Towards a software framework for the autonomous internet of things

Marco E. Pérez Hernández, Stephan Reiff-Marganiec
Department of Computer Science
University of Leicester, UK
Email: {meph1,srm13}@le.ac.uk

***Abstract*—IoT promises a world where Smart Objects (SO) are able to autonomously communicate and work together to make the (human) user's life easier. Popular approaches for development of IoT applications take for granted that on-object resources are evenly constrained. As consequence, we observe a trend towards a data-feeder architecture in which "smart objects" are simple data gatherers and senders. Raw data is stored and processed in cloud platforms feeding web applications and services. The autonomy of SO is then compromised as they are not able to operate without these platforms. We propose a framework and architecture for the development of IoT applications where smart objects exhibit autonomy in regards platforms and human users. We completed the successful evaluation of our proposal with the implementation of a prototype and the execution of a use case for physical resources provisioning.**

***Index Terms*—IoT Framework; IoT Software Architecture; Web Agents; Smart Objects; Autonomous IoT.**

I. Introduction

The future internet brings the promise of smart devices exhibiting some degree of intelligence and autonomy. In a first stage, the research community have been working actively to tackle the open challenges of device connectivity, interoperability, data gathering and processing, among others. The next stage implies supporting on these advances to build truly autonomous IoT systems.

IoT Software frameworks become crucial for this purpose. They provide blueprints and models for the development of IoT applications and allow for sharing expertise in the shape of behavioural and structural solutions. Development frameworks support the design of IoT software architectures ready to fit all the pieces of the puzzle, namely: heterogeneous cyber-physical objects, sensors, networks, traditional systems and cloud resources; with the challenge of keeping the human users in the centre of any solution.

One desired feature of IoT applications is to exploit the resources (local, remote, cloud) and take advantage of the unique situation of the distributed devices within the smart environments. In doing so, however, there is a risk that applications end up sub utilising device capacity and losing the expected autonomy by fully relying on platforms and remote resources.

Smart objects must exhibit functionality to cooperate with others and use remote resources but also they must keep control of their key data and functions, make their own decisions and take the most of the advantage of local capabilities. The goal of this paper is to propose a software framework aimed at enabling the development of IoT applications with these features. We work with agents and services as the building blocks of autonomous smart objects.

The main contributions of this paper are: (1) a novel approach for IoT applications that makes intensive use of local resources;(2) An IoT software architecture providing autonomy not only from human user perspective but also from remote platforms/gateways. (3) Demonstration of an enhanced use of agent concepts in the context of IoT applications compared to state of the art solutions.

The remainder of this paper is organised as follows: First, we bring some context about the gap in IoT applications we are tackling. Second, we describe our proposal including the framework an IoT software architecture. Then, we describe our implementation approach. Next, we detail the evaluations carried out and we discuss the findings. We finally offer the relevant conclusions and our future lines of work.

II. Background

Well known challenges in the IoT arena include heterogeneity and autonomy. Multiple solutions have been proposed to deal with heterogeneity at different levels and to endow autonomy to the Smart Objects (SO). We review some of the pertinent works in section VI. Service Oriented Architectures (SOA) and Agent paradigms, have been popular approaches for dealing with these challenges. SOA brings, among others, modularity, reusability and helps to reduce the problem of heterogeneity to the interoperability of the nodes. In the other side, agents and Multi-agent Systems (MAS) are also used to provide abstraction over SO differences and are the preferred choice to endow autonomy. We share the view that these are mature-enough tools for tackling aforementioned challenges, however we note the lack of a consistent approach to use them inline with the IoT common requirements.

The per se diversity of the IoT scenarios, requires an approach for the use of MAS and services in IoT solutions, avoiding the incorporation of even more complexity. A consolidated view beyond adding interfaces or translation entities into the systems architecture, would bypass the heterogeneity of two different programming models. Classifications of the existing approaches for agent/service integration were attempted for example in [1] and [2]. They made evident, the need of a joint perspective building over every paradigm's

strengths. While the former work indeed propose an approach, from the industrial automation field, the latter gives attention to the IoT middleware. Although attempts have been made to use both models in middleware or specific solutions, to the best of our knowledge, there is no conceptual framework to guide the IoT application development using agent and services from a consolidated perspective.

IoT solutions require the adequate exploitation of both the internet resources and the ones from the SOs to provide services suitable to user demands in regards proximity, reliability, timing and privacy among others. The popular architectural approaches favour the sub-exploitation of SO's resources. One way to ensure uniformity is to pair SO to the lower capacity. It is common, that IoT platforms conceive SOs as nodes that are just able to gather and distribute raw data and actuate on the environment upon request. It is clear that IoT devices are constrained, but they are not equally constrained and SOs need to take advantage of the most of resources available on object to truly offer smart capabilities beyond connectivity, digital representation and access to data and standard operations.

