Mobile Agents-Based Smart Objects for the Internet of Things

Teemu Leppänen, Jukka Riekki, Meirong Liu, Erkki Harjula and Timo Ojala

Abstract We propose mobile agents for enabling interoperability and global intelligence with smart objects in the Internet of Things, with heterogeneous low-power resource-constrained devices where the systems span over disparate networks and protocols. As the Internet of Things systems are in continuous transition, requiring software adaptation and system evolution, an adaptable composition is presented for the mobile agents. The composition complies with the Representational State Transfer principles, which are then utilized in agent creation, migration and control. Moreover, the smart objects' resources, their capabilities, their information and provided services are exposed to the Web for human-machine interactions. We consider the requirements for enabling mobile agents in the Internet of Things from multiple perspectives: the smart object, the mobile agent and the system. We present interfaces for smart object internal architecture to enable mobile agents and to enable their interactions. An application programming interface is suggested with a system reference architecture, which includes components in the information infrastructure. Lastly, an evaluation metrics for the mobile agent composition and for the smart objects' resource utilization are suggested, taking the different types of system resources and their utilization into account, assisting in the system, application, smart object and the mobile agent design.

Keywords Smart object · Mobile agent · Representational State Transfer · Interoperability · Internet of things · Human-machine interaction

T. Leppänen (✉) · J. Riekki · M. Liu · E. Harjula · T. Ojala
Department of Computer Science and Engineering, University of Oulu, Oulu, Finland
e-mail: teemu.leppanen@ee.oulu.fi

G. Fortino and P. Trunfio (eds.), *Internet of Things Based on Smart Objects*,
Internet of Things, DOI: 10.1007/978-3-319-00491-4_2,

1 Introduction

The Internet of Things (IoT) refers to a globally connected, highly dynamic and interactive network of physical and virtual devices [2]. The IoT requires integration and collaboration of disparate technologies in wireless and wired networks, heterogeneous device platforms and application-specific software. These IoT technologies include, but are not limited to, Ethernet, RFID, BlueTooth, Wi-Fi, ZigBee and 6LoWPAN. IoT systems require scalability beyond millions of devices, where centralized solutions could reach their bounds. To achieve global connectivity, standardized protocols and interfaces are necessary to address device heterogeneity and to enable universal resource access. Moreover, IoT systems cannot have fixed deployments or fixed system configurations, as the environment is in continuous transition. Runtime deployment of new services and applications has to be supported in IoT. As the systems evolve over time, it is necessary to consider software adaptation and evolution in order to cope with environment, system configuration and application requirement changes. Specifically, the issues here include interoperability between different standards, protocols, data formats, resource types, heterogeneous hardware and software components, database systems and human operators [2, 9]. The IoT paradigm transforms everyday physical objects into wirelessly networked, intelligent, autonomous and self-aware smart objects and enables smart objects to observe their environment through integrated sensors, store the information and interpret it proactively, cooperate by sharing information and react to changes in the environment [7, 11, 16]. The IoT then becomes a loosely-coupled and decentralized system of smart objects, based on distributed applications and global intelligence.

The agent-based systems feature decentralization and flexibility in the system configuration and allow abstracting heterogeneous subsystems and system resources for cooperation [3, 4, 7, 9, 10, 12, 16]. Agents act autonomously, possess self-properties and allow the direct manipulation of the hosting device and its physical components, such as sensors and actuators. Additionally, in agent-based systems, communication and information processing costs can be reduced by distributing information processing closer to actual data sources. Mobile agents are autonomous programs that transmit their execution state from device to device in networked systems [17], which provides means for software adaptation, system evolution, and tolerance to system or environment failures. Furthermore, mobile agents enable the dynamic reuses of hardware components and asynchronous execution of computational tasks.

This heterogeneous shared environment is not only a technical system, but IoT devices and smart objects also interact with human users. Therefore, smart user interfaces [9] are required for humans to interact with smart objects and access various services and use applications. Abstracting the system through RESTful Web services leads to the vision of Web of Things (WoT) [8], where each smart thing is equipped with a tiny Web server according to its capabilities, which then becomes an integral part of the Web. For embedded networked devices, Web connectivity can be enabled with embedded Web services [18], or by the smart gateways abstracting the most low-power resource-constrained devices, such as 8-bit microcontrollers

[12]. The Web is beneficial for human-machine interactions, as various services for information searching, aggregation and visualization are available.

We propose a method for the integration of mobile agents and smart objects in order to facilitate cooperation and global intelligence, extending our previous work on the mobile agent based integration in IoT systems and wireless sensor networks [12]. This leads to a number of research questions [7, 11]: (1) How are the distributed IoT system architecture designed and eventually deployed? (2) What are the middleware and programming models? (3) How to represent the smart objects capabilities and the distributed intelligence? (4) How to combine a coherent collective application with the distributed application logic? (5) What is the level of human involvement? We seek to answer these questions by enabling platform- and programming language-independent mobile agents in an open standards based framework and information infrastructure. Interactions with humans are facilitated by seamless integration into the Web.

