MINDiT: A Framework for Media Independent Access to Things

Daniel Corujo[a,*], Marcelo Lebre[a], Diogo Gomes[a], Rui Aguiar[a]

[a] *Universidade de Aveiro, Instituto de Telecomunicações, Campus Universitário de Santiago, P-3810-193 Aveiro, Portugal*

Abstract

Deploying Smart Environments often implies a plethora of co-existing devices and services, each with their own set of features, requirements and interfacing characteristics. These intricate scenarios are further exacerbated when such devices are coupled with networking capabilities, globalizing their interaction opportunities to create the so-called Internet of Things. In such interconnected heterogeneous environments, the joint operation of entities requires a flexible framework that enables and simplifies interfacing between elements. In this paper, we propose MINDiT, a framework that provides a common abstract interface towards the communication support with different entities. It incorporates cross-layer mechanisms inspired on the existing IEEE 802.21 technology, suitably modified to facilitate and optimize deployment in scenarios featuring both high-level, and low-powered network-restricted entities. MINDiT was validated through a prototype built over an open-source IEEE 802.21 implementation. We further compared its signaling impact against other solutions, and evaluated its performance over a smart environment featuring a multimedia scenario with multiple devices and services.

Keywords: Internet of Things (IoT), Media Independence (MI), Heterogeneity, Smart Environments, IEEE802.21

*Corresponding author

Email addresses: dcorujo@av.it.pt (Daniel Corujo), marcelolebre@av.it.pt (Marcelo Lebre), dgomes@av.it.pt (Diogo Gomes), ruilaa@av.it.pt (Rui Aguiar)

1. Introduction

Evolutions on networking access technologies and protocols have created a plethora of new connectivity scenarios featuring an ever-increasing amount of devices and networking entities. In the Internet of Things (IoT) vision, the traditional view of the Internet as a network for remote resource sharing has evolved into an heterogeneous environment, where a multiplicity of devices and services co-exist and use the Internet fabric to share their functionalities. This vision creates immense possibilities, where real-world devices (both sensors and actuators) and services deployed anywhere in the Internet can be accessed and manipulated to interact with one another, jointly conceiving and realizing Smart Environments able to impact and improve every aspect of our everyday life.

This heterogeneity (considering the diversity of connectivity technologies at pico, micro and macro level, as well as different services and interfacing possibilities that are reachable on-line) raises complex interoperation issues: different devices from different vendors, providing distinct features accessed by diverse services using disparate protocols, standards and interfaces for exercising various kinds of behaviors, all pose a difficult canvas for the definition and deployment of common-based scenarios.

Realizing this IoT vision requires a common interface that enables and facilitates networking and control procedures. However, such interface must be able to operate over specific technological functionalities and requirements from high-level services (e.g., mechanisms supporting decision modules, management entities, service provisioning), as well as low cost constraints (e.g., low-powered sensors, fast versus slow links, mobile wireless environments). It also has to take into consideration the way the Internet is evolving, with users accessing services while on the move, and reports [1] heralding the dominance of ubiquitous multimedia services and content.

To address these challenges, in this paper we propose and define MINDiT, a framework able to provide a common abstract interface towards the connectivity of agents wishing to interface with different entities in an IoT. It provides a comprehensive and flexible approach towards the definition and retrieval of interfaces for interacting with both services and devices. To efficiently cope with the different specific requirements posed by both higher and lower layers of the network stack, we couple MINDiT with existing management concepts assisting the network: we evolved MINDiT over the Media Independent Handover (MIH) services concept introduced by the IEEE

802.21 [2] standard. Taking advantage of the MIH versatility, MINDiT enables common interface mechanisms to operate in demanding mobile-aware scenarios, as well as providing the flexibility for interface dissemination and representation. The performance of our framework is evaluated through a prototype built over an open-source implementation of the IEEE 802.21 standard, interfacing different devices and services.

The remainder of the paper is organized as follows. Section 2 identifies related work on technologies for smart and ubiquitous services, followed by Section 3 which describes our framework architecture. Section 4 presents a functional and resource utilization comparison between MINDiT and major Web Service-based frameworks. Next, Section 5 presents a multimedia smart environment scenario which serves as a deployment example for our framework, whose implementation performance results are presented in Section 6. Finally, section 7 concludes the paper.

2. Related Work

The plethora of different smart devices and services that compose IoT scenarios [3] [4] [5], have since long raised the associated challenge of heterogeneous interfacing [6]. A proposed solution is the deployment of middleware that provides an intermediate software architecture and simplifies the access to different kinds of sensor technologies [7]. Other approaches consider enhancing this middleware with dynamic programmable features [8] and sensor clustering into virtual machines viewed as applications [9]. These solutions, although enlarging the scope of interfaced sensor information sources, adopt a vertical deployment approach that limit not only the adoption of new kinds of devices, but also its extension to multiple target applications (e.g., Urban Computing).

On the other hand, the differences in accessing low-powered devices and application frameworks often results in isolated and dedicated systems, hindering the design of new services or the support for new devices. To close this gap, increased research and standardization efforts were made towards the integration of small devices through service oriented architectures (SOA) using Web technologies [10].

Works developed under the umbrella of projects such as Service Oriented Device and Delivery Architecture (SODA) [11], Service-Oriented Cross-layer infRAstructure for Distributed smart Embedded devices (SOCRADES) [12] and Integrating the Physical with the Digital World of the Network of the

Future (SENSEI) [13], have explored frameworks deploying Web Services at the device level, through the usage of the Devices Profile for Web Services (DPWS[1]) protocol stack. This OASIS (Organization for the Advancement of Structured Information Standards) standard provides services for discovery, event processing, description and addressing. However, despite targeting resource-constrained devices, the XML nature of the used SOAP messages makes them large in size and requires devices to support HTTP. To counter the former issue, solutions such as the Efficient XML Interchange (EXI[2]) specifies XML encoding procedures that reduce this overhead. Nevertheless, such operations require computation resources that might not be easily available in low-powered devices. Lastly, DPWS still requires its integration with other services, such as Web Service Definition Language (WSDL[3]), to describe the services running in devices.

A different web service device information access (and incrementally supported in several works related to SENSEI), the REpresentational State Transfer (REST) approach provides lightweight means for accessing resources as web services, using standard HTTP methods with Uniform Resource Identifiers (URIs). Frameworks employing REST [14] [15] allow direct access from the Internet to the sensor network and minimize the overhead introduced by the transport layer, when compared to XML and WSDL. Particularly, [16] considers that the lightweight nature of this approach is feasible for low-powered nodes in terms of sensor data acquisition time and power consumption. However, RESTful solutions also require a HTTP client and server embedded in the device, and is also constrained by the size of the XML-based data. Like in the DPWS case, the latter issue has motivated the adoption of EXI-like solutions [17], demanding more processing from involved devices, and requiring the usage of other protocols (such as Wired Application Description Language, WADL[4]) for describing services.

Lastly, integrated frameworks tightening the connection between sensors and IP mechanisms such as mobility, have also surfaced [18] [19]. However, the availability of such features has to be made at a gateway or sink node level (due to the demands they exert for their operation) creating different "islands" of device access frameworks. In addition, many of the solutions

[1]DPWS Specification, `http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01`
[2]EXI Specification, `http://www.w3.org/XML/EXI/`
[3]WSDL Specification, `http://www.w3.org/TR/wsdl`
[4]WADL Specification, `http://www.w3.org/Submission/wadl/`

require the deployment of different protocols for different mechanisms (i.e., device discovery and information querying), increasing the complexity of not only the supported devices but also of its users.

3. The MINDiT Framework

When addressing an IoT that encompasses entities of very distinct nature, operation requirements have to be considered in diverse scenarios and meet stringent demands. IoT goes beyond interfacing with sensors and actuators, but also includes communication with other kinds of devices as well as interfacing with high-level services. This is where MINDiT aims to contribute: providing a flexible common abstract interface for controlling and accessing the information flow coming from both high-level services, as well as low cost devices.

MINDiT is built exploring MIH concepts defined in the 802.21 standard [2] as its base. Developed to facilitate and optimize handover procedures in different wireless technologies, 802.21 provides an abstraction cross-layer called the Media Independent Handover Function (MIHF). It provides a set of core services (Events, Commands and Information) for media independent control and information obtaining from different access technologies (i.e., Ethernet, WLAN, WiMAX and 3GPP), through a common abstract interface, which can be used locally or remotely via the MIH Protocol. Entities using this interface (dubbed MIH-Users) have their abstract requests turned into technology specific primitives by the MIHF, which maps them to existing Service Access Points (SAPs). MINDiT furthers this principle, not only extending the interfacing with different devices and services in a media independent way (e.g., beyond the wireless handover use case of the standard), but also introducing a series of enhancements to the base standard, enabling it to become a valuable asset in operating IoT scenarios. MINDiT goes beyond the static interfacing nature of 802.21, and provides mechanisms that allow dynamic interface definition and retrieval based on the needs of requesting consumers. MINDiT also introduces MIH Protocol enhancements, allowing our framework to efficiently operate with both demanding high-level services, and low cost devices.