As more IoT connected devices are quickly permeating homes, offices, cities and every physical environment, human users are facing the need to deal with the configuration/programming of many devices, every brand with its own ad hoc operating options, models and environments. One way to tackle this heterogeneity, is to promote, from the engineering of the IoT systems, an uniform operating model. In the same way that operating systems provide a set of standards functionality to users, with abstraction of the hardware platform, this uniform model for SOs can ease the task of programming/configuring multiple devices when required.

It is necessary to rethink how IoT solutions are built and the role of the SO to be able to attain more responsibilities to it, establish frontiers between what is done on-object and what is done remotely and ease its programming. Services and agents provide rich tools to tackle the mentioned challenges. Of course, this requires a minimum of hardware capacity in the SO, but it will promote higher utilisation of local resources. As a collateral advantage, this can allow users to gain control over tasks and data that, so far, are handled remotely with its advantages and also concerns.

III. Overall approach

A. Software Framework for Autonomous Things

The proposed framework is intended to serve as blueprint for the software engineering of IoT applications and middleware for Smart Objects as autonomous systems. It provides both concepts and an architecture that fit them all for the development of IoT applications based on agents and services. We built over existing concepts and principles from agent and cognitive systems literature [3][4][5] [6].

Since SOs are embedded in a cyber physical environment, they are not merely software agents. The physical dimension and the localisation of the software functions have impact in the autonomy and these are aspects to be considered in SO's software architecture.

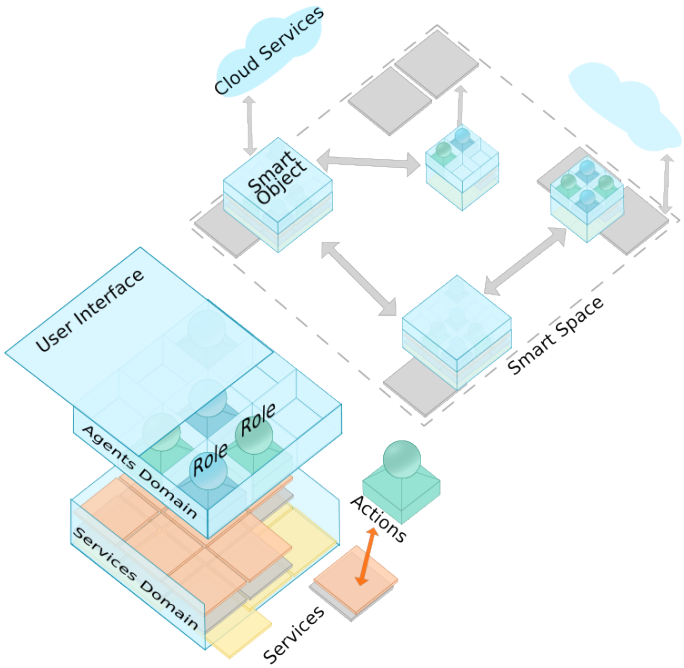

Fig. 1. Conceptual view of the Software Architecture for Smart Objects

The framework is intended to drive the development of IoT applications by addressing the heterogeneity and autonomy of SOs. It aims to enable encapsulation of platform-dependent functions and communication protocols through the service abstraction. In addition, the IoT application is built over a set of minimum functions that can be extended according to different SO configurations. we address autonomy with regard not only to the human user, but to other SOs and platforms. We adopt the view that autonomy depends on three factors: Sensing, Actuating and Reasoning. These functions can be either direct, if localised on-object, or indirect if localised in others (SOs, platforms, etc.).

We propose a decentralised approach for engineering of IoT applications based on autonomous SO. This entity is a cyber-physical object that exhibits autonomous behaviour based on the existing and available resources, capabilities and the programmed activities. The autonomous SO is able to keep up and running even if remote and powerful nodes (e.g. gateways) or platforms are not available. In contrast to other devices in the IoT spaces, autonomous SO not only offer basic features (e.g. connectivity or digital access), these SOs have control of the data they gather and are able to make decisions about what to do, when, where and how in behalf of the human users. Decisions might be regarding their own maintenance operations and also about what is expected from them in fulfilment of their intended purpose.

A graphical representation of the proposed architecture is depicted in figure 1. We observe that as multi-purpose computers are able to run multiple applications based on their hardware resources, SOs play different roles based on

the services they can offer. Therefore, conceptually SOs are boxes where roles can be either "installed" or "uninstalled" to achieve a particular behaviour in pursuit of some goals. The services are just wrappings of SO's cyber physical capabilities, that enable modular development. Although there are common roles the SOs can play, each SO has specific roles according to its particular capabilities, hardware platform and purpose.