In this chapter, we first present system design considerations for mobile agent-based smart objects in IoT and outline the requirements for the smart objects, agents and systems to enable cooperation. A mobile agent composition is then presented with a RESTful Web service API for smart object internal architecture and reference system architecture. Finally, an evaluation method is presented to assist in system and agent composition design, taking into consideration the different types of IoT system resources and their dynamic utilization.

2 System Design Considerations

IoT system architecture or middleware should facilitate general and non-specific design solutions for applications, because the systems are in continuous transition, where the system configurations and network topologies are ad-hoc, thus cannot be fixed in deployment. The IoT system should provide the augmentation of smart objects with internal and external services, object management capabilities and means for system adaptation and object evolution [6]. Then, the smart objects can control the flow of information, supporting global intelligence autonomously and cooperatively [6, 16]. For IoT, Internet-based information infrastructure is needed to leverage the capabilities of smart objects for provision of services to end-users.

The object-centric systems provide one solution for the heterogeneous fluctuating environments, enabling the abstraction of hardware, software, data and physicality [6]. The distinct features of smart objects include the ability to make complex intelligent decisions in information processing locally and to provide services for end users. In [11], three types of smart objects were presented: activity-, policy- and process-aware, with different levels of information processing and interaction capabilities. The individual different capabilities of the smart objects need to be exposed to the system, be discoverable and queried through well-defined interfaces. In [4], three types of multi-agent platforms for devices with limited resources was described, such as computing power, memory, limited user interface or real-time constrains.

First, the portal platforms do not execute the agents on resource-constrained devices, but only provide user interaction, sensing and actuation capabilities. Secondly, the embedded platforms execute the agents entirely in the device itself and, thirdly, the surrogate platforms execute the agent partially on a resource-constrained device and partially on the other devices. These high-level design considerations lead to some key challenges. The smart objects need to be globally identified and addressable. The resource access interfaces and object capabilities need to be globally discoverable. The dynamic availability of the capabilities in the smart objects is crucial in IoT system deployment. The application development should be decoupled from the smart object development as the smart objects take different roles and interaction models in the system, based on their capabilities and the agents they are hosting. Intelligent decision-making is required for information processing, but also in ad-hoc networking, data routing and providing inter-network relationships.

In Resource-Oriented architectures (ROA) [15], the main abstraction is a resource, referenced through its unique URI. The ROA is based on the principles of the Representational State Transfer (REST), which include separation of concerns with clients and servers, stateless communication, addressability of resources, link-based connectedness, uniform interface for resource access and various representations of the state of the resource. The individual capabilities and resources of smart objects, the state and composition of the agents, system and external services and application-specific tasks should all be considered as REST resources with URI. These resources are exposed to the system and be utilized through a uniform interface. The transportable URI identifies system resources and the URLs support resource hierarchies, linked resources and even private network overlays within the application, a particular task or an agent composition. The transportable URI enables to discover the smart objects, their relationships and contextual situations. This realizes the information infrastructure and system-, application- or task-based network structure repository [16]. Moreover, the IETF Constrained RESTful Environment Working Group [18] has published Internet drafts to enable embedded Web services in the low-power resource-constrained embedded networked devices. These drafts are crucial for IoT solutions, as the existing solutions based on HTTP and Simple Object Access Protocol (SOAP) may be too heavy for the most resource-constrained embedded devices. The CoRE framework enables direct access to the resources in embedded networked devices from the Web and facilitates limited human-machine interactions. These drafts additionally describe a number of infrastructure services such as resource directory facilitator and proxies for protocol translations, which are utilized to implement parts of the information infrastructure in this work.

In our previous work [12], we proposed a REST-based adaptable mobile agent composition, where the principles of REST are utilized in agent creation, composition, migration and control, realizing the requirement for single protocol in the agent transfer, messaging and control. The agent composition itself can be considered as a system resource, promoting re-use, and adaptable to react to unexpected system environment changes. With the RESTful Web services, we are able to utilize standardized uniform interfaces and communication primitives with heterogeneous IoT systems, smart objects and device platforms, based on loosely-coupled and flat distributed

system architecture in WoT. Seamless integration to external Web services follows from the rule of extending the systems over the Web.

3 Requirements for Mobile Agents in IoT

We gather the requirements to enable the system-wide interoperability of smart objects with mobile agents and heterogeneous resource-constrained object platforms as well. General requirements for smart object middleware are previously presented in [5], however we consider the REST principles in the agent creation, migration, control and its composition. Extended from our previous work in [12], the smart objects are capable of sensing and actuation, storing information, local decision-making, interacting with each other and with external entities and finally, of operating in ad-hoc networks [16].

3.1 Requirements for Smart Objects-Enabled Platforms

Sensing and actuation	Smart objects are equipped with physical components, such as sensors and actuators. Both of these components should be identifiable and accessed as resources of the smart objects.
Information gathering	Smart objects can locally process the gathered information, providing them capability to understand their contexts and to make intelligent decisions.
Information dissemination	Smart objects in IoT support many interaction models, such as client-server, publish-subscribe, event-based communication and broadcast messages.
Networking	Smart objects are capable of participating in both intra-network and inter-network communications over disparate networks.
Mobile code	Smart objects in IoT need to support a number of distributed programming models: macroprogramming languages, MapReduce, code migration, task offloading, cyber foraging and virtual machines.
Shared resources	Smart objects maintain their resources and capabilities, which include gathered and refined information, object's capabilities and furthermore hosted agents' resources. The resources are be then cooperatively utilized by other objects and agents in their operations.