3.1. General Architecture

The objective of MINDiT is to facilitate and generalize access by agents aiming to control or obtain information from services and devices in IoT en-

vironments. To allow that, entities containing services or devices available for interfacing are coupled with the MINDiT Media Independent Function (MMIF) as well as a specific SAP. The MMIF provides a set of enhancements over the 802.21 MIHF, but maintaining its behavior as an abstraction and interaction layer, enabling such entities to exchange both L2 and/or L3 enhanced MIH Protocol messages. The SAP provides the necessary definition in MIH format, of the primitives and parameters made available by services and devices. In addition, the MMIF and enhanced MIH Protocol allow MINDiT entities to reuse the service management mechanisms of the 802.21 standard, for entity discovery, registration and event configuration, as defined in [2].

802.21 MIH-Users become MINDiT Media Independent-Users (MMI-Users) which are high-level entities wishing to interface with such devices, according to supported SAPs. Contrary to the base 802.21 framework and the MIHF, MINDiT does not consider that the MMIF comes deployed with a pre-defined set of SAPs. As shown in Fig. 1, MINDiT allows MMI-Users to identify the necessary SAPs for interfacing with discovered devices (Step 1), and request their specification from a SAP repository. This repository is an enhanced version of the Media Independent Information Server (MIIS) (which we call the Information Domain MIIS, or IDMIIS), which can be queried using the MIH Protocol (Step 2). The requested SAP is sent back towards the MMI-User (Step 3), providing it with the information required to assess and create the necessary MIH Protocol commands for interfacing with the respective IoT device (Step 4). In this way, the MMI-User is able to obtain the necessary interfaces for interacting with virtually any kind of device and service, allowing a generic IoT interfacing environment.

To further support this vision, MINDiT provides a set of important evolutions over the base 802.21 mechanisms, comprising the support of dynamic SAPs (other than just specific link access technologies as in 802.21), the definition of a single purpose action primitive to interact with entities based on retrieved SAPs, an enhanced MIIS server allowing for the query of SAPs, and the support of integrated communications using different communication models, including a proxy mechanism.

3.2. Dynamic Plug-in of Device/Service Interfaces

To support the usage of different SAPs, the MMIF required the enhancement of the MIHF to couple, in a *plug-in* manner, with different SAPs in a

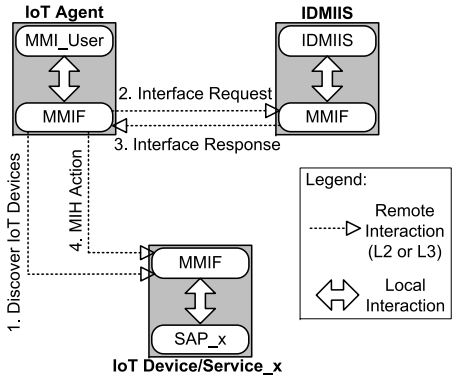

Figure 1: The MINDiT Framework Generic Architecture

dynamic way. We call this new ability "Dynamic SAPs", where we decoupled the SAPs from the MMIF, as shown in Fig. 2. Instead of implementing SAPs as a software API towards the link layers using specific driver calls, we cloned the MIH Protocol used in remote communications between peer MMIFs, and reused it in local communications between MMI-Users and the MMIF, as well as between the link layer modules and the MMIF. In this way, developers of different services and devices can create their own SAP, and couple it to the MMIF via usual MIH Protocol definitions.

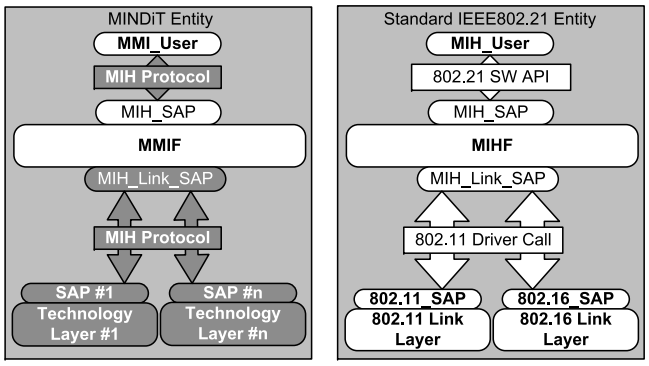

Figure 2: Decoupled SAP architecture for the MINDiT framework

3.3. Interfacing with Devices and Services

Interfacing between MMI-Users and a large number of different devices and services raises the need for an adaptive mechanism allowing access to

different primitives from different SAPs, using the same generic MIH protocol. The standard MIH Protocol defines header and payload formats, using Type-Length-Value (TLV) encoding. The payload includes the Source MMIF identifier TLV, the Destination MMIF identifier TLV and MIH service specific TLVs. The header contains, amongst others, a Service identifier (e.g., events, commands or information), an Operation code (e.g., *request*, *response* or *indication*) and an Action identifier (e.g., specific command identifier), which are filled according to the primitives supported by the MIH Protocol. The base 802.21 framework features a fixed set of MIH primitives, used for service management, handover and link control.

However, with MINDiT allowing the flexible coupling of new and different SAPs, a more generic MIH Protocol is required, supporting a different number of primitives from distinct entities. To support this, we defined a new generic *MIH Action* message that, through the inclusion of a SAP Identifier, is able to specify which actions to execute on an entity, as shown in Fig. 3. Each SAP has its own identifier, and each of its supported primitives is accessed by indicating a specific Action identifier.

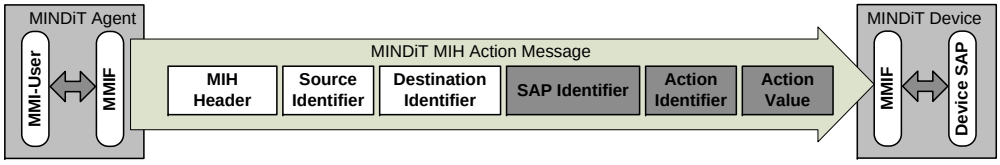

Figure 3: New Generic MIH Action Message

The Action Value parameter (which specifies how the action should operate) can be simple or complex. For simple situations, such as turning on/off an actuator, the parameter is composed by one single element/datatype, whereas in more complex situations, sequences of parameters can be provided. This generic approach allows MMI-Users to use a common message to operate the different *actions* supported by receiving devices and services. The definition of what each action procedure does in each device is provided by its specific SAP, which is obtainable from the IDMIIS, defined next.

3.4. Information Domain MIIS

By retaining the base mechanisms of the original MIH Information Server, the IDMIIS provides a queryable MMI-User that is composed by Information Elements (IEs) organized in containers reflecting a network's topology. IEs

provide details about the operator, network and Points of Attachment (PoAs) which can be used for optimizing network selection and network interface preconfiguration. IEs can be represented in either binary form, allowing TLV queries, or using a Resource Description Framework (RDF) that is queryable through the SPARQL Protocol and RDF Query Language (SPARQL). The base MIIS also provides filtering mechanisms, allowing the definition of a maximum query response size and prioritizing some IEs over others.

Our proposed IDMIIS evolves the standard MIIS, fulfilling the added role of SAP repository. The process of obtaining access interfaces towards the different elements present in the network is performed by querying the IDMIIS (as shown in Fig. 1).

3.4.1. Representation of Things - Technology, Phenomena and Primitives

In order to facilitate the definition of SAPs and primitives, we developed a simple tri-partite model that defines the relationship between devices, phenomena and primitives. Tags are used as identifiers for combinations of such relationships, aiding in querying and identifying intended behavior. In this way, it becomes possible and simpler to define the *things*, their *actions* and over which *phenomena* they operate (Fig. 4). For example, an agent can query an IDMIIS indicating that it is interested in interacting with *sensors* for obtaining *temperature* (respectively using these names as the tags for identifying the *technology* and the *phenomena*). The IDMIIS would then reply with a list of possible primitives related to temperature sensors, or even provide a whole SAP as response.

We have created our IDMIIS extensions in a way that enables deployers to define the specific mechanism used to implement this model. In our tests (and the mechanisms presented in the next subsections), we used a simple XML schema, but relational databases or more advanced semantic and ontology schemes based on Semantic Markup for Web Services (OWL-S[5]) or Web Service Modeling Ontology (WSMO[6]), using descriptions in WSDL and XML, can be devised. The important factor here is to retain the SAP definition according to our extensions defined over the MIH Protocol format.

In this way, MMI-Users can query the interfacing information to interact with different devices and services based on simple behavior representation.

[5]OWL-S Specification, `http://www.w3.org/Submission/OWL-S/`

[6]Web Service Modeling Ontology, `http://www.wsmo.org/`

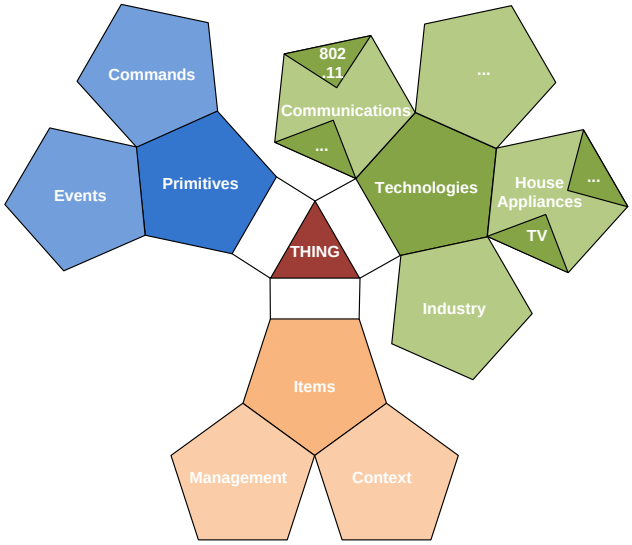

Figure 4: Tri-Partite representation

Moreover, since this process only involves the interfacing agent at the services invocation and the IDMIIS, no processing or memory restrictions are imposed from the devices represented by the SAPs.