The software for autonomous SOs is organised around a set of management functions which provide the infrastructure for the development and operation of the SO. From these functions, services are built to encapsulate particular functionality provided by the SO and to extend the infrastructure functions. On top of these services, an agents layer is intended to provide a framework for programming SO operation through configuration of knowledge items including representations for roles, activities and actions among others. Infrastructure functions and layers are described below.

B. Infrastructure Management Functions

Service Management *:* These functions cover the basic service life cycle operations from the registry, discovery, selection, binding and execution. For the sake of autonomy, service discovery is intended to be decentralised.

SO Agent Management *:* These functions give support to the agent layer enabling the SO management and interfacing with services. It includes operations for the initialisation of the SO, including the loading of all the roles playable by the SO as well as the available capabilities (services). In addition, it provides operations for the discovery and selection of goals and roles and the triggering of the ad hoc activities. For the sake of autonomy, role discovery is intended to be decentralised.

KB Management *:* These include the configuration of SO's Knowledge base (KB) and operations for enabling addition, update and removal of knowledge items. These functions also convert data gathered from sensing services into knowledge items exploitable by reasoning services. Knowledge items include conceptual representations of, for example, services, actions, roles or neighbour SOs.

C. Services Layer

SOs wrap capabilities as services that can be consumed either locally or remotely. The SO is both service consumer and provider. If a service is device-dependant, it includes functionality and routines to carry out the device management including the configuration (e.g. sensor, actuators or hardware interfaces) relying on the manufacturers APIs. Each SO has a local path where implementation of the services are stored. On SO booting, the management functions query and generate knowledge items (documents) with the interfaces for each service. These documents are independent of the programming language of the implementation. This mechanism enables decoupling of concrete action implementations from the overall SO sytem behaviour. Multiple SO behaviors can be programmed without changing services at this level and the same services can be easy deployed by copying the implementation in a SO with the required hardware configuration (e.g. sensor or actuator). Services also wrap protocols and message processing for communication on top of supported network interfaces and protocols.

The proposed service layer of the SOs is a mix of common and specific services. Each mix is particular to the SOs capacity, with a set of minimal common services. The more the hardware resources in the SO, the greater the quantity of services or the complexity of them. Adding new capabilities becomes a process of deploying, on-object, new services that wrap the operations supporting the capability. As result, the SO becomes compact and able, not only to make their own decisions, but also to carry out fundamental tasks in line with these decisions.

Communication Services*:* These services include operations for the configuration of network and human interfaces and the processing required for receiving or sending messages. Network-aimed services receive as arguments: a transmitter, the message type, the message contents and a receiver; and using the supported protocols, they build (disassemble) the message and carry out the transmission (reception) using the configured interfaces. These services ensure at least support to bidirectional communication with other SO or remote systems. Human-aimed services enable the SO to receive/send messages from/to human users, while the communication principle is the same that the Network-aimed, the output of these services must be complemented with specific services for the delivery/recognition of the message. For example, a message can be delivered/recognised through a display/touch screen or a speech synthesizer/recogniser, in either case it is constrained by the hardware platform and would require a specific process which is out of the scope of the communication services. Communication services enable SO to indirectly sense and act, which is enough in certain IoT scenarios, e.g. when sensors are offered by other Smart Objects.

Decision-making (Reasoning) Services*:* Multiple approaches exist for reasoning (e.g. theoretical, practical, rule-based, ontology-based, case-based, probabilistic, etc) and so multiple services could perform different kinds of reasoning based on user-defined rules, preferences, data sensed or defined knowledge models. Although they are resource intensive, reasoning services are essential for SO autonomy which imposes the requirement of at least a minimum of them be localised on-object. The service approach allows definition of basic reasoning services giving support to a (enough although non-optimal) local decision-making, that can be extended with remote services.

Actuating Services*:* These services include functionality for modifying the status of (resources and properties of) the SO's and the Smart space (SS). This functionality depends on the hardware platform and the specific nature of the object. Actuating on other SOs or human users is only possible indirectly through communication services. Autonomy of SO is increased when the actuating services provided by the SO allow control of the SO's properties of interest. e.g Would be

desirable that a microwave oven controls the oven temperature based on the conditions of the meal instead of an fixed pre defined time. For that to be possible, the oven should be capable of either decrease or increase the temperature when required.

Sensing Services: Sensing services enable the SO to be aware of an updated view of itself and the environment which is needed for a proper decision-making. These services include configuration (based on manufacturers APIs) and operations for data reading from sensors. Autonomy of the Smart Object is as constrained by the actuating services as it is by the sensing ones. Continuing with the microwave oven example, these sensing services enable the oven to read meal temperature, appearance or smoke presence on-object, and then enable required actions without relying on connection health or remote platforms availability.