3.2 Requirements for Mobile Agents

Shared resources	The agents maintain their own state, exposed by the smart object. As the task state is not tightly-coupled into a physical device, the task state is cacheable and agents provide limited robustness in case of failures.
Agent composition	Agent implementation should be platform- and programming language-independent to address the software and hardware heterogeneity. The agent composition should be adaptive, modified by the hosting devices. The composition can also be exposed as a system resource.
Lightweight composition	Agents must be lightweight in composition, serializable and transferred as a whole or as sequential parts. Agents should be executable in platforms with limited processing power, memory, communication capabilities and battery lifetime. Binary message formats are a necessity for most of the resource-constrained embedded devices.
Dynamic deployment	The agent life-cycle is application-dependent.

3.3 Requirements for IoT Systems

Standardized interfaces	Standard, unified and simple interfaces are required to address device heterogeneity, resource abstraction, and for universal access. To simplify the implementation of mobile agents, agent transfer, messaging and control protocols should be integrated into a single protocol, based on basic communication primitives.
Abstracted objects	Smart objects and their resources should be utilized through basic and standardized communication primitives with unified interfaces, where the primitives should be interface and protocol independent. The agents can also provide primitive or atomic operations in the system.
Abstracted resources	In addition to the resources in smart objects, the system resources include internal and external services, which may register themselves into the system with various roles, such as data producer, aggregator or interaction enablers. These resources may also introduce their own restrictions and priorities.
Dynamic deployment	The systems are in continuous transition, therefore the runtime injections of objects and agents into the system are common, where the i.e life-cycle is application-dependent.

Table 1 Mobile agent composition for the smart objects

Segment	Elements	
Metadata	Name	Agent i.e. resource name or URI
	Migration	Policy identifier
	Authorization	Access rights
	Timestamp	Time of last state update
Code	Type identifier	Task code
	Reference	URL
Resource	Local	URL list
	Remote	URL list
	Static	URL list
	Reference	URL list
State	State variable list	
	Local variable list	
Historical data	Variable list	
	URL list	

Dynamic binding	The objects are simultaneously acting as servers for their local resources and as clients for the resources in other objects. The agents should allow dynamic binding to resources and dynamic mapping of the task into any system configuration. An agent composition should, in general, be exposed to the system by the devices and be adaptable. Runtime lookups and loose coupling to the resources are facilitated by stateless communication.
Scalable configuration	Scalability beyond current networked systems is required. Thus distributed architectures with loosely-coupled services become necessity. Gateways and proxies are introduced to abstract heterogeneous subsystems, spanning over networks, protocols and communication interfaces.

4 Composition for the Mobile Agents

We extend the agent composition presented in [12], to fulfill the smart objects and system requirements. The composition, illustrated in Table 1, consists of three segments: code, resource and state. In addition the composition includes metadata, such as a unique name or URI, to register the agent into the system and to enable resource lookups. The metadata also contains a description of globally known migration policy to control the agent migration. The metadata may also include the last time the agent state was updated and authorization information for the agent and the required resources. With proper authorization, we allow smart objects, smart gateways and

proxies to modify the composition to adapt and evolve to system environment, application or task configuration changes and to dynamic resource availability. Then, the agent composition itself becomes a system resource. Historical data can be included in the composition for agent tracking purposes.

4.1 Code Segment

The agent task code is stored into this segment. The code can be presented in any programming language: high-level macroprogramming language, scripting language, precompiled binaries, bytecode or even as machine language instructions. The segment allows multiple code segments for multiple heterogeneous platforms, which requires an identifier of the code type. Additionally, to minimize the composition size, the segment can contain a reference to the code in the system repository for on-demand code retrieval.

4.2 Resource Segment

The resource segment lists the local, remote and static resources for the task execution. The local resources refer to the resources exposed by the hosting smart object, whereas remote and static resources are external to the object. The remote resources are accessed each time the agent migrates or the task execution iterates. The static resources are remote and constant for the lifetime of the agent, hence the representation is requested only once and moved into the state segment as a variable. How the object binds to the remote resources are determined by the references and the resource access interfaces. If the resource segment is a reference, resources are requested from a application-specific or global system service. This allows sharing the segment becomes a system resource and enabling runtime modifications as well. The resource segment therefore presents dynamic and partial view of the system resources utilized by the agent, as an overlay.

4.3 State Segment

The state segment contains the current state of the agent, i.e. the intermediate or final results of the task. The state is then returned as the agent resource representation. Other local data, such as a program counter, local variables and retrieved static resources are stored in this segment as well [12].

4.4 Historical Data

This segment is optional. It contains the previous states of the agent, its local variables and previously visited locations, for tracking the agent and its behavior. Also, to minimize the composition size, this segment may contain URLs to a repository hosting this information.