3.4.2. Accessing Interfaces

The process of linking the semantic and syntactic description for interfacing with an entity is typically called *grounding*. Fig. 5 illustrates the full MINDiT grounding procedure, showing the involved entities and exchanged messages, composed by three phases. In phase 1, a MultiMedia Agent (MMAgent) acting as a MMI-User, after discovering a MINDiT-enabled device, queries a IDMIIS for obtaining the necessary interfacing information. In phase 2, the IDMIIS replies with the necessary interfacing information. Lastly, in phase 3, the MMAgent builds a action command based on the interfacing information received from the IDMIIS, allowing it to interact with the device. For simplicity, the figure only shows part of the messages payload.

During the query phase, through a *MIH Get Information request* message, the querying MMI-User provides tags and attributes in the form of a list, as shown in phase 1 of Fig. 5. Each tag can be accompanied by various attributes that facilitate the identification of the intended interface. When the IDMIIS receives a query in this format, it transforms this list into an

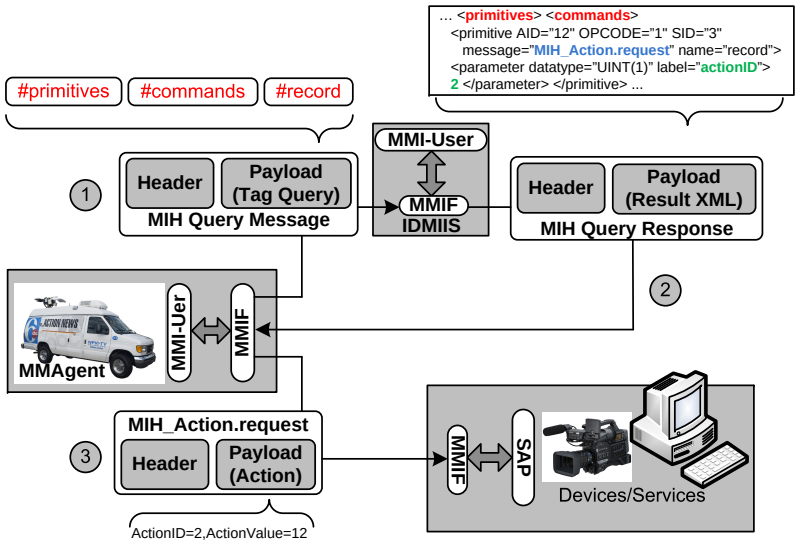

Figure 5: Grounding procedure for obtaining a SAP using a IDMIIS

XML Path Language (XPATH[7]), and applies it to the main XML Database, retrieving and sending the result, as shown in phase 2 of Fig. 5.

When the requesting agent entity receives the query results obtained from the IDMIIS, the first step is to retrieve the data regarding intended primitives from the XML (e.g., the OPCODE, SID, and AID values) to construct the MIH Header of the MIH Action message (as shown part 3 of Fig. 5). All entities of the resulting XML can provide tags that may be used as guidelines for the grounding or to simply provide extra information on the parameter.

3.4.3. Querying Support Mechanisms

As implied before, our framework also extends the filtering mechanism supporting queries, allowing the requesters to send queries with different degrees of detail. It enables querying agents to execute phased queries, using the information obtained from responses to refine sequential queries, with the aim of better pinpointing the intended interfacing primitives. Considering the example depicted in Fig. 5, instead of asking directly for elements present in the network that comply with the three tags at once, we could divide the query into three separated messages. In this case, first the MMA would search

[7]XML Path Language Specification, `http://http://www.w3.org/TR/xpath/`

for all available primitives and, depending on the ones discovered from the corresponding response, it could create another query message to retrieve all available commands for a specific technology. Again, through the received results, it could create one last query specifying the required SAP.

An often overlooked procedure is how interfaces are added to servers providing them. The IDMIIS is able to have definitions of new SAPs pushed into its information structure, using a modified *MIH Push Information request* message. In its original 802.21 form, this message is only available for the operator to push network information towards a user terminal. We have enhanced it to allow service providers, manufacturers and other parties, to push SAPs into the IDMIIS.

3.5. Dissemination Models

In 802.21, MIH Protocol messages are exchanged between peer MIHF, enabling a MIH-User in one entity to interface with the SAP existing at another. However, low cost devices might have hardware limitations, and might not be able to include a full MIHF and/or SAP module in their stack.

By decoupling the SAPs from the MMIF (as identified in section 3.2), MINDiT explores the exchange of remote information between these two modules, allowing the deployment of Gateway nodes which contain a MMIF, and interface remotely with devices which are only capable of incorporating a simple SAP. Through the realization of discovery and registration procedures, the Gateway node can discover SAP-enabled devices, and act as a serving MMIF on their behalf and simplifying their design.

Considering even more stringent device deployments (e.g., devices with closed APIs, or legacy devices), MINDiT is able to employ a SAP module in the Gateway node, enabling it to translate the MIH Protocol into whatever API or protocol necessary to interface with the target device. In this way, the MINDiT framework is able to encompass different scenarios, according to the capacities of the involved devices and services. [20] provides details on the deployment of such dissemination models.

3.6. Proxied Actions

IoT environments provide different connectivity deployments and opportunities for the devices and agents operating therein. It is possible that some types of devices are connected to a specific infrastructure, while others are made available in ad-hoc manner. Considering these matters, we extended

our dissemination models to provide the means for MINDiT to support proxying. Our design allows MMI-Users to send commands to third-party entities (after gaining knowledge of them through the proxy entity), using the proxy node as an intermediary. For this procedure we introduce a Proxied Destination Identifier header, which allows the receiver of the message to forward it towards other entity, and then retrieve its response back to the original source (exemplified in Fig. 6).

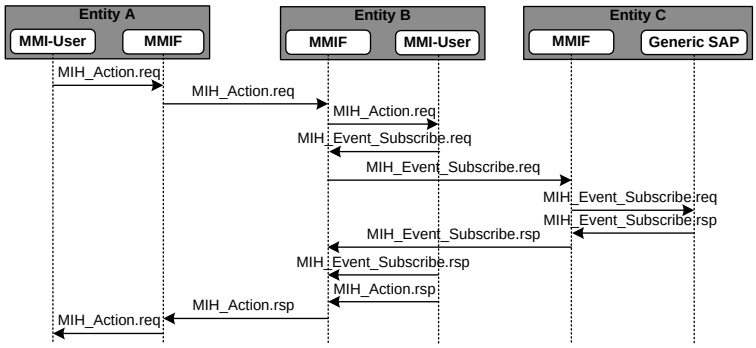

Figure 6: Sequence Diagram of Action Message for Proxy

The proxying mechanism design presented here focuses on the connectivity provisioning mechanism (in deployments of this framework, security and authentication considerations are important). For simpler deployments, the execution of the proxying mechanism can be dependent of configuration parameters of the MMIF and Proxy MMI-User: MMIFs can be configured to support access from all proxy requests, or ignore them. Also, the execution of MIH Protocol procedures can be integrated and occur only after specific link access technology security and authentication procedures (e.g., 802.1X) have been executed. The 802.21 standard itself does not provide any considerations regarding secured and authenticated interactions between peer MIHFs (such efforts are currently being pursued in a new amendment, 802.21b). Although we do not address the deployment of such procedures for our framework in this paper, we argue that its flexible design is able to encompass the definition of secure and authenticated communications at the MMIF level, incorporating them into discovery and capability exchange procedures.

4. Comparison with Other Frameworks

The deployment of solutions providing access to different kinds of devices and services in IoT environments have almost naturally resulted in SOA-based architectures using Web Services for the realization of necessary procedures. In this section, we highlight the features provided by MINDiT, identifying them as the evolutions made over base 802.21 mechanisms for the enablement of such functions, and compare them with traditional Web Services based technologies.

4.1. Functional Comparison

Table 1 presents a functional comparison on the features provided by the base 802.21, our MINDiT framework and generic Web Services based frameworks.

Related with feature (1), MINDiT facilitates information inter-exchange by generalizing it into events, commands and information elements. It inherits the base structure from 802.21, but enhances it with the ability to go beyond handover-related operations, and a static set of SAPs. Our extensions on 802.21 bring it closer to the interface dissemination mechanisms such as WSDL in web services-based deployments. It actually goes one step further by providing a common protocol for both accessing the interfaces and using them, unlike web services which can require different protocols and XML structures, potentially hindering deployment.

About (2), MINDiT reutilizes L2 and L3 discovery mechanisms from 802.21, providing for capability interexchange using the same protocol. Web-service based frameworks supports discovery using specific protocols (such as the Universal Description Discovery and Integration (UDDI[8]), thus requiring the additional support of yet another protocol by the framework.

