This document describes the second version of the Morphing Mediation Gateway (architecture, concepts) and its modules developed by the Wise-IoT consortium.
Editor: Martin Bauer (NEC)

<table>
<thead>
<tr>
<th>Deliverable nature:</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dissemination level:</td>
<td>PU</td>
</tr>
<tr>
<td>Contractual/actual delivery date:</td>
<td>M14</td>
</tr>
</tbody>
</table>

**Disclaimer**

This document contains material, which is the copyright of certain Wise-IoT consortium parties, and may not be reproduced or copied without permission.

In case of Public (PU):

All Wise-IoT consortium parties have agreed to full publication of this document.

Neither the Wise-IoT consortium as a whole, nor a certain part of the Wise-IoT consortium, warrant that the information contained in this document is capable of use, nor that use of the information is free from risk, accepting no liability for loss or damage suffered by any person using this information.

This project has received funding from the European Union’s H2020 Programme for research, technological development and demonstration under grant agreement No 723156, the Swiss State Secretariat for Education, Research and Innovation (SERI) and the South-Korean Institute for Information & Communications Technology Promotion (IITP).

**Copyright notice**

© 2017 Participants in project WISE-IOT
# Table of contents

- Executive summary .......................................................... 4
- Authors ............................................................................... 5
- Glossary ............................................................................. 6
- 1 Introduction ...................................................................... 7
- 2 The Morphing Mediation Gateway concept ...................... 9
- 3 Morphing Mediation Gateway R2 ................................. 12
  - 3.1 Architecture ............................................................... 12
  - 3.2 Morphing Mediation Gateway Manager ................... 12
  - 3.3 MMG modules ............................................................ 14
    - 3.3.1 Adaptive Semantic Module ................................. 14
    - 3.3.2 Context-Aware Auxiliary Gateway .................... 19
    - 3.3.3 Z-Wave-oneM2M ................................................. 20
    - 3.3.4 OCF-oneM2M ..................................................... 21
    - 3.3.5 LoRa-oneM2M ..................................................... 22
    - 3.3.6 GS1-oneM2M ....................................................... 23
    - 3.3.7 sensInact-oneM2M ............................................. 24
    - 3.3.8 Insator-oneM2M .................................................. 25
- 4 Conclusions ..................................................................... 27
- 5 References ....................................................................... 28
Executive summary

This document describes the second version of the Morphing Mediation Gateway – architecture, concepts and modules – developed by the Wise-IoT consortium.

The Morphing Mediation Gateway (MMG) is the key Wise-IoT component for enabling the interworking between heterogeneous IoT platforms on the one hand and between IoT device and communication technologies and IoT platforms on the other hand. Conceptually, the MMG is part of the Integration and Management Layer in the Wise-IoT Layered Architecture View. Since the oneM2M platform was chosen for instantiating the Integration and Management Layer, the MMG is used for enabling the interworking with specific device & communication technologies, in particular Z-Wave and LoRa, and also for interworking with local or domain-specific platforms, in particular OCF, GS1 and sensiNact. Since the FIWARE platform was chosen for instantiating the Information Access Layer in the Wise-IoT Layered Architecture View, a special focus is on the translation between oneM2M and the NGSI context interfaces supported by FIWARE, which are based on the OMA NGSI standard.

In the MMG approach, the MMG is the driving component behind the translation, decoupling the platforms or technologies. The reason for choosing this approach is that the interfaces of involved platforms and technologies are conceptually so different that a direct integration becomes infeasible. For example, taking the Mca interface (M2M communications application) of oneM2M and the FIWARE GE with their NGSI interfaces as examples, the Mca allows REST operations on a certain set of generic resource types, whereas the NGSI interface allows the retrieval of entities and their attributes, i.e. the former is resource-centric, whereas the latter is information-centric and a direct translation is not possible. Instead information is retrieved from the source system and, where possible, translated to the target system. Thus applications can use the respective operations of the systems they interact with to handle and manipulate the information.

Semantics plays an important role in the MMG approach. Even if the representations on different platforms are very heterogeneous, the underlying semantic concepts have to be the same, thus enabling the translation of information from one representation into the other. This deliverable focuses on the architectural aspects of the MMG, whereas the semantic aspects are presented in Wise-IoT Deliverable D2.5 [2].

The MMG has an MMG Manager that allows the configuration and deployment of MMG modules. MMG modules typically support one type of source platform or technology and one type of target platform. The respective source and target platform instance to be used is then configured through the MMG Manager. MMG modules are implemented as Docker images that can be instantiated at runtime as Docker containers.