Management Services: These services extend the basic infrastructure functions offering common operations for different SOs that can afford it. Some of these services include:

- *Monitoring Services:* Watch over the operation of the attached devices and hardware platform of both physical and digital resources.
- *Cost Management Services:* Calculate and monitor cost of services in terms of resources used, transform this information into knowledge items.
- *Continuity Services:* Manage replication and backup of KB.
- *Security Services:* Provide operations for ensuring data integrity and proper access to resources.

D. Agents Layer

We built this layer over the existing agent theory [5] [7]. A software agent gives digital identity to the SO and provides the standard behaviour of checking pending goals and work to achieve them. To do so, it uses communication and sensing services to update status from itself, the environment, other SOs and user preferences. With this updated view, reasoning services are used to identify actions to trigger which are calls to actuating and communication services with knowledge items as arguments. This layer is aimed to provide abstractions for a light programming based on the definition of:

Properties of Interest: These are a narrowed subset of the relevant properties from the environment (Smart Space) and the agents (SOs, human users or other software agents) of a particular IoT scenario. The status of these properties is sensed directly through sensing services or indirectly shared as messages received by other agents. Similarly, actuating services are used to directly modify some of these properties, but for others it is only possible to asks for an update through communication services. These properties are the basis for defining goals, trigger conditions, activity inputs and outputs.

Goals: Goals are defined as target states of the properties of interest. Multiple roles can be set to a SO establishing hierarchy relations between them.

Roles: They group a set of activities the SO can perform for a scenario. A role specification is common for all SOs within a particular smart space. Therefore, any SO able to play an specific role will carry out the same activities. This implies that, in some scenarios it is not necessary that a particular SO be available but any SO playing the required role can carry them out as long as it has the required input knowledge. It is also foreseeable that in some scenarios, although two SOs can play the same role, it is required that a particular one take over one activity, that should be specified as conditions for the activity. Properties of interest, enable a general specification of roles, providing a common behaviour playable for multiple SOs in diverse scenarios differing only in the properties checked. For example, the *SO Resource Manager* role includes activities for resources monitoring and notification to the human users (or other SOs) about the lack of SO's resources and asking for replacements; or the *SS Resource Manager* role can include the same activities but applied to smart space resources.

Activities: These include the intended behaviour of the agent playing a role. Activity specifications include the inputs, triggers and outputs in terms of knowledge items (e.g. properties of interest) and the actions. The same activity can be implemented in two different SOs using different actions (according to their capabilities). At this level, it does not matter how the action is actually implemented, as long as the output is the expected according to the specification. The output of an activity includes the effect it has on a knowledge item. e.g. an activity of SO can have the effect " increase" of the property of interest: "lighting" in a room. Since, ultimately activities are composed by a set of services, one activity can be seen as a service orchestration, where Agent Management functions include a single point of control based on the sequence and dependencies defined within the activity. Data flow is also ensured by the definition of the input and output knowledge as pre and pos conditions of each activity.

Actions: Actions are the glue between services and agent activities. They are atomic an specify a particular use of the available services. An action include the arguments to pass in to the service in terms of knowledge items. An activity can be dependent on updated observations of the properties of interest or from knowledge base. For example, sending a message to other SO, requires the selection and use of a communication service. The asynchronous nature of an agent's message is implemented through services, sending and receiving the message in source and target SOs respectively. The message itself in this example, is composed by data gathered through sensing services and also from pre-configured rules in Knowledge base.

Scenarios: These describe the activities (from different roles) to be carried out in order to achieve a particular goal. The scenario is a ordered list of activities from different roles that can be defined either by the human users or calculated by the SO. Definition of scenarios by human users is similar to defining a library of agent plans, they are loaded in knowledge base either in design or runtime. If scenario is not pre-defined,

```
{
    "_id": "activity/keepAirFresh",
    "type": "activity",
    "categories":
    [
    ],
    "input": {
        "knowledge":[{
                "scope": "saf_agent",
                "name": "freshener",
                "kind":"resource",
                "attrNames":
                [ "model" ] }],
        "operator": "AND",
        "operands": [{  "operator": ">=",
                        "operand1": {   "scope": "saf_agent",
                                        "   name": "freshener",
                                            "kind":"resource",
                                        "   attributeName": "level"},
                        "operand2": { "value": "10" }},
                                {   "operator": "==",
                                    "operand1":     {
                                                    "scope": "room",
                                                    "name": "activity",
                                                    "kind":"feature",
                                                    "attributeName": "level"},
                                    "operand2": {"value": "true"}}]},
    "description": "actions related to keep the air fresh ",
    "actions":  [{  "name": "spray",
                    "args": []}],
    "output":[{     "scope": "saf_agent",
                    "name": "freshener",
                    "kind": "resource",
                    "effect": "decrease"},
                {   "scope": "room",
                    "name": "freshness",
                    "kind": "feature",
                    "effect": "increase",
                    "attrNames":["level"]   }]
}
```


Fig. 2. JSON Activity Definition

it can be calculated in runtime if the ad hoc reasoning services are available. The reasoning services take the goals to achieve as argument, determine the effect required on the properties of interest to achieve that goal and look for activities producing that effect on the property. In order to achieve one scenario, SO might require cooperation from other SOs. Therefore, it looks for SOs playing the role where the activity is included, asks (send messages to) others to carry out activities, it shares from its knowledge items to achieve the activity.