Feature (3) is also supported by all three frameworks, with MINDiT enhancing the 802.21 event service beyond-handover features. This allows it to have an uniform way of reaching for events in IoT devices and services, as well as defining thresholds and period for event reporting. Web Services handle this information inter-exchange in a P2P fashion: each device (or other services) needs to implement the means for its integration with the event registration and dissemination mechanism of the framework, whereas MINDiT is able to use the same event service over all SAPs.

[8]UDDI Specification, `http://www.uddi.org/pubs/uddi_v3.htm`

Num.	Feature Description	IEEE802.21	MINDiT	Web Services
1	Defines a uniform description for information dissemination as well as entity interfacing	–	++	+
2	Provides the means for discovering, determining capabilities and registering at other framework-enabled entities	+	+	+
3	Allows the subscription of events, directly at the services and devices providing them, allowing threshold and period for event reporting	+	++	++
4	Remote interaction is able to be sent either via L2 or L3 protocols	++	++	–
5	Provides abstracted services to higher layers, interfacing them with both other higher-layer services (e.g., context and content servers) as well as lower layer technologies (e.g., sensors and actuators)	–	++	+
6	Different dissemination models	–	+	++
7	Integrates proxying mechanisms to extend access to other framework-enabled entities beyond the discovery range	–	++	+
8	Provides a uniform protocol, able to be used for both service management/control, as well as operating entities according to the obtained interfaces	–	++	+
9	Flexible enough to support different interface definition schemes	–	++	++
10	Supports Energy-Efficient operation	+	++	–

Table 1: Functional comparison between base IEEE802.21, MINDiT and Web Services based frameworks ("plus" is better)

Feature (4) refers to the transport capabilities for remote interaction. The MIH Protocol used in MINDiT can be transported over different L2 (e.g.,Data and Management frames) and L3 (e.g., UDP, TCP and others) transport mechanisms. By supporting L2, MINDiT is able to reach non-IP devices, whereas Web Services provide application level protocols only.

(5) highlights a major feature, where the base 802.21 abstractions are extended in MINDiT to access not only different link layers, but also different high-level services. This is in fact an evolution over the web services approach as well, since their realization requires device support of application protocols and mechanisms required by web services (which can hinder deployment in low cost devices).

Feature (6) highlights the enhancements done over 802.21 by MINDiT allowing not only the interaction between peer-MMIF entities, but between entities acting as gateways for low-powered devices. Web Services do not impose any dissemination model.

In (7) MMI-Users in MINDiT can obtain the interfaces and operate other services or devices through entities working as proxies, unlike 802.21 and Web Services architectures. The last ones, however, can achieve this behavior by introducing special mechanisms in combination with HTTP proxies. However, there are no specific definitions of proxy mechanisms at web service

level, that enable reaching the procedures of third-party nodes.

(8) highlights the extension of the MIH Protocol to support IoT interaction, in an uniform way: the MIH Protocol in MINDiT is an adaptative mechanism, working as a transport mechanism between the framework-enabled entities, allowing their management and operation. Web Service frameworks can use different SOAP-based protocols, but these produce varied flavors for interacting with entities (e.g., the XML structure of the different protocols involved).

In terms of supporting the grounding for the interfaces of different entities, feature (9) shows that although MINDiT has been tested with a simple XML definition of a tag-based tri-partite definition of technologies, phenomena and devices existing in a IoT, more refined methods can also be employed (e.g., OWL, relational databases). This represents an important evolution made over 802.21 for the support of IoT scenarios. This feature is also a core concept of web-based solutions.

Lastly, (10) concerns with operating in environments where low-power consumption is important. In multi-interface battery-powered devices, both 802.21 and our framework allows obtaining information about different technologies, without having to power up the respective interface, by querying the MIIS. MINDiT also supports optimized energy-aware L2 transport. Web Services based frameworks do not possess any intrinsic support for this, thus being less energy-efficient than MINDiT.

4.2. Impact on Device Procedures Comparison

To further analyze the effectiveness of the generic interfacing capabilities provided by the MINDiT framework, we developed a comprehensive study focusing on the signaling size impact on a sensor device, against a set of Web Services based frameworks: DPWS, REST and DPWS featuring EXI encoded messages. A simple testbed was assembled, featuring a Sun SPOT[9] mote connected via USB to a Linux Laptop, acting as Gateway. Sun SPOT devices contain a JAVA programmable embedded microprocessor and have simple networking capabilities using 802.15.4, with limited processing capabilities (512KB SRAM, 4MB Flash and a 400MHz ARM 926ej-S processor). It contains various kinds of sensors, providing information on luminance, accelerometer and temperature. We simulated a scenario where the Sun

[9]Sun SPOT World, Oracle Labs, `http://http://www.sunspotworld.com/`

SPOT mote randomly accesses one of the sensors, encapsulates that information according to the format of the underlying framework, and sends it over the wireless interface with different event number generation rates. For each framework, we implemented the necessary event messaging format directly in the sensor, according to that framework specification. In the case of MINDiT, this evaluation was done using a prototype implementation built over ODTONE[10] [21], an open-source IEEE 802.21 implementation. For all frameworks, we analyzed in the sensor the amount of time for sending the messages, the battery consumption and memory usage. As a guideline, we also compare these values when sending this information using raw sensor data, to verify the impact caused by the Java Event Device Drivers in operating the devices. Each execution set of multiple events was run 100 times, a trade-off between execution time and confidence intervals. Tab. 2 presents the size of the messages used by each framework. Take into consideration that string-based frameworks vary the message size depending on the the value being measured (i.e., more bytes allocated for extra characters).

Framework	Message Size (bytes)	Total Allocated Memory (Kbytes)
Sun SPOT Raw	3	206.013
MINDiT	60	213
DPWS	551-553	209
DPWS w/ EXI	377	226
REST	219-221	214

Table 2: Message Size and Memory Utilization per Framework

4.2.1. Average Time for Sensor Event Dissemination

These results account for the average wireless utilization time taken by the sensor to send an event message through its wireless interface. As shown in part a) of Fig. 7, results are impacted by the different framework message sizes, when the event generation rate is increased. The most impacted framework was DPWS with an average time of 576ms, followed by DPWS using EXI (431ms), REST (292ms), MINDiT (105ms) and raw (98ms) (in this order and for the case of 10 events being sent). Compared to the other

[10]ODTONE - Open Dot Twenty ONE, `http://atnog.av.it.pt/odtone`

frameworks, MINDiT has a very low wireless utilization time to send the same event. In average for the different rates of event generation, MINDiT requires less 83% wireless utilization time than DPWS, as well as 77% and 66% for DPWS using EXI and REST. In fact, it only requires a marginal increase of 6.6% more time than the raw SUN SPOT information to be sent.

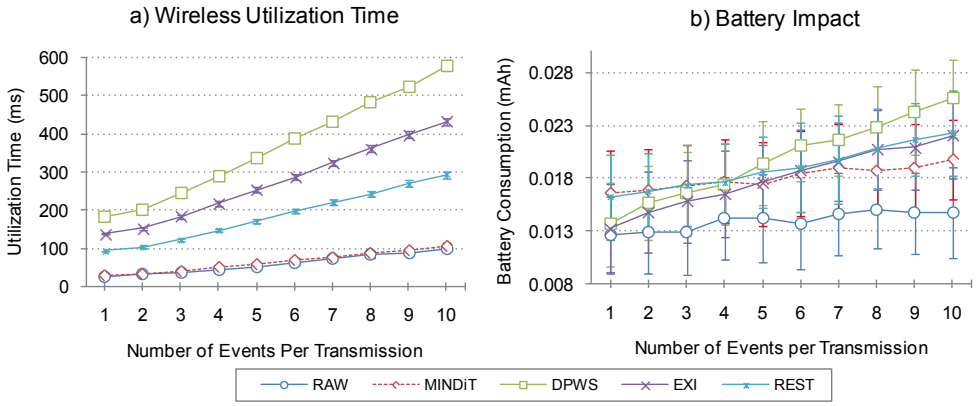

Figure 7: Average Wireless Interface Utilization Time and Battery Consumption

Due to its relatively small message size requirement, MINDiT is able to send its events and free its wireless resources faster. This not only allows for the next message to be sent sooner, but also requires the wireless interface to be active for a lower amount of time, saving energy.

Tab. 2 also presents the impact that the different signaling formats place over the Sun SPOT motes. The amount of allocated memory was measured in the device after each message transmission. It was verified that the message generated rate did not impact the results. Values include the Sun SPOT SQUAWK Java Virtual Machine footprint. These show that the most memory-demanding signaling scheme is DPWS with EXI, which requires more 10.2% allocated memory and 15.5% freed memory, when compared to Sun SPOT raw. This is followed by MINDiT (3.4% and 5.7%), REST (4% and 6.3%) and DPWS (2% and 3%), in this order respectively. In this way, MINDiT is able to reduce its signaling size, making its memory utilization on-par with other XML-used SOA approaches, while requiring less memory resources than binary-format SOA mechanisms.