Depending on the platforms or technologies between which the translation is supposed to happen, the MMG modules can be relatively simple or highly complex and may even have more fine-grained adaptation capabilities. In particular the Adaptive Semantic Module (ASM) can be used to discover new types of information, to check whether suitable translation components are available or can be downloaded from a repository, and to dynamically instantiate the translation component, enabling the automatic translation.
## Authors

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Author (Organisation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Introduction</td>
<td>JaeSeung Song (SJU)</td>
</tr>
<tr>
<td>2 The Morphing Mediation Gateway Concept</td>
<td>Martin Bauer (NEC)</td>
</tr>
<tr>
<td>3 Morphing Mediation Gateway R2</td>
<td>Martin Bauer (NEC)</td>
</tr>
<tr>
<td>3.1 Architecture</td>
<td>Martin Bauer (NEC)</td>
</tr>
<tr>
<td>3.2 Morphing Mediation Gateway Manager</td>
<td>JongGwan An, JaeSeung Song (SJU)</td>
</tr>
<tr>
<td>3.3 Docker Modules</td>
<td>Martin Bauer (NEC)</td>
</tr>
<tr>
<td>3.3.1 Adaptive Semantic Module</td>
<td>Benjamin Hebgen (NEC)</td>
</tr>
<tr>
<td>3.3.2 Context-Aware Auxiliary Gateway</td>
<td>JaeYoung Hwang (SJU)</td>
</tr>
<tr>
<td>3.3.3 Z-Wave oneM2M</td>
<td>Haris Aftab (SJU)</td>
</tr>
<tr>
<td>3.3.4 OCF-oneM2M</td>
<td>SeungMyeong Jeong (KETI)</td>
</tr>
<tr>
<td>3.3.5 LoRa-oneM2M</td>
<td>SeungMyeong Jeong (KETI)</td>
</tr>
<tr>
<td>3.3.6 GS1-oneM2M</td>
<td>Yalew Kidane (KAIST)</td>
</tr>
<tr>
<td>3.3.7 sensiNact-oneM2M</td>
<td>Rémi Druilhe (CEA)</td>
</tr>
<tr>
<td>3.3.8 Insator-oneM2M</td>
<td>Hyokeun Choi (SDS)</td>
</tr>
<tr>
<td>4 Conclusion</td>
<td>Martin Bauer (NEC)</td>
</tr>
</tbody>
</table>
# Glossary

<table>
<thead>
<tr>
<th>Term or Abbreviation</th>
<th>Definition (Source)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASM</td>
<td>Adaptive Semantic Module</td>
</tr>
<tr>
<td>CAG</td>
<td>Context-Aware Auxiliary Gateway</td>
</tr>
<tr>
<td>CSE</td>
<td>Common Services Entity (oneM2M)</td>
</tr>
<tr>
<td>EPCIS</td>
<td>Electronic Product Code Information Services (GS1)</td>
</tr>
<tr>
<td>HTTP</td>
<td>HyperText Transfer Protocol</td>
</tr>
<tr>
<td>IN</td>
<td>Infrastructure Node (oneM2M)</td>
</tr>
<tr>
<td>IPE</td>
<td>Interworking Proxy Entity (oneM2M)</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>LoRa</td>
<td>Long Range</td>
</tr>
<tr>
<td>MQTT</td>
<td>Message Queue Telemetry Transport</td>
</tr>
<tr>
<td>MMG</td>
<td>Morphing Mediation Gateway</td>
</tr>
<tr>
<td>MN</td>
<td>Middle Node (oneM2M)</td>
</tr>
<tr>
<td>NGSI</td>
<td>Next Generation Service Interface</td>
</tr>
<tr>
<td>OMA</td>
<td>Open Mobile Alliance</td>
</tr>
<tr>
<td>OCF</td>
<td>Open Connectivity Foundation</td>
</tr>
<tr>
<td>OSGi</td>
<td>Open Services Gateway initiative</td>
</tr>
</tbody>
</table>
# Introduction

Many standardized IoT protocols (e.g., HTTP, CoAP, MQTT) and APIs (e.g., oneM2M, NGSI) are used for IoT devices, gateways and servers. By definition, IoT is a technology interconnecting various devices via the Internet to exchange messages for better services. This means that IoT entities (implemented using different protocols) should communicate with each other. However, as each protocol and API defines its own message format, semantics and modelling mechanism, it is not easy to support interoperability.