E. Implementation Approach

Implementation of autonomous SO-based IoT applications is intended to be reduced to the traditional programming of the specific services and the light programming for the agent layer of the SO. Agent layer programming is based on the common functions and is intended to be tailored through the configuration of the needed entities using documents in human-readable data formats e.g. (JSON, YAML. XML, etc.). Optionally, additional user-interfaces can be programmed on top of these documents, considering different user requirements.

Common functions, services, actions, activities and roles are packaged as a middleware which is installed on top of operating system of every SO. Middleware provides APIs for access to common functions from the service implementations in specific programming languages. After middleware is installed, up and running, services or elements from the agent layer can be added, removed or updated in runtime. Once services are implemented and deployed in the SO, the service contracts are intended to be discovered and generated automatically in runtime (part of the common functions).

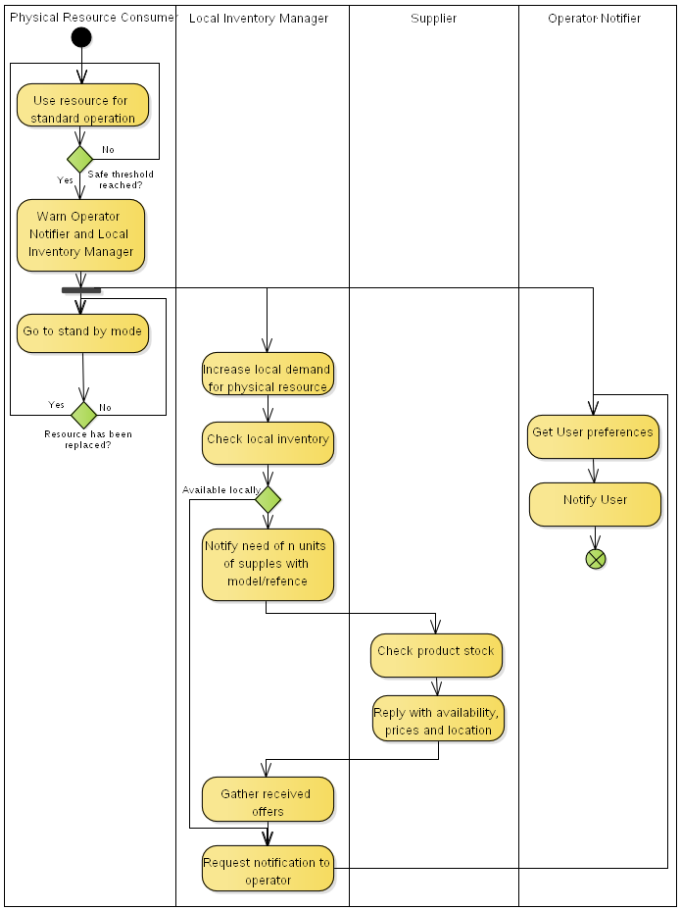

Fig. 3. Scenario for Physical Resource Provisioning

IV. Implementation

We used the framework for the implementation of a prototype. This consists of a middleware —common functions— and a specific application —specific services—. The middleware implementation was developed in Java using the EVE agent platform [1] and CouchDB [2] for the Knowledge base. This implementation is the second iteration over our initial release presented in [2]. Some of the relevant enhancements include: support to goal hierarchies, p2p role discovery, a message protocol, decoupling of role and scenarios and separation of device management functions. Specific services were implemented in Java. JSON Documents were defined for properties of interest, activities, and roles. Properties of interest for the physical resource included usage level, the model/reference or last date replaced. One example of the activities defined for each role is presented in figure 2.

V. Evaluation

We evaluated the framework covering three aspects: A. Feasibility for IoT Development, B. Common features and C.

[1]http://eve.almende.com/
[2]https://couchdb.apache.org/

Basic performance. We approached these aspects respectively in the tests described below:

A. Feasibility Assessment: Case Study

We considered a case with common IoT requirements such as support to heterogeneous capabilities, physical data gathering, adaptation based on the context and cooperation among others.

The scenario is presented in figure 3 and describes the steps for a physical resource provisioning, each swim lane is a role. Everyday objects require consumables to operate. For example, a printer requires printing cartridges/toners, a vacuum cleaner requires filters, an air freshener a fragrance, etc. With SOs in place, sensors detect promptly when the physical resource get consumed and notify the operator, giving also information about where to get the supplies from (local stock or nearby supplier) or even trigger a purchase order.