4.2.2. Battery Consumption

Regarding battery consumption, results show very small variations considering the battery total capacity of 720mAh. Focusing on the consumption differences between frameworks, as can be seen in part b) of Fig. 7 (confidence intervals varied between 0.003 and 0.004 mAh and were removed from the figure to improve readability), greater event generation rates show an increase in battery consumption for all frameworks. The SUN SPOT raw provides the lowest consumption of all and, with a low variation of 0.002 mAh between its maximum and minimum consumption values, and provides an indication bound for the different rates. It is interesting to notice that up to 5 events generated, the different formats are able to exercise a similar or lower battery consumption, when compared to MINDiT (in average, DPWS, DPWS with EXI and REST consume less 7.5%, 11.7% and 0.9% respectively, between 1 and 4 events sent). Past this point, DPWS consumption reaches to 0.026 mAh, in the case of 10 events generated, (more 22.5% and 42.1% consumption than MINDiT and raw, respectively), whereas MINDiT becomes the least consuming of the four frameworks with 0.019 mAh (more 25.4% than raw). Thus, for greater event rates, the lower message size from MINDiT not only allows for a lower wireless utilization time, but lower battery usage as well.

5. Scenario Test Case

This section presents a deployment scenario for MINDiT, considering a multimedia musical concert for the grand opening of a major shopping mall. The location has the latest technology in ambient intelligence and assisted living, integrating sensors (e.g., light, temperature), actuators (e.g., mechanical window shades), devices (e.g., surveillance cameras) and services (e.g., shop advertising), all interconnected by our connectivity framework. The mall owners have decided to cover the concert by their own news report team, allowing them access to all the devices and services of that area. The crew has got a camera, but also have a MultiMedia Agent (MMA) able to connect to our framework, for controlling the video production with the aim of generating a high-quality news coverage of the event.

5.1. Phase 1 - Discovery

In the first part of Fig. 8, the MMA accesses MINDiT to discover and dynamically learn how to use the devices and services in its surrounding,

finding input for the news cover. The unit is deployed in the concert hall and the discovery mechanisms of our framework alert it to the existence of several smartphones from the mall personnel (coupled with multimedia abilities), as well as the cameras from closed-circuit television (CCTV). The MMA is programmed to discover the capabilities of the devices by broadcasting a *MIH Capabilities Discover request* message, to which entities within range respond providing their capabilities.

5.2. Phase 2 - Grounding

After learning the capabilities, the MMA obtains the necessary SAPs to interface with the cameras of the smartphones and the CCTV, with the objective of activating them and obtaining different perspectives and video-effects for covering the concert. As shown in the second part of Fig. 8, this is achieved via a query to the IDMIIS, providing tags existing in the MMA routines, related to *VideoCamera* (e.g., the technology), *Picture* (e.g., the phenomena) and *Record* (e.g., the primitive), included in a *MIH Get Information request* message. Upon receiving this, the IDMIIS is able to process the received query message (as defined in 3.4.3) identify the proper interfacing SAP, providing it to the MMA in a query response.

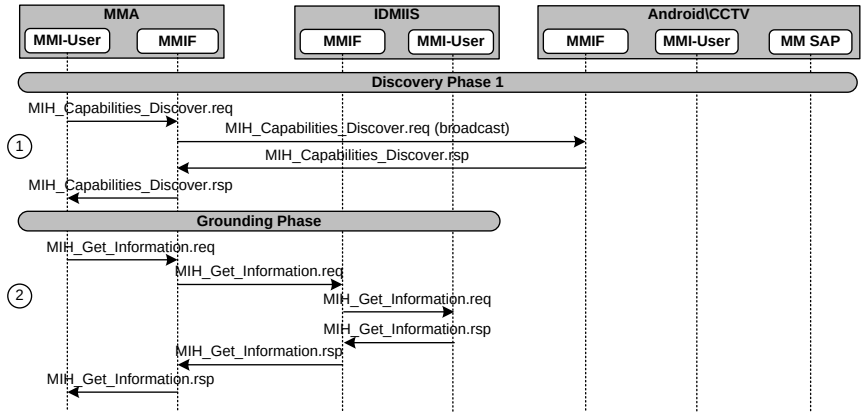

Figure 8: Discovery and Grounding Phases (Phase 1 and 2)

5.3. Phase 3 - Smartphone and CCTV Interface Access and Usage

With the received SAP, the MMA agent is able to create the necessary *MIH Action request* primitive to interface with the CCTV, and request it to start recording and send the feed back to the MMA agent, as shown in Fig. 9.

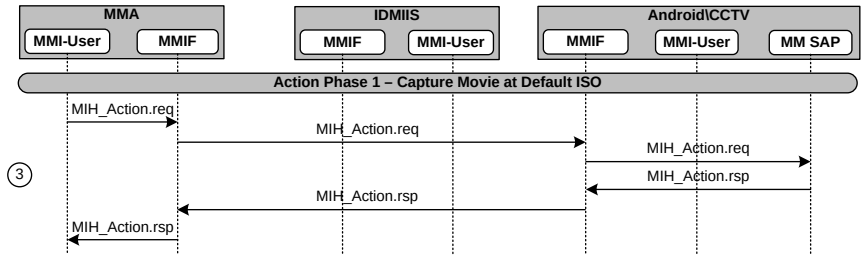

Figure 9: Action for Capturing Movie at Default Quality (Phase 3)

5.4. Phase 4 - Proxy Device Discovery

While experimenting with the video feed, the news crew verifies that the lighting multimedia system used for special effects during the concert is very dynamic, and can potentially hinder the quality of the video shooting. Since the CCTV cameras are spread in different spots of the concert hall, the lighting parameters can be different for the distinct devices. The MMA determined, in the discovery phase, that certain smartphones also possessed a proxy-dedicated MMI-User. With this, the MMA agent is able to interface with services and devices beyond its connectivity range, and instructs the proxy to do a device discover in their surroundings, learning that some of the smartphones are within range of the shopping complex luminance sensors. To achieve this, the MMA sends a *MIH Action request* message to issue a proxy broadcasted *MIH Capability Discover request* to have the smartphones execute a discovery in their surrounding, as shown in Fig. 10. The respective response provides the MMA with a list of found devices with the ability to be controlled as luminance sensors.

5.5. Phase 5 - Proxy Sensor Event Subscription

By using the smartphones as a proxy, the MMA uses the generic MIH protocol service management primitives to subscribe to luminance events on the detected sensors, as shown in the first part of Fig. 11. There, a *MIH Event Subscribe request* message allows for the definition of a threshold, indicating the luminance value that is configured in the Sensor SAP for triggering the event. In this way, our framework is able to optimize network usage, by allowing the event information to be sent at a specific time, instead of periodically or requiring a constant query/response approach.

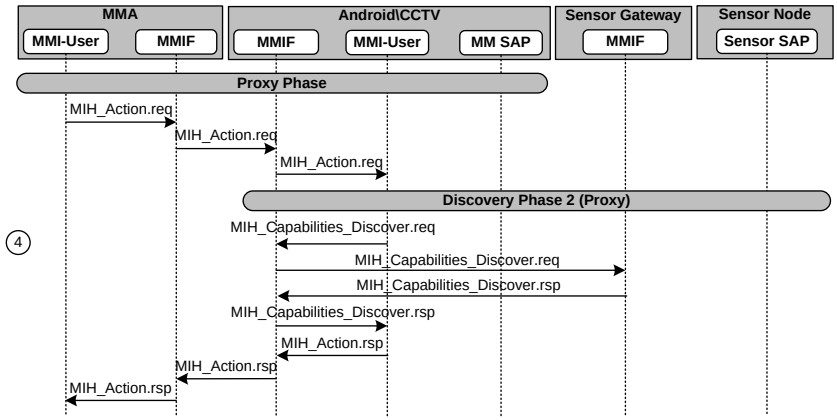

Figure 10: Proxy Discovery (Phase 4)

5.6. Phase 6 - Proxy Sensor Event Triggering and Dissemination

When the sensors obtain a luminance value that crosses the pre-defined threshold, an MIH event is sent towards the smartphones. The event is then proxied towards the MMA, as seen in the second part of Fig. 11.

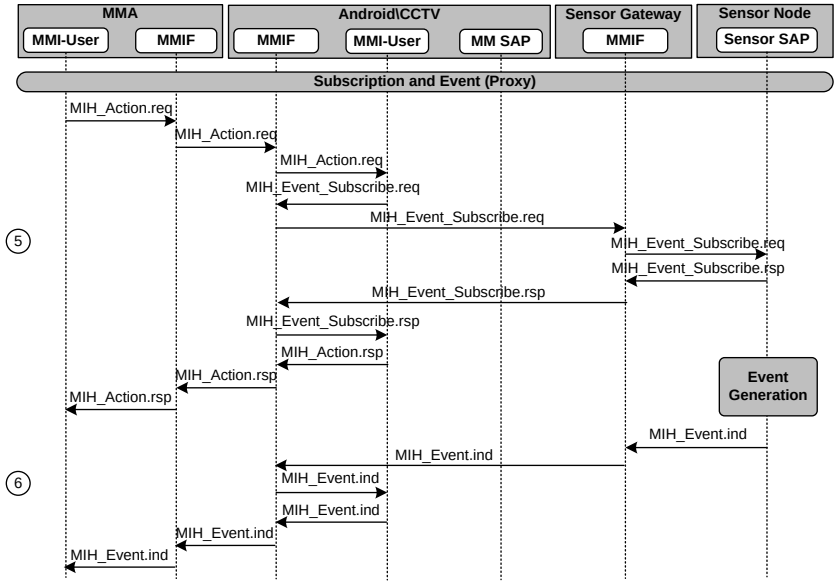

Figure 11: Proxy Subscription and Event (Phases 5 and 6)

5.7. Phase 7 - Optimal Multimedia Configuration using Sensor Information

Using the event information related to the luminance perceived by the sensors, the MMA is able to instruct the camera recording configuration to dynamically adjust video parameters (e.g., type of environmental lighting, ISO, etc.), and optimize the recorded video quality. For this purpose a new *MIH Action request* message is sent towards the CCTV cameras (as shown in Fig. 12) with new video recording parameters.