4.5 Agent Mobility

The local resource segment dictates where the agent migrates in the system, with the particular migration policy given in the metadata segment. We can utilize any migration police, for example: (1) the agent visits the objects listed in the local resource segment each only once, (2) the agent considers the local resource segment as a ring buffer circulating though the devices, (3) the agent message is broadcasted to all the objects at once, (4) the local resource segment lists gateways or proxies, which distribute the agent to any number of abstracted smart objects. Actually, the migration policy can also be considered as a system resource. Whenever the local resource segment is a reference, the smart objects rely on the information infrastructure, such as the network structure repository [16] for migration instructions.

This agent migration procedure requires, at minimum, that the agent is cloned in the host device and sent to the new host, where the state is first updated by executing the computational task. Then the host registers the agent into the system name server or directory facilitator. Here the updated agent state is exposed into the system and other objects can access the new state in the new host. After successful registration, an acknowledgement is sent to the previous host, which can then delete the agent from its memory and free the utilized resources.

4.6 Implementation Considerations

Considering different utilization of the adaptable agent composition, we assume the following. The state segment cannot be omitted, as it is the agents resource representation. If the code segment is omitted, then the agent works as a data aggregator, migrating through the listed smart objects. Moreover, this enables event-based communication [1] as the agent composition can be considered an event with a state. If the local resource segment is omitted, the agent does not migrate autonomously, which implements task offloading. The remote and static segments are optional based on the required resources. All the activity-, policy- and process-aware smart object functionality [11] become now possible. This agent composition inherently supports the multi-agent platform types in [4], as the requested resource representations dictate which parts of the agent task code are executed and where.

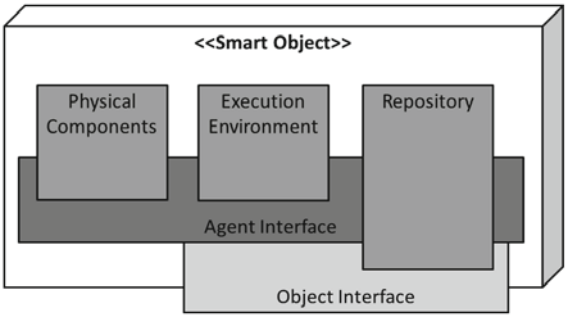

Fig. 1 Smart object internal architecture to facilitate mobile agents

With the flexible structure of the resource segment, objects and intermediates can modify the composition [12]. The client-server paradigm is the default; agents send state and resource requests to other agents. Publish-subscribe paradigm can be achieved through mobile agents as events. MapReduce can be implemented by cloning the agent, or by broadcasting the resource requests to the system devices, where partitioning the task into smaller computational units can be considered before sending the task to the devices. Macroprogramming languages can be supported through agents as high-level code abstractions or as code primitives, which can be introduced to the system as on-demand task code or as global system resources. These primitives representations can be considered remote methods or in-line code for the task.

Mapping the agent composition to different protocol messages needs to be considered. With the HTTP, we can assume the composition for example as HTML or XML document, EXI XML representation or as JSON object. However, these human-readable formats may introduce too much overhead in communication with low-power resource-constrained embedded devices. Therefore, we presented the agent composition mapping into a significantly smaller, in size, binary Constrained Application Protocol (CoAP) message structure in [12].

5 Smart Object Reference Architecture

For the smart objects, we identify three software components, which are necessity to enable mobile agents: the execution environment to run the actual agent task, a repository to store the resources in these objects and the physical components, such as sensors and actuators. The repository contains both data and the knowledge base, typically in a relational database. We also define two interfaces: the agent interface to enable the handling of mobile agents and the object interface for communication with other smart objects and the system. See Fig. 1 for the proposed smart object internal architecture.

The execution environment (EE), it is a hardware and operating system dependent. The EE is capable of querying information from the repository and from the physical components to compile a runnable code for execution, after retrieving the required resource descriptions. Additionally, EE provides methods for actuating and controlling the physical components. The EE must feature a method to immediately stop the agent code execution, called by the EE or by the agent task code, which enables the agent to control its own execution.

The implementation of EEs for the Android operating system in Java and for Atmel microcontrollers in the C programming language was presented in our previous work [12]. The Android EE allows scripting languages Python and JavaScript as agent task code, where a language-specific engine is invoked to execute the script code. A HTTP server component is used for communication and a SQLite database for the repository. The EE in the microcontrollers uses IntelHEX precompiled code for agent task code. The task code is flashed into the memory in the device, as code cannot be run from the RAM in the ATmega architecture. However, the architecture allows flashing program memory sections without a reset, a crucial feature here. The local and remote resource representations are stored into a shared memory block in RAM, from where the executable code accesses them as 16-bit variables through common pointers. These pointers and the API methods are defined in a common C header file. In the program memory, a number of slots are reserved to store the agent code and it is accessible until overwritten. The communication API was implemented in C for the CoAP protocol.

5.1 The Agent Interface

The interface is internal within the smart object, providing the methods for handling the agent messages and agent composition, the execution of the agent tasks and local resource queries from the repository. Methods are provided to control the integrated physical components and to stop the agent execution immediately.