The hardware architecture for the SOs was based on Raspberry Pis Model B and B+ with WiFi dongles. On top of them we installed Arch Linux 4.x, Open JDK Zero VM 1.7 and then the middleware prototype. The middleware enabled an overlay network between the participant SOs.

There was one SO per role, except for "Operator Notifier" which was deployed in two SOs. These two SOs had a different implementation of the task to notify the user. Here we tested the tolerance to heterogeneous capability/service implementations. We tested the autonomy from others by running the scenario four times and shutting down a different SO every time except the resource consumer. When Local Inventory Manager, was not available, the Resource Consumer could still notify the user about the lack of consumables. When one of the SOs playing the notifier role was not available, the one available made the notification.

The framework abstractions sped up the time to implement each SO. Although every SO was based in the same hardware platform they were heterogeneous in the capabilities (sensing/actuating services) they hosted. Each other SO was able to discover them when joining the network and identifying which roles each SO was able to play. No central directory of platform having all the available roles was required.

The framework applicability of course is not limited to this scenario. Additional roles and scenarios can be defined even using the same capabilities. For example, in the case of SO having movement detection sensor, the sensing service can be used for triggering an alarm, if it checks that it happened in a building during non working hours. Then, security staff can receive a notification in either a screen, a phone call or a voice message depending on where they are and the nearby SOs In other situation, the same sensing service can be used just to activate the lights during a particular time.

B. Qualitative Assessment

The goal was to compare our framework with similar solutions. For this part, we focused in the common characteristics that were packaged as middleware. We used the criteria proposed by Fortino et al. in [8] and the solutions there surveyed. The aspects considered cover middleware requirements such as support to heterogeneity and management of SOs; and some specific features such as system programming, discovery and knowledge management. The reader is advised to follow the reference for further details on the criteria and the compared middlewares. We present in table I the evaluation of our solution based on this criteria.

There are conceptual and technological differences of our proposal. One first difference is the way we address the SO management. The infrastructure functions provide fixed and minimun routines for management the SO. They are required for the SO to run even with the simpest goal. Management services are conceived to allow for extension of these functions when the SO has enough resources to host them. Our solution architecture and the underlying models (e.g. metadata) introduce well known agent notions not exploited in surveyed works such as roles, scenarios, activities or actions. We also articulated them as part of a light programming model for decoupling of the agent's overall behaviour (Document-based) from the concrete action implementation (Service/Object-oriented). The combination of an agent model with a distributed discovery of SOs has not been achieved before in the works under comparison. The distributed discovery is more aligned with autonomy properties inherent to agent paradigm.

Technological differences are addressed to our prototype implementation. Although we also used Java, we varied in the agent platform. This decision required more effort and prevent us for reusing robust facilities. It is justified in that chosen platform give us independence from discovery and message protocols, as well as a intrinsic agent/web service interaction. These characteristics provide an open integration environment for SO development [2].

C. Performance Assessment

The aim of this assessment was to check how key components of the implementation behaved with different load units. The data was taken from a Raspberry Pi B+ using the Hyperic Sigar [3] for monitoring. We initially focused on two key functions the capability/service loader and the SO Discovery. For the service load, we simulated loading up to 30 services. The simulated services where very basic, since the intention was to monitor the pure load process. The results presented in 4 show that the memory usage increase is in the order of KB, with an acute raise with more than 20 services. CPU usage starts in around 21 seconds which is roughly the time taking for SO booting, without triggering the SO discovery. When the number of services is increased the CPU usage rises at proportional pace.

For the SO discovery we simulated up to 20 nodes in an Intel Core i5 laptop. The Raspberry Pi had to discover them using our p2p protocol implementation. Demands of memory for this function are at the order of MB. For the CPU time, the difference between discovering 5 and 20 nodes was in the range of 8 seconds. These measurements give us an

[3]https://support.hyperic.com/display/SIGAR/Home

order of magnitude of the variations in performance when the load is increased. From the works reviewed we did not find similar metrics to compare with. We plan to carry out further measurements in different hardware platforms.

Our current prototype requires platforms with the java VM support. Despite that everyday there are more IoT hardware platforms supporting JVMs [4], it imposes a constraint about the required resources for running the middleware. The framework, however does not hold that restriction and we will explore in the future other middleware implementations (e.g. C and javascript/nodejs) with the idea of reducing the hardware requirements. It is also true, that our architecture is aimed at non-trivial objects, it is clear that in some scenarios with trivial objects it is not worthwhile to endow autonomy. e.g. A pen, at most it would be desirable to store owners information and be able to be tracked if lost, probably also alert when ink is about to gone, but in this case autonomy for the purpose of the object is probably not worthwhile.