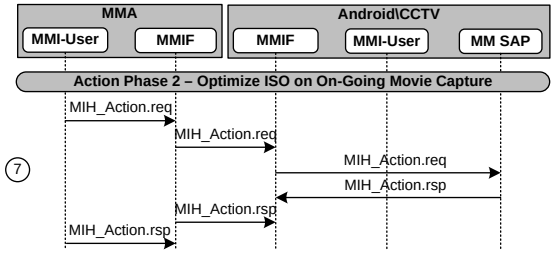

Figure 12: Proxy Action Optimizing Movie Capture (Phase 7)

6. MINDiT Framework Results

This section evaluates the performance of our framework when subjected to the operations defined in the scenario presented in section 5. Even though MINDiT supports multiple scenarios, we present the footprint placed by our approach in this example, focusing on its flexibility aspects by deploying its mechanisms in different kinds of devices. As such, we measure time duration and size of the signaling, considering the different phases on the scenario, and their impact in the complete scenario. We further explore the scalability of our framework, by evaluating its behavior and signaling impact in the different devices, when subjected to different message generation rates.

6.1. Testbed Implementation

Beyond the implementation of the MIH extensions defined in section 3 and compared in section 4.2, we have also implemented different MMI-Users and SAPs to operate with different devices, as well as a IDMIIS to provide SAP querying.

Fig. 13 shows our scenario testbed composed by two laptops connected via a Wi-Fi infrastructure to a Linksys WRT54G wireless router, and a desktop PC connected via Ethernet. All entities are coupled with our MMIF, except the wireless router which only provides connectivity between the nodes.

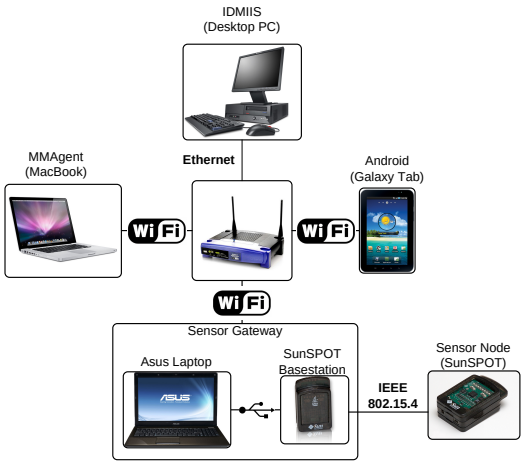

Figure 13: Testbed deployment

The first wireless laptop is a Apple Macbook (2.4GHz CPU and 2GB RAM) coupled with a MMI-User acting as a MMA. The desktop PC connected via Ethernet (Pentium 4 3.2GHz CPU and 1GB RAM) is coupled with a MMI-User acting as a IDMIIS. A Samsung Galaxy Tab (1GHz ARM Cortex A8 CPU, 512MB RAM, 2.2 Android Froyo firmware, 2.6.32.9 kernel version) coupled with a Proxy MMI-User and a Multimedia SAP allows media independent access to its video camera controls. This device fulfills the role performed by the smartphones identified in the scenario. Finally, our second laptop connected to the Wi-Fi infrastructure (T5450 Intel Centrino Duo CPU and 2GB RAM) is coupled with an USB Sun SPOT Basestation, connected to a Sun SPOT sensor via IEEE802.15.4. All the MINDiT entities defined in this testbed operate under the signaling defined in section 5. To simplify the scenario deployment, the Android device doubles as the proxying entity as well as the CCTV, due to its camera and processing capabilities.

6.2. Signaling Footprint

For each scenario phase, we measured the time taken by the internal procedures of each entity involved in the information exchange. In Tab. 3, we present these local results per entity, as well as the total amount taken by the local and remote procedures. We also present the total time taken by the procedures of each entity for the duration of the phases, and the whole scenario, as well as total local and remote times. Results presented

reflect average values taken from executing the scenario ten times, presenting confidence intervals calculated at 95%.

Phase	Entity				Total Local	Remote	Total Time
	MMA	IDMIIS	Android	Sensors			
1	29.4±3.26	0.0±0.0	21.5±4.53	0.0±0.0	50.9±5.95	145.2±37.97	196.1±40.24
2	4.1±0.35	17.73±2.88	0.0±0.0	0.0±0.0	21.9±3.10	3.3±0.51	25.20±3.19
3	61.5±16.46	0.0±0.0	2411.7±118.75	0.0±0.0	2473.2±120.75	147.1±91.1	2620.3±118.94
4	3.0±0.29	0.0±0.0	21.8±8.15	2.50±0.33	27.3±8.25	26.20±10.6	53.5±8.79
5	3.0±0.41	0.0±0.0	25.0±6.98	283.7±10.41	311.7±11.95	286.8±58.86	598.50±62.81
6	1.1±0.20	0.0±0.0	49.4±3.33	263.0±57.75	2313.5±116.43	139.7±57.86	2453.2±121.72
7	61.5±16.46	0.0±0.0	2411.7±118.75	0.0±0.0	2473.2±120.75	147.1±91.1	2620.3±118.94
Total	163.6	17.73	4941.1	549.2	5671.63	2894.8	8567.1

Table 3: Scenario Delay in milliseconds with totals calculated using the average values

We can verify that the total amount of time taken by phases 4 to 7 is twice as much that from phases 1 to 3, composing almost two thirds of the total scenario duration of 8.5 seconds (in average). Further analysis also reveals that, in average, each phase sees 66.2% of its duration dedicated for local processing, with the remaining time for remote message sending over the air (the smallest and largest values are respectively 26% for phase 1, and 94% for phases 3 and 7). Three main points have to be considered: i) phases 3 and 7 involve accessing the video camera functionalities of the Android device, ii) phases 4 to 7 contain proxy behavior whose communications have to pass through the proxy entity, (whereas phases 1 to 3 are done in an end-to-end manner) and iii) interfacing with the Sun SPOT sensors is done over 802.15.4 which are operations that take a considerable amount of time.

In fact, when analyzing the impact in time duration per entity, the Android device is the most solicited entity, being involved in 58% of the duration of all the phases in the scenario, when considering local processing times (85.6% of its time in phases from 1 to 3, which are end-to-end). This is quite a large amount of time when compared with the local processing in other entities (e.g., 2% for the MMA, 0.2% for the IDMIIS and 6.4% for the sensors in total scenario duration). We investigated further the amount of time involving the Android device, and discovered that the main impact factor was the activation of the video procedures (e.g., these interactions take about 92% of the time duration of phases 3 and 7). We did separate experimenting with the primitives available by the Android Software Development Kit on the Samsung Galaxy Tab, and, from the local processing values obtained by that device in phases 3 and 7 (which took over 2 seconds), around 95% of the time was dedicated to Android-specific processes involved in activating

and starting the recording in the video camera of the device. As such, in those phases, the time taken for actual MIH processing was around 131ms, and 147.1ms for remote message sending and receiving.

We have also analyzed another source of delay in the scenario, which was the time taken to produce the event sent by the sensors, with the luminance information. In fact, the local procedures belonging to phase 6 took around 94% of the total phase duration. Due to erratic values being obtained by the luminance sensors, we analyzed the internal hardware procedures provided by the sensor available on the Sun SPOT board and verified that accessing the sensor values took (in average) 39ms. However, in order to allow an accurate value to be sent to the MMA, the pre-defined threshold configured at the sensor SAP required an assessment of the average luminance value of the past two seconds. When this average value crossed the defined threshold, the event would be triggered and sent. This was, however, a design decision implemented to deal with the specificities of the values produced by the Sun SPOT sensors and is independent of our framework. In fact, MINDiT is flexible enough to allow such case-by-case implementations to be included in the SAP, to better integrate with the diversity of devices and technologies, allowing for different results if other kind of sensors were used.

Considering the impact produced by remote MIH message signaling, phase 5 takes the longest time. This is due to the fact that it consists of a proxy command being sent by the MMA, proxied by the Android device and acted upon the sensors, which send a response indicating the command execution result all the back to the origin (also, through proxying). This means that, in total, six remote message exchanges were executed (including the 802.15.4 path between the Sun SPOT basestation and sensors), for a total of 286.6ms.

An important factor to note is that the SAP querying (e.g., Phase 2) requirements in terms of time are quite low (around 4.1ms for the MMA and 17.73ms for the IDMIIS, for a total time of 25.20ms). This highlights the feasibility of MINDiT to adopt more powerful (and of course more demanding) methods and mechanisms for the definition and querying of device and services interfacing, as pointed out in section 3.4.

6.3. Data Footprint

According to the results presented in Table 3, the largest verified performance impact was the local processing of device-specific controlling aspects (e.g., camera activation in the Android Tab and erratic sensor values). Remote signaling times have been minimal, despite the different wireless and

wired technologies involved. Considering this, we analyzed the amount of signaling information exchanged by each node and in each phase, presenting the results in Table 4.