The objective of the Wise-IoT project is to propose a framework to facilitate the interoperability between various IoT platforms and protocols currently used in Europe and in Korea and offer portability through an information access layer interconnecting different platforms. However, it is unrealistic to completely uniformize the data and semantics model for each of the underlying platforms, thus this framework needs a component that enables building bridges between those models, in particular towards the information access layer and the agreed information model.

The Morphing Mediation Gateway (MMG) is the entity for enabling the interworking between heterogeneous IoT platforms on the one hand and between IoT device and communication technologies and IoT platforms on the other hand. This deliverable introduces the architectural aspects of the MMG, whereas Wise-IoT Deliverable D2.5 [2] introduces the semantic aspects of the MMG and related components, in particular it describes how semantics is used to achieve interoperability between heterogeneous IoT standards.

In release 1, various MMG components were designed and implemented. This shows the ability of Wise-IoT to provide a flexible and dynamic framework interconnecting various IoT entities. However, most Rel-1 MMG components are mainly focusing on generalization of message translation and are not fully integrated into a single framework. In addition, the interface between MMG components and MMG manager (an umbrella MMG component managing MMG components) is only designed for very limited functionalities, e.g., show the number of managing devices.

The overall goal is to provide an integrated, flexible MMG solution, in which different translation modules developed by different partners can be dynamically instantiated at runtime. As for the first release the underlying mechanisms needed are not in place yet, different standalone components were developed by partners.

The MMG components we have implemented so far can be categorized into one of the following gateway entities:

- **Fixed Configuration Gateway**: Such a gateway explicitly maps information/resource instances from source to target system. This approach works well for small set of static sources. New sources require updated configurations, new source types may require installation of new translation modules. → FIWARE-oneM2M-Proxy MMG, Z-Wave-oneM2M MMG, GS1-oneM2M MMG, etc.

- **Discovery-based Gateway**: Such a gateway regularly discovers (subscription/polling) relevant sources and adapts resource mappings according to configuration. This works well for a changing set of sources of fixed type. Changes of relevant source types require updated configuration, i.e. discovery specification – a new source type may also require installation of new translation module.

- **Semantic Mediation Gateway**: A Discovery-based Gateway using semantics for describing information / resource instances and semantic discovery.
- **Morphing Mediation Gateway**: A Discovery-based Gateway with a discovery functionality that discovers a larger variety of candidate information / resource instances. Discovering new types of information / resources may then trigger the dynamic download, instantiation and configuration of new translation modules given that such modules exist. Through this mechanism, new information and suitable translation modules may be provided at any time and the Morphing Mediation Gateway can adapt itself (“morph”) at runtime to handle the translation.

In the second release, all MMG components are transformed into MMG modules that can be flexibly and dynamically deployed in MMG instances.

The remainder of the document is structured into two main parts. The concept of the MMG Release 2 is introduced with an introduction to the MMG manager. The second part is devoted to address all MMG modules developed by Wise-IoT partners. The full functionalities of each MMG module and the Manager are provided.
2 The Morphing Mediation Gateway concept

The Morphing Mediation Gateway (MMG) is the entity in charge of translating, at runtime, a representation of information from one platform or technology to another (cf. Figure 1).

![Abstract architecture of the MMG](image1)

The MMG decouples the source platform / technology and the target platform. The reason for choosing this approach is the high diversity of the interfaces, making a direct translation between the requests for accessing information impossible. Taking the oneM2M platform with its Mca interface (the interface between oneM2M applications and oneM2M platform) and the FIWARE GEs with their NGSI interfaces as examples (cf. Figure 2), the Mca allows REST operations on a certain set of generic resource types, whereas the NGSI interface allows the retrieval of entities and their attributes, i.e. the former is resource-centric, whereas the latter is information-centric and a direct translation is not possible. Instead information is retrieved from the source system and, where possible, translated to the target system. Thus applications can use the respective operations of the systems they interact with to handle and manipulate the information. oneM2M applications use the Mca interface to discover and retrieve/manipulate resource content, whereas NGSI-applications request entities, identifying attributes of interest.

![Example of MMG instantiation](image2)

In the first release, partners had provided an initial set of standalone gateways to be able to start supporting use cases. In the second release, the standalone gateways are replaced by MMGs on which different modules can be dynamically instantiated at runtime. The term “morphing” indicates that the functionality of the gateway can be dynamically changed at runtime, adapting to the changing
environment of the different deployment scenarios. For example, sensors of a new type may become available or existing sensors may be replaced using a different underlying technology. The MMG enables the handling of these changes without disruptions.