TABLE I
QUALITATIVE ASSESSMENT OF THE MIDDLEWARE COMPONENT

Characteristics	Assessment of our solution
Heterogeneity & Application Dev.	Yes, see III-C.
Augmentation	Yes, see III-C
Variation of SO	Yes, see III-C
Management of SO	Yes, by programming.
Evolution of SO	
Systems	
System	Agent-based model &
Programming	JSON & Java
System	Distributed
Architecture	based on EVE
Metada	Properties, roles, agents, activities & services with JSON.
Discovery	Distributed P2P / Web services
Communication	Asyncronous Messages over EVE JSON-RPC
Knowledge Management	Local & remote Document-oriented: CouchDB
SO Architecture	Agent-based & service components &
SO Proactivity	By configuration
SO Cooperation	Through roles & services
SO Application Coupling	Partial

VI. RELATED WORK

In addition to the works in [8] we used to evaluate the middleware part of our framework (section V-B), we present here literature with a different focus but tackling IoT challenges of heterogeneity management and the autonomy of the IoT devices, using the agents paradigm. We distinguish three groups: System engineering approaches, concrete IoT solution architectures and domain-specific applications.

In regards engineering approaches, aspect-oriented agents are jointly used with a Common Variability Language within a Software Product Line process in [9]. Their goal is to provide a

[4]http://www.oracle.com/technetwork/java/embedded/overview/index.html

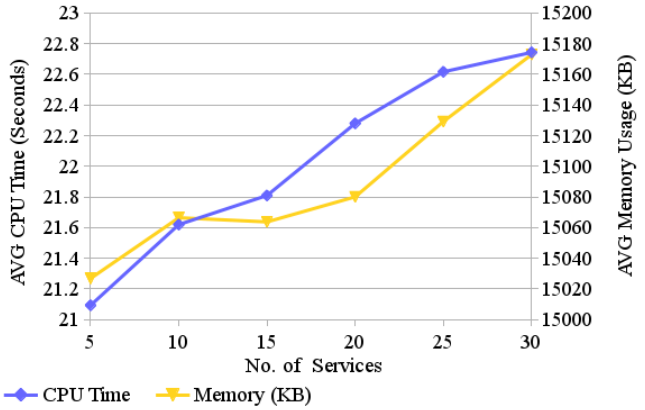

Fig. 4. Performance Metrics Local Service Load

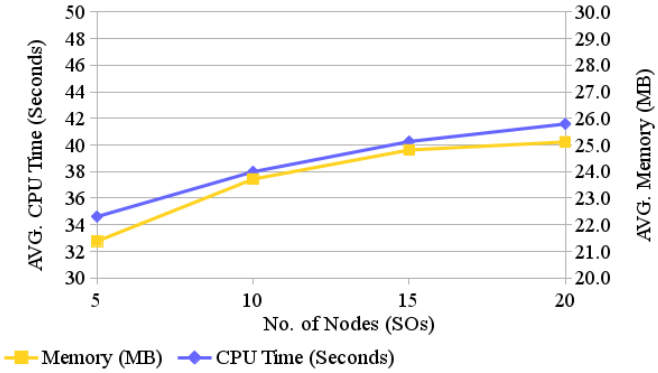

Fig. 5. Performance Metrics SO Discovery

reusable approach for development of IoT application focusing in self-management and heterogeneity. Other more conceptual approaches are found [10] and [11]. In contrast to our work, these ones do not provide insights of how well known agent-world concepts such as roles, activities, actions fit in the IoT application development as well as they are concentrated in the autonomy from the user perspective.

Several IoT solution architecture have been proposed using agents. Authors of [12] propose a 4-strata multiweb architecture for the management of diversified data. Authors of [13] propose a lightweight approach with agent-based and application-specific proxies to trigger services. In [14], an agent architecture is proposed for dealing with heterogeneity at device level. Their strategy rely on separate device-specific functions from agent core with portable service abstractions and using a message bus for communication. Agents are intended to be deployed remotely to the objects and so they inherently depend on the IoT platform. Finally, the work in [15] proposes lightweight agents embedded in constrained devices which are supported by an Agent Platform hosted in a kind of gateway. They provide also a thorough use case within an Intelligent Museum solution [16]. Our proposal is different to these since we aim for independence of platforms or gateways in order to offer a further degree of autonomy for

IoT devices.

Authors of [17] and [18] propose an architecture relying on agents that can be deployed on the cloud or in the object itself. Agents are *avatars*, that pro-actively search for physical actions called capabilities or higher level functionalities. Reasoning is possible with ontologies and SPARQL queries. Our approach is different since we rely not only in the use agents but also roles and scenarios. Besides, we do not require additional frameworks such as OSGI, that could demand more hardware requirements. Finally, our architecture is tailored to specifically to be localised on-object.