Phase	Entity				Total
	MMA	IDMIIS	Android	Sensors	
1	72	0	37	0	109
2	347	80	0	0	427
3	70	0	34	0	104
4	49	0	93	37	179
5	70	0	80	46	196
6	51	0	51	0	102
7	70	0	34	0	104
Total	729	80	329	83	1221

Table 4: Amount of information remotely received in bytes during the scenario

The total amount of information received over the air using the MIH protocol in this scenario was 1221 bytes, considering only MIH Protocol sizes. The entity most involved in this message exchange was the MMA, which was responsible for 60% of the information received. Of that information, 67% was related to phases from 1 to 3. In fact, phase 2 was the most data consuming procedure, involving 347 bytes from the MMA and 80 bytes from the IDMIIS, highlighting the importance of the query for obtaining the SAP.

In contrast, the sensor device was the second least entity involved in the message signaling exchange (6.8% against 6.6% of the IDMIIS). This emphasizes the feasibility and flexibility of MINDiT in interfacing in different ways with low cost devices. In this particular case, the usage of MIH mechanisms allowed the definition of a threshold value to trigger the sending of an event with luminance information towards the MMA, only when such threshold value was crossed.

The second most involved entity in the signaling exchange was the Android device, participating in 27% of the overall remote information. Of these, 78% belonged to the set of phases from 4 to 7, indicating that this message amount is due to the proxy mechanism. But even so, the Android device was only involved in sending 258 bytes of information in that group of phases, which is a small value for a device with its capabilities.

Not considering the query phase with the IDMIIS, the average amount of

information exchanged per phase is 132 bytes. This is more predominant in the set of phases from 4 to 7, due to the proxy behavior between the MMA and the sensors, using the Android as a proxy.

6.4. Performance Evaluation

We submitted the different nodes involved in the previous scenario to a command message being generated at different rates. The objective is to analyze the framework impact over the different kinds of nodes, in increasingly stringent conditions. For these tests, the MMA transmits 100 *MIH Action request* messages towards the sensor node, which sends a respective *response* message (both using the Android as proxy).

6.4.1. Average time per packet

Fig. 14 presents the average packet reception rate of the packets received by the nodes. Results highlight that both the *request* (Fig. 14-a) and the *response* (Fig. 14-b) reception rate are slightly affected by the different message intervals. Part a) of the figure distinguishes the performance of the WLAN network involving the MMA, the Android and the GW (39.13 ms in average), and the 802.15.4 network (679.12 ms in average). This clearly identifies the sensor communication as the most limiting factor in terms of message reception rate. This is further emphasized by part b) of the figure, where the delay caused by the sensor reception rate of the *request* message, is propagated all the way back towards the MMA for the reception of the *response* message (average reception rates of 672.02 ms for the gateway, 684.36 ms for the Android and 701.70 ms for the MMA).

6.4.2. Incomplete Transactions

We measured the percentage of incomplete transactions (due to packet loss), in terms of unsuccessful exchanges of *request* and *response* messages between each pair of nodes involved. Fig. 15 shows that incomplete transactions are only verified between two entities that share the WLAN medium: the MMA and the Android. We can see that the values start high (46% for the MMA and 24.3% for the Android) for low message intervals. These decrease considerably for larger message intervals, stabilizing at 25 ms interval time (11.75% for the MMA and 4.16% for the Android). It is interesting to note that there are no incomplete transactions with the sensor node or the gateway. With some of the messages between the MMA and the Android

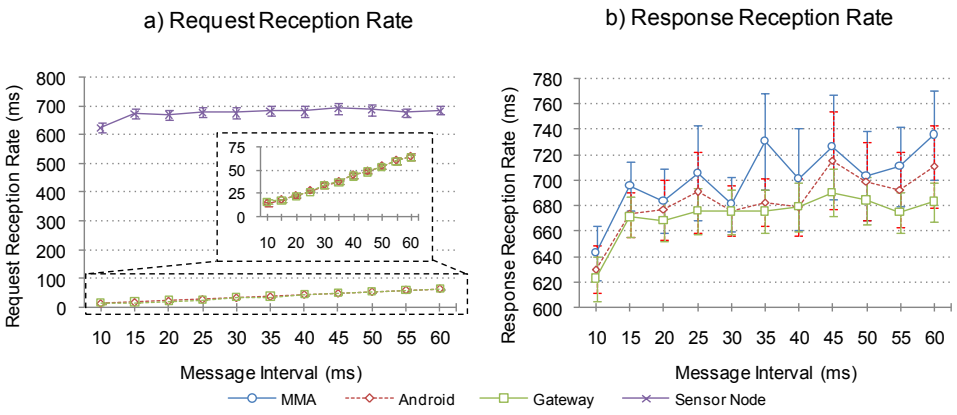

Figure 14: Average Packet Reception Rate

being lost, the rate at which they arrive to the gateway decreases. Considering the slow sensor network response verified in section 6.4.1, the rate at which the *response* messages reach the gateway is even lower. This means that there are not only fewer messages on the way back, but they are also slower. We performed evaluations on the scenario regarding the amount of lost packets on the return path, confirming that these never raised above 4 packets per test.

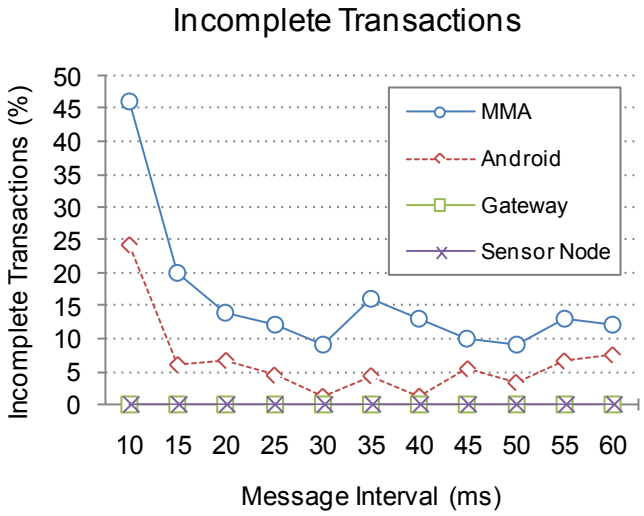

Figure 15: Percentage of Incomplete Transactions

6.4.3. Amount of Packets Exchanged

Fig. 16 shows the direct consequence of the incomplete transactions affecting this metric: with fewer requests reaching the gateway, fewer responses also are returned. This provides an approximate difference of 86 bytes between the amount of data received by the MMA/Android and the gateway/Sensor pairs. Like in the previous case, it is noticeable that the values start to stabilize past the 25 ms message rate. Results also indicate that, with shorter message intervals, the amount of packet loss reduces the amount of data received by the MMA, which can impact its utilization of the devices interfaced.

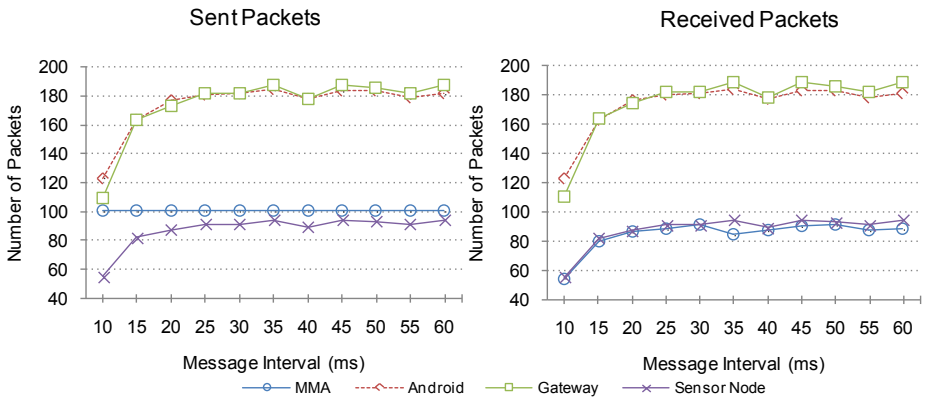

Figure 16: Amount of Sent and Received Packets

6.5. Handling Incomplete Transactions

Motivated by the number of incomplete transmissions, we devised a simple reliability mechanism similar to the one defined by 802.21: when the *response* fails to be received by the source of the original *request* message within a timer-defined window, the later is re-sent. For simplicity, we evaluated this mechanism only at the MMA, which is the original sender of the command. Fig. 17 shows the total number of necessary retransmissions for obtaining 100 successful transactions, as well as the number of received, sent and lost messages. Results show that all metrics gradually decrease their value as the timer duration increases, stabilizing at 800 ms. Past this timer configuration, the number of retransmissions drops to nearly zero, which also reduces the other metrics.