Marshal/Unmarshal	Handles the serialization and deserialization of the agent composition into an internal data structure in the device memory and back into the transferable agent composition, then utilized by the object interface. The data structure stores the binding of the remote and local resources.
Map/Unmap	Maps the internal data structure and local resource representations into the executable code object. After the task execution, the internal data structure is updated with the new resource representations.
Execute	Runs the executable task code object.
Getter	Retrieves the intermediate state of the agents task from the internal data structure, to respond to external state queries.

Poster	Used for disseminating events from the task code and for actuating the physical components from the code.
Stop	Called by the EE or from the agent task code to stop the task execution and to immediately transfer the agent.

5.2 The Object Interface

This interface provides functionality for inter-object communication, including resource access and registration into the system. The methods allow query parameters for retrieving information with location information, different granularity and historical data from the repositories in the objects. This allows discovering nearby smart objects and resources dynamically.

Post	Transmit the agent between smart objects, according to the resource segment addresses.
Get	Enables two-way communication by responding to the external queries of local resources, including the agent state. Secondly, it is used to request remote and static resource representations from other objects. It may be needed to first perform resource lookup into the name server or directory facilitator.
Delete	Deletes the resource, including the agent, from the hosting object or from the system.
Register/Unregister	Registers the object, its resources and capabilities into the system. Unregister is used to remove the resource description from the system. Whenever an object is hosting an agent, its identifier with the object network address is registered into the system. The address of the name server or directory facilitator should be globally known by all system components.

6 System Reference Architecture

The system reference architecture is generally based on the framework described by IETF CoRE Working Group in [18]. The benefit of the CoRE framework is that it allows embedded Web services, i.e. Web connectivity, for the most resource-constrained embedded networked devices.

In the Fig. 2, the resource directory (RD) acts as a name server and stores the resource descriptions as a part of the information infrastructure. Smart objects can lookup exposed resources in the system from the RD by the presented API. As described in [13], the RD can be a part of an P2P overlay over the IoT system. Queries can be based on URI or resource name, output type, semantic interpretation, and both virtual and physical location. Secondly, in the system architecture we

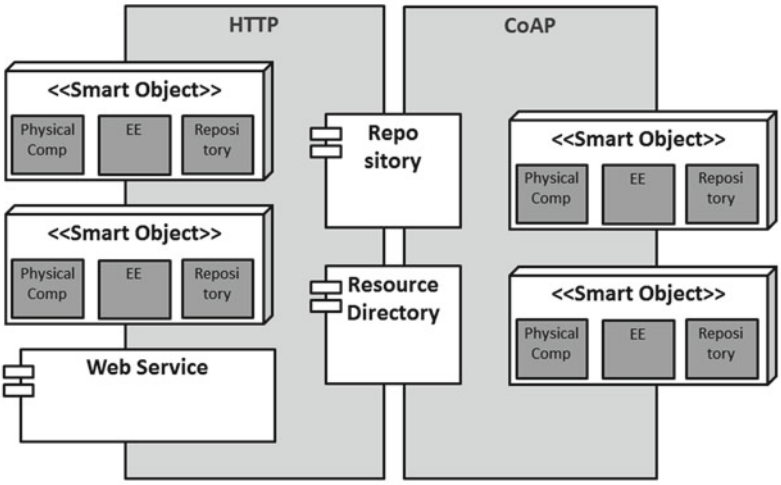

Fig. 2 Smart object-based IoT system reference architecture

utilize generic Web service in several different roles. Web service can abstract heterogeneous hardware and software technologies, heterogeneous systems and smart objects resources, coordinate application or task execution to provide application-specific intelligence, expose external services into the system, work as a gateway or proxy over disparate networks enabling interoperability and finally facilitate human-machine interactions over the Internet. Lastly, we utilize a repository component to store and expose global and application-specific resources into the system, such as agent task codes and agent compositions, accessed through the RD lookups. Therefore, the repository enables smart objects to adopt any agent-based role and facilitates the re-use of system resources as a part of the information infrastructure. The repository can also be part of, for example, a P2P overlay. These system components could provide communication interfaces for different protocols, namely HTTP and CoAP, to allow access over disparate networks.

7 Application Programming Interface

We extend the API presented in [12] with smart objects-based features for a reference RESTful API for mobile agent-based application development, complying with the ROA. With this method, the role of the objects depends solely on their local resources and on the agents they host. The API features mobile agent creation and control, agent migration, communication between devices and agents, and also local and remote access to the resources in the smart objects. It facilitates interobject and interagent

Table 2 Application programming interface

GET/object/resource	
GET/web_service/resource	This method is used for requesting resource representations either from a smart object or from a Web service
POST/object/agent_name	
POST/web_service/resource/agent_name	With the first method, the agent migrates between objects. In the second case, an agent is injected into the system for the given resource, exposed by the Web service. If an agent composition is not provided in the message body, a resource lookup is done to locate the smart objects hosting the particular resources. The resource segment is compiled automatically, if resources based on the agent name are available in the system
DELETE/object/resource	
DELETE/web_service/resource	This method will delete the given resource from the addressed smart object and request the deletion of the object from the RD. In the second case, a lookup to locate the hosting object for the resource is first performed, if needed to locate the hosting agent or service
GET/repository/agent_name	
POST/repository/agent_name	These methods will retrieve from or store the particular agent task code into the repository, with the given platform identifier as a query parameter. When adding new code, included in the message body, the repository will register the task as a system resource
GET/resource_directory/resource	
POST/resource_directory/resource	The lookups to the resource directory follow the same methods: GET returns the description of a resource and POST injects new resource description from the message body to the directory. The resource description follows the format outlined in [13]