<table>
<thead>
<tr>
<th>Application Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knowledge Processing Layer</td>
</tr>
<tr>
<td>Self-Adaptive Recommender</td>
</tr>
<tr>
<td>Integration and Management Layer</td>
</tr>
<tr>
<td>oneM2M: Mobilis</td>
</tr>
<tr>
<td>oneM2M: Insofis</td>
</tr>
<tr>
<td>oneM2M: ThingPlug</td>
</tr>
<tr>
<td>Data Collection and Device Actuation Layer</td>
</tr>
<tr>
<td>GS1: Oliot</td>
</tr>
<tr>
<td>sensiNact</td>
</tr>
<tr>
<td>OCF</td>
</tr>
<tr>
<td>LoRa</td>
</tr>
<tr>
<td>Z-Wave</td>
</tr>
</tbody>
</table>

Figure 3: MMG in the Wise-IoT Layered Architecture View

Figure 3 shows the MMG in the Wise-IoT Layered Architecture View [1]. Conceptually, it is part of the Integration and Management Layer and it is used for two purposes. On the one hand, MMGs are used for translating from different technologies in the Data Collection and Device Actuation Layer to a common IoT platform in the Integration and Management Layer. In case of Wise-IoT the standardized oneM2M platform was chosen for this purpose. On the other hand, MMGs are used for translating between the Integration and Management Layer platform and the Information Access Layer platform, which in Wise-IoT is implemented using the FIWARE platform using FIWARE Generic Enablers (GEs) based on the standardized OMA NGSI context interfaces.

Overall, the focus of Wise-IoT is to make information available to applications. Especially on the information access layer, applications can specify the information they are interested in using a standardized interface. Also on the other layers the focus is on making the IoT-related information available to the oneM2M and FIWARE platforms respectively. The MMG is the component that enables this translation.

In contrast, some standards and projects take a service-centric approach. In particular, the W3C Web-of-Things group [6] standardizes a Thing Description [7]. The Thing Description describes the Thing (in the sense of an IoT device, not an abstract entity like a room or a table), in particular the interface, how applications can interact with it. Thus, in order to interact with the Thing, IoT applications need to understand the respective Thing interface – and there could be a large number of heterogeneous interfaces. The focus is on how Things and related services are described, not on how information is modelled, which could be highly heterogeneous.

The goal in Wise-IoT is to provide applications with a single point of access, ideally on the abstraction level of the Information Access Layer, where information can be directly accessed. Another approach is to provide a central point for discovering services or information and then accessing these directly in a second step. The Big IoT project [8] develops a Market Place where registered platforms and services can be discovered using a unified Web API [9].
The symbIoTe project [10] takes the approach of providing core services and enable platforms to plug into these. In order to do so, platforms have to implement key functionalities and there are multiple compliance levels, identifying how and on what level platforms interoperate. Level 1 is advertising and making resources available, level 2 enabling the federation of platforms, level 3 the simple integration and dynamic reconfiguration of IoT devices in local spaces and finally, level 4 that enables device roaming between local spaces. In Wise-IoT, the goal is to keep standardized and deployed platforms as they are and simply make the information from one platform or technology available in another. This means existing platforms do not have to be modified for compliance, but instead additional MMG modules supporting the translation have to be made available.

The INTER-IoT [11] approach seems to be closest to the MMG approach of Wise-IoT as it provides building blocks and a methodology to connect different IoT platforms at different layers. This indicates that some kinds of mediation gateways (in Wise-IoT terminology) are envisioned. However, there is no discussion about supporting an MMG adaptation at runtime as in Wise-IoT (by dynamically downloading and instantiating new modules) or dynamically discovering new kinds of information and self-adapting to enable automatic translation as in the case of the Adaptive Semantic Module (ASM) for the MMG.

Overall, the goal of Wise-IoT is to use standardized platforms – oneM2M on the Integration and Management Layer and NGSI-based FIWARE GEs on the Information Access Layer. Instances of these platforms are already part of existing deployments, e.g. oneM2M in Busan and FIWARE in Santander. The MMG allows connecting these platforms without having to interfere with the currently deployed platforms, simply adding a new component that interacts with the platforms. In the same way, the MMG allows connecting new technologies to the existing platform deployments. The approach is information-centric, i.e. the focus is on translating information from a source platform or technology to make it available on a target platform. To enable this, a semantic approach is taken, i.e. even if the representations on different platforms are very heterogeneous, the underlying modelled concepts are the same and this enables the translation. The semantic aspects are presented in Wise-IoT Deliverable D2.5 [2]. The focus of this deliverable is on the architectural aspects of the MMG. In Release 2 it consists of a management part that can deploy different modules at runtime. Each module is responsible for the translation between a source platform or technology and a target platform. Chapter 3 introduces the design and implementation of the MMG and its modules.
This section defines the Morphing Mediation Gateway components that have been developed in the second release of the MMG.