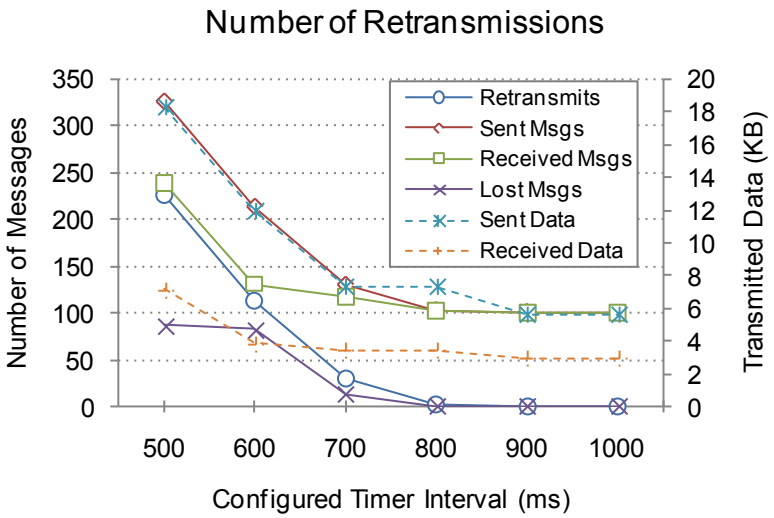

Figure 17: Number of Retransmissions

Fig. 18 follows the trend of the previous metric, indicating that the average time for receiving a *response* with reliability active stabilizes at a 800 ms configured timer. Moreover, these last two metrics indicate that retransmission timers below that value tend not only to generate more transmissions but also larger average time. Timer configuration is thus an important factor to be considered in these heterogeneous environments: its incorrect configuration can lead to over-zealous side-effects, where a small timer can trigger a *request* re-send while the *response* is still in transit, originating superfluous retransmissions and overhead.

7. Conclusions

In this paper, we presented MINDiT a novel framework whose features enable the operation of IoT scenarios for ubiquitous smart environments.

Our framework relies on a set of profound evolutions from the standardized IEEE 802.21 MIH (Media Independent Handovers) framework and its services, expanding them for the facilitation and optimization of media independent access to IoT devices and services.

We proposed the concept of Dynamic Service Access Points (SAPs) that allow the definition of interfaces towards different devices and services using a unique communication procedure, based on an unified and enhanced MIH Protocol from the IEEE 802.21 standard. We have defined an Information

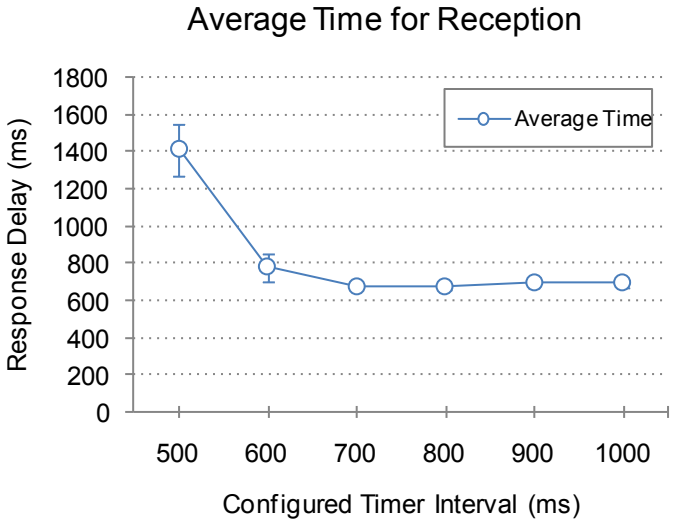

Figure 18: Average Time for Successful Response

Domain Media Independent Information Service (IDMIIS) server, with the ability to store SAPs from different devices and services, allowing for the integration of different query mechanisms for accessing interfaces (ranging from simple XML query procedures to powerful high-level semantics and ontologies). To allow its deployment in a broad set of cases (ranging from interfacing with sophisticated services residing in powerful mainframes using optical wired connections, to low cost devices using simple wireless networking stacks) we have defined a proxy mechanism aiming to increase the interfacing opportunities with different kinds of devices and services.

To validate the benefits of our proposal, we implemented a prototype based on an existing open-source IEEE 802.21 implementation (ODTONE). The message signaling mechanisms from MINDiT were compared to other SOA-based frameworks using Web Services (also implemented), showing that its lower message size required less resources utilization. To illustrate its flexibility, MINDiT was also deployed in a multimedia-enabled smart environment scenario featuring agents, mobile terminals in wireless environments and sensor technologies. The footprint imposed by our mechanisms was evaluated in terms of interfacing procedures, and information exchange requirements. Furthermore, to analyze the scalability, we studied the framework response when subjected to different rates of signaling generated.

Results demonstrated that our framework does support a broad range of

features while requiring a significant low amount of information. This has the benefit of facilitating and optimizing its deployment in scenarios featuring low cost devices with stringent requirements (both in terms of processing power as well as wireless networking capabilities), which are currently increasing in availability (e.g., sensors in mobile phones and smart environments), without hindering flexible interactions with high-level services (e.g., through L3 transport protocols and broad support of information definition).

Acknowledgements

This work bas been partially funded by the European Community's Seventh Framework Programme (FP7-ICT-2009-5) under grant agreement no. 258053 (MEDIEVAL project) and by the Portuguese Innovation Agency / National Strategic Reference Framework (AdI/QREN) under grant agreement no. 2011/021580 (APOLLO project).

References

[1] Cisco visual networking index: Global mobile data traffic forecast update, `http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.pdf/`, 2011.

[2] IEEE, IEEE Standard for Local and Metropolitan Area Networks- Part 21: Media Independent Handover, IEEE Std 802.21-2008 (2009).

[3] J. Lertlakkhanakul, S. Hwang, J. Choi, Developing spatial information framework for urban computing environment: A socio-spatial computing framework for smart urban environment, in: Management and Service Science, 2009. MASS '09. International Conference on, pp. 1 –5.

[4] E. Aarts, Ambient intelligence: a multimedia perspective, Multimedia, IEEE 11 (2004) 12 – 19.

[5] G. Kortuem, F. Kawsar, D. Fitton, V. Sundramoorthy, Smart objects as building blocks for the Internet of things, Internet Computing, IEEE 14 (2010) 44 –51.

[6] S. Karnouskos, V. Villasen andor Herrera, M. Haroon, M. Handte, P. Marro andn, Requirement considerations for ubiquitous integration of cooperating objects, in: New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on, pp. 1 –5.

[7] W. Masri, Z. Mammeri, Middleware for wireless sensor networks: A comparative analysis, in: Network and Parallel Computing Workshops, 2007. NPC Workshops. IFIP International Conference on, pp. 349 –356.

[8] S. González-Valenzuela, M. Chen, V. C. Leung, Programmable middleware for wireless sensor networks applications using mobile agents, Mob. Netw. Appl. 15 (2010) 853–865.

[9] Y. We, D. Bein, S. Phoha, Middleware for heterogeneous sensor networks in urban scenarios, in: Information Technology: New Generations (ITNG), 2010 Seventh International Conference on, pp. 654 –659.

[10] C. Thiede, C. Tominski, H. Schumann, Service-oriented information visualization for smart environments, in: Proceedings of the 2009 13th International Conference Information Visualisation, IEEE Computer Society, Washington, DC, USA, 2009, pp. 227–234.

[11] ITEA EU project SODA, `http://www.soda-itea.org///`, 2011.

[12] L. M. S. de Souza, P. Spiess, D. Guinard, M. Khler, S. Karnouskos, D. Savio, Socrades: A web service based shop floor integration infrastructure., in: C. Floerkemeier, M. Langheinrich, E. Fleisch, F. Mattern, S. E. Sarma (Eds.), IOT, volume 4952 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 50–67.

[13] H. Abangar, M. Ghader, A. Gluhak, R. Tafazolli, Improving the performance of web services in wireless sensor networks, in: Future Network and Mobile Summit, 2010, pp. 1 –8.

[14] T. Luckenbach, P. Gober, S. Arbanowski, A. Kotsopoulos, K. Kim, TinyREST: A protocol for integrating sensor networks into the internet, Citeseer.

[15] A. Castellani, N. Bui, P. Casari, M. Rossi, Z. Shelby, M. Zorzi, Architecture and protocols for the internet of things: A case study, in: Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on, pp. 678 –683.

[16] D. Yazar, A. Dunkels, Efficient application integration in ip-based sensor networks, in: Proceedings of the First ACM Workshop on Embedded

Sensing Systems for Energy-Efficiency in Buildings, BuildSys '09, ACM, New York, NY, USA, 2009, pp. 43–48.

[17] A. Castellani, M. Gheda, N. Bui, M. Rossi, M. Zorzi, Web services for the internet of things through coap and exi, in: Communications Workshops (ICC), 2011 IEEE International Conference on, pp. 1 –6.

[18] S. Hong, D. Kim, M. Ha, S. Bae, S. J. Park, W. Jung, J.-E. Kim, SNAIL: an IP-based wireless sensor network approach to the Internet of Things, Wireless Communications, IEEE 17 (2010) 34 –42.

[19] M. Chen, S. Mao, Y. Xiao, M. Li, V. C. M. Leung, IPSA - a novel architecture design for integrating ip and sensor networks, Int. J. Sen. Netw. 5 (2009) 48–57.

[20] D. Corujo, M. Lebre, D. Gomes, R. Aguiar, A framework for flexible sensor information dissemination, in: Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on, pp. 1 –6.

[21] D. Corujo, C. Guimarães, B. Santos, R. Aguiar, Using an open-source IEEE 802.21 implementation for network-based localized mobility management, Communications Magazine, IEEE 49 (2011) 114 –123.