilities of a higher level.

The prevalent approach for agent-service integration is the one in figure 1(a). In most cases, the control of the SO and its data is fully or partially carried out away the SO. Besides, SO autonomy is usually constrained by the dependency on either a (central) directory or a message bus.

VIII. CONCLUSIONS AND FUTURE WORK

Multiple approaches have been used to agent/service integration for development of IoT middleware. We identified three basic approaches: (a) enclosed agent interface, (b) wrapped agent service and (c) open integration. We consider the latter one the most suitable for a SO middleware in order to boost autonomy and keeping control of data and services.

Table II
CASE STUDY: S.O. AGENT’S KNOWLEDGE BASE STAGES

| Service                  | Arguments                  | Provider | Access Policy | SAF  | SHC | SCC |
|-------------------------|----------------------------|----------|---------------|------|-----|-----|
| spray                   | None                       | SAF      | private       | X    | X   |     |
| updateDepositLevel      | newLevel                   | SAF      | private       | X    | X   |     |
| notifyByEmail           | to, subject, message       | SHC      | specific (home)| X   | X   | X   |
| notifyBySMS             | to, message                | SHC      | specific (home)| X   | X   |     |
| lookUpNearbyStores      | model, location            | SCC      | public        | X   | X   | X   |
| checkHomeAvailability   | model, location            | SCC      | specific (home)| X   | X   | X   |
| checkHomeAvailability   | model, location            | External Object | public    | X   |     |     |
| checkHomeAvailability   | model, location            | External Object | specific (no SAF)| X| X |     |
| checkHomeAvailability   | model, location            | SHC      | private       | X   | X   |     |

S0: KB’s Initial State. S1: KB Final State (including discovered services).

4https://liris.cnrs.fr/asawoo/doku.php?id=homepage
We presented a novel approach for IoT middleware architecture, following an open integration approach. Our architecture is based in lightweight agent facilities and services. It allows performing essential processes in the SO, while using external services for process-intensive activities such as reasoning. This approach allows us to exploit SO resources and delegate work to third-party providers when required. We establish access policies for gathered data and services in order to keep control with the SO.

The challenge of using an open integration approach implied the use of non-traditional directory facilities trying to leverage the existing Internet infrastructure. We proposed a method for service search, selection and use based on map-reduce functions and inspired by Gnutella protocols for discovery of service providers.

We demonstrated the feasibility of our approach by implementing a middleware and IoT application using it, solving a problem in the Smart Home domain. A Smart Air Freshener, was programmed with high-level roles and activities, can rely on other SO’s capabilities to achieve its own goals. It does not need a central directory to find other SOs. We are extending the initial implementation of our architecture with the aim of establishing a testbed that allow us to improve our design. We will also add to the case study seeking to implement other participant SOs and expose SO’s capabilities with other purposes. E.g. SAF movement detection can be used by the SHC for alarm triggering. Currently, in service queries, we limited the use of the RE to keep this process light and simple. However, we will explore the use of this component to give support to complex queries and possibly service mashups. In addition, we will look to enhance the RE to take more advantage of SO’s descriptions and to switch between available services based on different criteria.

REFERENCES

[1] G. Kortuem, F. Kawser, D. Fitzon, and V. Sundramoorthy, “Smart objects as building blocks for the internet of things,” Internet Computing, IEEE, vol. 14, no. 1, pp. 44–51, 2010.

[2] S. Haddad, A. El, and F. Seghrouchni, “Web-MASI : Multi-Agent Systems Interoperability,” in 2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology(IAT’05), 2005, pp. 2–5.

[3] S. Bandopadhyay, M. Sengupta, S. Maiti, and S. Dutta, “Role of middleware for internet of things: A study,” International Journal of Computer Science & Engineering Survey, vol. 2, no. 3, pp. 94–105, Aug. 2011.

[4] G. Fortino, A. Guerrieri, W. Russo, and C. Savaglio, “Middleware for smart objects and smart environments: overview and comparison,” in Internet of Things Based on Smart Objects. Springer, 2014, pp. 1–27.

[5] L. Sterling and K. Taveter, The Art of Agent-Oriented Modeling. The MIT Press, 2009, vol. 47, no. 06.

[6] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters,” Communications of the ACM, vol. 51, no. 1, pp. 107–113, 2008.

[7] S. Reiff-Marganiec, H. Q. Yu, and M. Tilly, “Service selection based on non-functional properties,” in Service-Oriented Computing-ICSO 2007 Workshops. Springer, 2009, pp. 128–138.

[8] I. Ivkovic, “Improving gnutella protocol: Protocol analysis and research proposals,” Prize-Winning Paper for LimeWire Gnutella Research Contest, 2001.

[9] J. D. Jong, L. Stellingwerff, and G. E. Puzienza, “Eve: A Novel Open-Source Web-Based Agent Platform,” in 2013 IEEE International Conference on Systems, Man, and Cybernetics. IEEE, Oct. 2013, pp. 1537–1541.

[10] A. Katasonov, O. Kaykova, O. Khiyenko, S. Nikitin, and V. Terziyan, “Smart Semantic Middleware for the Internet of Things,” in Fifth International Conference on Informatics in Control, Automation and Robotics, 2008, pp. 169–178.

[11] F. C. Delicato, P. F. Pires, and T. Batista, Middleware Solutions for the Internet of Things. Springer, 2013.

[12] G. Fortino, A. Guerrieri, and W. Russo, “Agent-oriented smart objects development,” in Computer Supported Cooperative Work in Design (CSCWD), 2012 IEEE 16th International Conference on. IEEE, 2012, pp. 907–912.

[13] G. Fortino, A. Guerrieri, M. Lacopo, M. Lucia, and W. Russo, “An agent-based middleware for cooperating smart objects,” in Highlights on Practical Applications of Agents and Multi-Agent Systems. Springer, 2013, pp. 387–398.

[14] G. Fortino, M. Lackovic, W. Russo, and P. Trunfio, “A discovery service for smart objects over an agent-based middleware,” in Internet and Distributed Computing Systems. Springer, 2013, pp. 281–293.

[15] T. Leppänä and J. Riekki, “A lightweight agent-based architecture for the Internet of Things,” in IEICE workshop on Smart Sensing, Wireless Communications, and Human Probes, 2013, pp. 2–4. [Online]. Available: http://www.greenorbs.org/events/IEICE13.html

[16] E. Jung, I. Cho, and S. M. Kang, “iotSilo: The Agent Service Platform Supporting Dynamic Behavior Assembly for Resolving the Heterogeneity of IoT,” International Journal of Distributed Sensor Networks, vol. 2014, pp. 1–11, 2014.

[17] J.-p. Jamont, M. Lionel, and M. Mrissa, “A Web-Based Agent-Oriented Approach to Address Heterogeneity in Cooperative Embedded Systems,” Advances in Intelligent Systems and Computing. Trends in Practical Applications of Heterogeneous Multi-Agent Systems. The PAAMS Collection, pp. 45–52, 2014.

[18] M. Mrissa and N. L. Sommer, “An Avatar Architecture for the Web of Things,” IEEE Internet Computing, 2015.

[19] M. E. Perez Hernandez and S. Reiff-Marganiec, “Classifying smart objects using capabilities,” in 2014 International Conference on Smart Computing (SMARTCOMP). IEEE, 2014, pp. 309–316.