### 3.1 Architecture

The Morphing Mediation Gateway consists of an MMG Manager that can dynamically deploy MMG modules as shown in Figure 4. The MMG modules are provided as Docker [5] images that are instantiated by the MMG Manager – as Docker containers. The Docker images are provided in a repository. By adding a new Docker image to the repository, e.g. for supporting a new translation between platforms or technologies, it can immediately be instantiated by an MMG Manager, dynamically enabling the translation in a running system.

In Section 3.2 the Morphing Mediation Gateway Manager is introduced. The MMG Modules, as shown in Figure 4, are then described in subsections of Section 3.3.

### 3.2 Morphing Mediation Gateway Manager

The Morphing Mediation Gateway Manager is a containerized management software component to manage various MMG modules. The MMG Manager allows users to select and instantiate required MMG modules. For example, a user who wants to operate Z-Wave devices and OCF field devices in his/her room using a oneM2M IoT platform, he/she can instantiate Z-Wave-oneM2M and OCF-oneM2M MMG modules using the MMG manager. The MMG manager provides the same interface for each MMG module. As each MMG module is containerized using docker, it can be dynamically
instantiated based on the needs from the user. The basic information to configure each docker MMG module is communicated using REST operations (e.g. HTTP Get, Post). If a user wants to introduce a new MMG docker module to the MMG manager, the user needs to develop the module based on the following interface, MI interface, which enables two main features:

- **Configuration part**: when the MMG Manager instantiates a new MMG docker container from the repository, it has to provide basic information to configure the MMG module. (e.g., IP address and port numbers of source and target IoT entities, data information, basic configuration information) Optionally, the MMG manager can provide a query or selection criteria for devices that need to be monitored by the MMG module.
- **Information part**: This part is used to exchange the current status of an MMG module running in the MMG manager. Using this interface, the user can select devices to control, get the number of managing devices and check running time.

As shown in Figure 5, each docker MMG module has to support two functions, control and data parts. The control part provides an interaction with the MMG manager to manage various control related information such as the number of connected devices, while the data part is to convert data from the source platform to the destination platform.

Currently the repository containing various IoT MMG modules is located in the MMG manager’s local environment. However, the repository can be located at any location. Therefore, if the user intends to add a new MMG module to the repository and wants to run it in his/her MMG manager, he/she must provide the IP address of the repository to download MMG modules.

The user interface of the MMG Manager (see Figure 6) provides the following functions:

1. Displays a list of available source types, e.g. NGSI context model, oneM2M ;
2. Displays a list of available MMG container images from the repository ;
3. Displays a list of available target types, e.g., oneM2M, NGSI context model ;
4. Gets user input to specify sources to be converted into the targeted system, e.g., sensors in Santander, actuators in Busan.
5. Shows the system resource consumption of each docker module. (CPU usage, Memory usage)
3.3 MMG modules

The MMG modules have been implemented as Docker images that can be dynamically instantiated as Docker containers by the MMG Manager. In the following the MMG modules available in Release 2 of the MMG are described.

3.3.1 Adaptive Semantic Module

3.3.1.1 Motivation

The Morphing Mediation Gateway concept as such enables the dynamic deployment of complete modules. Modules are, in general, relatively coarse-grained units providing the translation from a source to a target system and their deployment is generally triggered by an administrator. To enable self-adaptation on a more fine-grained level, the Adaptive Semantic Module (ASM) has been designed. The idea is that for given source and target platforms, a running ASM can adapt itself when new types of information become available.

A configured ASM can discover new sources of information and check whether suitable ASM subcomponents for translation have already been instantiated. If this is the case, they can be re-used. Otherwise, the ASM can check whether fitting components are available in the Component Repository and, if this is the case, dynamically instantiate them.