Some works for the ambient assisted living (AAL) domain are found for example in [19] and [20]. The first solution is based on the integration of agents embedded in wireless nodes with a service-oriented architecture. They use SOAP and FIPA/ACL for communication requiring translation between each other. The second work includes a three-layer architecture covering environment, reasoning and learning functionalities. This solution is conceptual and is particularly constrained by the domain since some of the agents are specific for it. We try to offer a solution for multiple domains and avoiding the need of a translation between agent and services by building the agent behaviour from the service execution workflow.

VII. Conclusions and Future Work

We introduced a novel framework for building IoT applications exploiting Smart Object's local resources. Our framework is built around an architecture composed by a set of infrastructure management functions that give support to both services and agents layers. Management functions provide the necessary support for enabling basic agent and service operations (e.g. life cycle, discovery, selection, etc.)

The service layer is intended to wrap the cyber physical capabilities of the SO as services. These services include communication, reasoning, actuating, sensing and management. The agent layer is rich in the constructs used including roles, activities, actions and properties of interest. Roles care intended to be installed/uninstalled to endow SO with specific behaviour based on the available capabilities.

We evaluated our proposal with the implementation of an IoT application. The IoT application covered a common scenario in multiple smart environments: the physical resource provisioning. With our framework, the implementation was based in a middleware deployed in every SO which enables the reuse of common functions and reduced the implementation effort of the whole system. The scenario was tested facing unavailability of the participant SO and the system was able to cope with the absence of them, which demonstrated the autonomy of these SO not only in regards performing tasks in behalf human users but also with regards the remote support from platform or gateways.

We continue extending our implementation and making improvements in our design. We plan to enhance reasoning services aiming to a better adaptation in runtime based on utility functions. We are implementing new roles for validation of the framework in a smart building scenario. We will explore the implementation of versions of the middleware in other programming languages.

References

[1] J. M. Mendes, P. Leitão, F. Restivo, and A. W. Colombo, "Service-oriented agents for collaborative industrial automation and production systems," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5696 LNAI, pp. 13–24, 2009.

[2] M. E. Perez Hernandez and S. Reiff-Marganiec, "Autonomous and self controlling smart objects for the future internet," in *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*. IEEE, 2015, pp. 301–308.

[3] H. Hexmoor, C. Castelfranchi, and R. Falcone, *Agent Autonomy*. Springer Science + Business Media, LLC, 2003, vol. 1.

[4] C. Castelfranchi and R. Falcone, "Founding autonomy: The dialectics between (social) environment and agents architecture and powers," in *Agents and Computational Autonomy*. Springer, 2003, pp. 40–54.

[5] M. Wooldridge, *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[6] D. Vernon, *Artificial Cognitive Systems: A Primer*. MIT Press, 2014.

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

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

[9] I. Ayala, M. Amor, L. Fuentes, and J. M. Troya, "A software product line process to develop agents for the iot," *Sensors*, vol. 15, no. 7, pp. 15 640–15 660, 2015.

[10] H. Yu, Z. Shen, and C. Leung, "From internet of things to internet of agents," in *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*. IEEE, 2013, pp. 1054–1057.

[11] A. M. Mzahm, M. S. Ahmad, and A. Y. Tang, "Enhancing the internet of things (iot) via the concept of agent of things (aot)," *Journal of Network and Innovative Computing*, vol. 2, no. 2014, pp. 101–110.

[12] P. Leong and L. Lu, "Multiagent web for the internet of things," in *Information Science and Applications (ICISA), 2014 International Conference on*. IEEE, 2014, pp. 1–4.

[13] T. Leppänen 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

[14] 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.

[15] I. Ayala, M. Amor, and L. Fuentes, "The sol agent platform: Enabling group communication and interoperability of self-configuring agents in the internet of things," *Journal of Ambient Intelligence and Smart Environments*, vol. 7, no. 2, pp. 243–269, 2015.

[16] I. Ayala, M. Amor, M. Pinto, L. Fuentes, and N. Gámez, "imuseuma: An agent-based context-aware intelligent museum system," *Sensors*, vol. 14, no. 11, pp. 21 213–21 246, 2014.

[17] M. Mrissa, L. Medini, and J.-P. Jamont, "Semantic Discovery and Invocation of Functionalities for the Web of Things," in *2014 IEEE 23rd International WETICE Conference*, 2014, pp. 281–286.

[18] 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.

[19] D. I. Tapia, J. A. Fraile, S. Rodríguez, R. S. Alonso, and J. M. Corchado, "Integrating hardware agents into an enhanced multi-agent architecture for ambient intelligence systems," *Information Sciences*, vol. 222, pp. 47–65, 2013.

[20] S. Ferilli, B. De Carolis, and D. Redavid, "An intelligent agent architecture for smart environments," in *Foundations of Intelligent Systems*. Springer, 2015, pp. 324–330.