Table 3 Additional query parameters

object={list of smart object URIs}	Allows directly manipulate the resources, including agents, in these particular smart objects. The requests are only sent to these particular devices. This can also override the resource segment in the agent composition
location={URI}	This identifier can be both physical location or logical address of the resource
time={start_time, end_time}	Access historical information in the agent, smart object or system resource. Additionally, when this parameter is sent to the RD with lookup request, it allows tracing the particular object or resource
rate={integer}	Set the granularity in the information requests, if available. This is an application-specific parameter

communications with basic HTTP and CoAP methods, additionally with inherent content negotiation and authorization methods. This realizes the requirement for a single protocol with a uniform interface. See Table 2 for the API description. We introduce additional parameters, in Table 3, for querying historical data with different information granularity and tracing the agent or objects location or status.

8 Evaluation of Agent-Based IoT Systems

To evaluate mobile agent-based smart objects in IoT, with the possibility of dynamic resource utilization, a set of specific measures are needed. We propose here metrics such as resource utilization costs in terms of access and communication latencies, to describe system-, application-, device-, object- and agent-based characteristics. In our earlier work [12], we proposed communication, remote resource access and agent migration latencies and computational overhead in agent task execution as the measures. Furthermore, we can compare different configurations of the above.

1. With the resource access latencies, we measure the latencies either directly between heterogeneous devices or through the abstracting Web services. Additionally, we should measure the access latency from the Internet to system platforms, considering resource access over disparate networks. This measure could include the request processing time in the hosting device. In the agent composition, we assume that this latency is dominated by the number of remote resource queries and should linearly increase as in [14]. Queries to the RD are considered the same as standard resource accesses.
2. With the computational overhead, we measure the computational latency in the particular EE in executing agent task code. Platform-specific latencies include time for system atomic service invocation, marshaling and mapping the composition into the device memory, running the code and composing the agent message again.
3. Agent migration latency includes the overhead of agent registration into the RD by the hosting device, sending the agent as a message to the next device and waiting for acknowledgement, after which the agent can be deleted from the memory. This does not include the computational overhead or additional resource access latencies. This latency would increase with introducing security measures, such as guaranteed reliable message transmission, and with large-size agent composition.

However, we found out that in the real-world environment [12, 14], conclusive evaluation would be difficult to conduct with heterogeneous smart object platforms, as the system configuration, device and object deployment, agent composition, required resources and their locations are largely application-specific. Additionally, the varying communication latencies, changing network conditions, device failures and resource availability are difficult to consider. The evaluation additionally introduces overhead, which would reduce query response times in the most resource-constrained platforms, such as wireless sensor networks nodes. Therefore, we should

utilize of indirect evaluation, for example in smart object or agent communication through application-specific Web services or by tracing the resource accesses or agent migration indirectly from the RD. Echo request message's, such as the ping, round-trip latencies could be useful for measuring communication latencies as a baseline.

To assist in the evaluation of mobile agent-based IoT system, application and agent composition design, we proposed simplified equations in our previous work [12]. We extend the equations to include smart object-specific features, such as the repository component in each device. The equations identify the relevant factors in each case, and with modifications allow calculating application-specific costs with different system and resource configurations. In Eq. 1, we estimate the maximum latencies C in particular execution environment k, including the resource access, executing the agent task and the following agent migration latency. Here r is the number of remote resources, T_r is the response time for remote and static resource requests, T_k is the computational overhead in the device and T_m is the migration latency from sending the agent message to receiving an acknowledgment message. T_r is added once for agent registration to the system. The local resource query latencies T_l can be considered negligible, however with large information chunk retrieval or with large number of local resource accesses l, this can be considerable. Based on the observations in [12, 14], the remote resource queries and migration latencies dominate these costs.

$$C_k = (r + 1)T_r + lT_l + T_k + T_m \tag{1}$$

The Eq. 2 gives the total agent migration costs C_{Total}, for a particular mobile agent-based service as the agents migrate over disparate networks.The additional latency for static resource queries is included, where s is the number of static resources. The number of disparate networks is d, and here it is assumed that the agent migrates only once to each network. The equation can be modified to cover different scenarios. The agent migration time between networks is given as $T_{m,d}$. We include the latency of possible message translations in the gateways as T_p. The latencies in each execution environment are given in $C_{n,t}$, from Eq. 1, where the number of devices running each execution environment is n.

$$C_{Total} = sT_r + \sum_{d-1}(T_p + T_{m,d}) + \sum_{d}\sum_{n-1}(C_{n,t}) \tag{2}$$

The cost, as latencies, is dominated by the number of platforms in each network and the previously noted remote resource accesses [12, 14]. Therefore, the remote resources in the system design and agent composition should be considered as static or local resources as much as possible [12]. However, this is an application-specific tradeoff between the agent migration costs and resource access latencies, as the composition allow the different utilization of the resources. In IoT, we can envision systems over a number of disparate networks, all with their own characteristics and technologies, therefore, the migration cost and resource access latencies over

disparate networks are significant factors in the system and application design and in the deployment phase.

9 Related Work

To start with, an extensive evaluation of middleware for smart objects and smart environments in IoT, can be found in [5]. Here we consider the previous work related to agent-oriented smart object-based systems in IoT.

In [9], the authors envision agent-based IoT system architecture, where the resources are represented by agents. Agents handle monitoring the state of the resource, historical data storage and the interactions with other components and humans. Monitoring and coordination of the resources is done through specific roles played by the agents. Communication is based on the role of the agent and not to its name or identifier. For resource discovery, the semantic queries are addressed to the directory to locate the resource identifier. The tasks are written in a rule-based language, where the agents provide the system configuration for the tasks and react to configuration changes.

In [7], the authors present agent-based architecture for smart objects, where the IoT system heterogeneity is abstracted with layers. The system architecture provides communication middleware to abstract underlying details, a component for managing communication with external systems, a resource discovery module, adapters to abstract sensors and actuators as system resources, and lastly components for managing contexts, knowledge base and reasoning. The implementation is Java-based. The master-slave model is used with smart objects, where a coordinator manages the set of software entities, running on other smart objects. The coordinator controls the hosting smart object though internal communication protocol and is the sole component to communicate with other smart objects or system devices, through external communication protocol. An internal software framework provides API for atomic services and runs the EE as an additional internal software framework.

In [16], an event-driven smart object framework for IoT is presented. The smart objects communicate by forming ad-hoc clusters, based on the common context of objects, with electing representative to each cluster. The objects communicate within the cluster, where only the representative communicates with the infrastructure. In communication, XML documents are disseminated over SOAP and HTTP through a Web service in a gateway node. Two types of events have been defined: network structure changes to manage the clusters and events to disseminate sensor data. For addressing and routing, the authors have developed their own mechanism, considering merge and split operations with unique and reusable addresses. The role of cluster representative rotates according to available resources in objects, balancing the communication load.

In [1], an agent-oriented and event-based framework for cooperative smart objects, based on the architecture in [7], is presented. Smart objects' behaviour, in the form of tasks, is separated from the event-based communication management. The tasks are

separated as system tasks, providing basic services for the smart objects and as user-defined tasks, in application-level, to define the smart objects' behaviour as plans. Events are categorized as information, request, log and error. Event types include system internal events, external events and another smart object as an event source. The communication model is publish-subscribe, where each smart object publishes its topics and services for others to utilize.

In [3], interoperable agents in IoT are presented, abstracting heterogeneous devices and communicating over different access technologies simultaneously. The agents register their identifier, type and transport protocols to the directory facilitator. The facilitator enables the registration and discovery of agents, group memberships of agents, system services and a messaging service for messages between agents. The groups, as IoT applications, enable multicast messaging with the members.

Considering agent platforms for smart objects, the authors in [10] present a multi-agent platform for embedded systems, based on the Java virtual machine. The device platforms include static system agents providing interfaces to the system services and, on the other hand, dynamic service agents running the smart home applications. In [1], mobile agent framework for SunSPOT platforms is implemented in Java. The agents are modeled as multi-plane event-based state machines, where the state transitions come in response to events. New events can then be emitted asynchronously. A distinct feature is the timing of the agent operations by system components, which additionally offer services for communication and agent control.

In comparison, we presented a novel, language- and platform independent composition for mobile agents-based smart objects. This method is based on open standards for communication over disparate networks and for collaboration support without specific interaction models or middleware. The information infrastructure is realized with the IETF CoRE framework [18], additionally enabling resource-constrained device platforms for smart objects becoming integral part of the Web. The system architecture is flat and is not restricted to specific interaction, communication or programming models. Centralized system configuration or agent coordination is not facilitated and we do not apply any specific system configuration or task plan with the smart objects. Instead, we expose the modifiable agent composition into the system as a common resource. This method facilitates dynamic interlinked many-to-many communications, including external systems, despite the roles of the agent or smart objects. The REST design principles and unified interfaces are utilized for agent creation, migration, messaging, control and exposing system resources to the Web. Lastly, although Java software components are modular, portable and provide object-oriented features for programming, Java virtual machine-based solutions may be too heavy for the most resource-constrained embedded devices.

10 Discussion

In this work, we proposed a method for integration of autonomous smart objects with mobile agents, with open standards for communication and cooperation sup-

port without a specific middleware solution. We presented language- and platform-independent mobile agent composition, which enables global intelligence and different interaction models for the smart objects and mobile agents. The roles of the smart objects are decided by the agent composition, which promotes the dynamic re-use of the system resources with different simultaneous applications. Mobile code is inherently supported as the mobile agents can be considered as application-level tasks, high-level programming abstractions and code primitives. The expected benefits include: mobile agents enable global intelligence, mobile agents facilitate adaptable system configurations and dynamic service composition, distribute computational load in applications, exploit locality in communication, and finally provide re-usability and robustness for the smart objects.

The REST principles are utilized in agent creation, migration, control and, in larger-scale in smart object communication, system resource access and exposing the resources to the internet, including the agent composition itself. This realizes the single protocol for uniform interface in ROA-based architecture. Moreover, the system resources, services and smart objects are exposed to the Web for human-machine interactions, which provides integration into the WoT.

The presented evaluation method, albeit generic and simplified, can assist in application-specific IoT system design, in smart object- or mobile agent-based dynamic service composition and in system service response latency estimations. Additional system and network-specific parameters should be introduced to real-world evaluations. The inevitable security and privacy issues in agent-based approaches were omitted in this work, but to some extent the security mechanisms of communication protocols are available with RESTful Web services.

Acknowledgments This research was conducted with the MAMMotH Project, funded by the Finnish Funding Agency for Technology and Innovation (Tekes), at the Department of Computer Science and Engineering, University of Oulu, Finland.

References

1. Aiello, F., Fortino, G., Gravina, R., Guerrieri, A.: A java-based agent platform for programming wireless sensor networks. Comput. J. **54**(3), 439–454 (2011)
2. Atzori, L., Iera, A., Morabito, G.: The internet of things: a survey. Comput. Netw. **54**(15), 2787–2805 (2010)
3. Ayala, I., Amor, M., Fuentes, L.: An agent platform for self-configuring agents in the internet of things. In: Infrastructures and Tools for Multiagent Systems, p. 65 (2012)
4. Carabelea, C., Boissier, O.: Multi-agent platforms on smart devices: dream or reality. In: Proceedings of the Smart Objects Conference, pp. 126–129. Grenoble, France, (2003)
5. Fortino, G., Antonio, G., Russo, W., Savaglio, C.: Middlewares for smart objects and smart environments: overview and comparison. In: Fortino, G., Trunfio, P. (eds.) Internet of Things based on Smart Objects: Technology, Middleware and Applications, Internet of Things. Springer, Berlin (2014)
6. Fortino, G., Guerrieri, A., Lacopo, M., Lucia, M., Russo, W.: An agent-based middleware for cooperating smart objects. In: Corchado, J., Bajo, J., Kozlak, J., Pawlewski, P., Molina, J., Julian, V., Silveira Ricardo, A., Unland, R., Giroux, S. (eds.) Highlights on Practical Applications of

Agents and Multi-Agent Systems, Communications in Computer and Information Science, vol. 365, pp. 387–398. Springer, Berlin (2013)
7. Fortino, G., Guerrieri, A., Russo, W.: Agent-oriented smart objects development. In: 16th IEEE International Conference on Computer Supported Cooperative Work in Design, pp. 907–912 (2012)
8. Guinard, D., Trifa, V., Wilde, E.: A resource oriented architecture for the web of things. In: Internet of Things 2010 Conference, pp. 1–8 (2010)
9. Katasonov, A., Kaykova, O., Khriyenko, O., Nikitin, S., Terziyan, V.Y.: Smart semantic middleware for the internet of things. In: 5th International Conference on Informatics in Control, Automation and Robotics, Intelligent Control, Systems and Optimization, pp. 169–178. Funchal, Portugal (2008)
10. Kazanavicius, E., Kazanavicius, V., Ostaseviciute, L.: Agent-based framework for embedded systems development in smart environments. In: Proceedings of International Conference on Information Technologies. Kaunas, Lithuania (2009)
11. Kortuem, G., Kawsar, F., Fitton, D., Sundramoorthy, V.: Smart objects as building blocks for the internet of things. Internet Comput. **14**(1), 44–51 (2010)
12. Leppänen, T., Liu, M., Harjula, E., Ramalingam, A., Ylioja, J., Närhi, P., Riekki, J., Ojala, T.: Mobile agents for integration of internet of things and wireless sensor networks. In: IEEE International Conference on Systems, Man, and Cybernetics, pp. 14–21 (2013)
13. Liu, M., Leppänen, T., Harjula, E., Zhonghong, O., Ramalingam, A., Ylianttila, M., Ojala, T.: Distributed resource directory architecture in machine-to-machine communications. In: IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications, pp. 319–324 (2013)
14. Malek, S., Medvidovic, N., Mikic-Rakic, M.: An extensible framework for improving a distributed software system's deployment architecture. IEEE T Softw. Eng. **38**(1), 73–100 (2012)
15. Richardson, L., Ruby, S.: RESTful web services. O'Reilly (2008)
16. Sanchez Lopez, T., Ranasinghe, D., Harrison, M., McFarlane, D.: Adding sense to the internet of things. Pers Ubiquit Comput. **16**(3), 291–308 (2012)
17. Satoh, I.: Mobile agents. In: Nakashima, H., Aghajan, H., Augusto, J.C. (eds.) Handbook of Ambient Intelligence and Smart Environments, pp. 771–791. Springer, Berlin (2010)
18. Shelby, Z.: Embedded web services. IEEE Wirel. Commun. Mag. **17**(6), 52–57 (2010)