If a new service provider wants to make information available in the source platform of the ASM and make sure they also become available in the target platform of the ASM, it needs to make sure that a fitting translation component is made available in the Component Repository and that the new information can be discovered by the ASM. If these conditions are given, new types of information can be made available dynamically in the ASM target platform without re-starting the MMG and the ASM.
While the ASM provides a flexible framework that can be applied to different source and target systems, its primary application area in Wise-IoT is the translation from a oneM2M source platform to an NGSI-based FIWARE platform.

To enable this translation, semantic annotations in the oneM2M system are being used. Together with the actual information, e.g. sensor values, the semantic annotation needs to provide the basis for creating the representation according to the NGSI data model used in FIWARE, which models the world as entities that have attributes with values and metadata. For example, the entity may be a bus with an identifier Bus123, which is of type Bus. The bus has an attribute speed whose value is measured by a sensor and the unit of measurement (km/h) is part of the metadata. In the oneM2M the actual speed value may be stored in contentInstance resources in a container resource and all the additional information(entity id, entity type, attribute name, data type, unit) may be provided as a semantic annotation that is stored in a semantic descriptor resource attached to the container resource. Instead of directly providing all information in the semantic descriptor, it is sufficient to provide enough information to be able to derive missing information. For this purpose additional information is used that can be made available using a Context Provider, which may connect to external systems.

The following subsections describe the architectural framework, the individual components and the internal data format of the ASM. More details about how semantics is used in the translation from oneM2M to FIWARE can be found in the Wise-IoT Deliverable D2.5 [2].

3.3.1.2  Adaptive Semantic Module architecture

The ASM consists of multiple components. The modular design shown in Figure 7 and an intermediate Internal Data Format allow for an easy extension of multiple source or target systems. The separation of API interaction (Source Adapter and Target Adapter) and content translation (Input Translator and Output Translator) allows the system to efficiently create translation chains between multiple source and target systems. The ASM uses OSGi to enable the modular design as well as to configure and instantiate multiple instance of specific implementations of the core components. The core components are the following.
Source Adapter

An implementation of the Source Adapter connects to an external IoT system and provides the raw data from the IoT System into the ASM. In order to be linked together with Input Translators a Source Adapter will publish what type of IoT system it is capable of connecting to and providing data from. A Source Adapter can request multiple Input Translators during runtime, allowing it to adapt to specific data formats it might discover from its source system. Additionally a Source Adapter can provide hints towards the Input Translators which might be required for the translation process.

Input Translator

An Input Translator takes the raw data from a Source Adapter and translates it into an intermediate Internal Data Format (see Section 3.3.1.3). If the Source Adapter is missing information, which is needed to translate the raw data into the Internal Data Format, the Input Translator can request one or multiple Context Providers to enable the translation process. Hints provided by the Source Adapter can be forwarded to the Context Provider. An Input Translator publishes which kind of Source Adapter it supports.

Context Provider

A Context Provider is used to support the translation process. Its main goal is to provide additional information in case a source IoT system is missing relevant information for the translation, e.g. type of the data is missing. Since this additional information is specific to the source IoT system Input Translators have to be aware of the specific additional information given by a Context Provider. The Adaptive Semantic Module provides a set of constants to support the exchange between Context Providers and Input Translators.
Output Translator

An Output Translator will translate from the Internal Data Format to the required format of a specific IoT target system. Using the intermediate Internal Data Format no additional information from a Context Provider is required. Hence an Output Translator implementation can straight forward translate into the required raw format of the target IoT system.

Target Adapter

The task of a Target Adapter is to communicate with the target IoT system API and publish the raw data coming from a matching Output Translator into the target system.

Mediation Manager

The main task of the Mediation Manager is to create and disassemble translation chains. In order to achieve this, the Mediation Manager uses the OSGi service discovery to be aware of all Translators, Context Providers and Adaptors present in the ASM. On request it will create and configure new instances of the required component to create a translation chain during runtime. These requests can either come from the External Management Interface, from already instantiated Source Adapters which require new Input Translators or from Input Translators which require a new Context Provider. If a required component with a specific configuration is already instantiated it will be reused to save resources. In the case that a required component is not available within the OSGi container the Mediation Manager will look up the component in a remote component repository. If a component is not available to build a requested translation chain the Mediation Manager will store the request and fulfill it when all required components are available.

External Management Interface

The External Management Interface provides a REST interface to request new translations from a specific source IoT system instance to a specific target IoT system instance. The requests are forwarded to the Mediation Manager which takes care of linking all the required components with each other.

3.3.1.3 Internal Data Format

The Internal Data Format is used as an intermediate format during the translation process. Hence it has to be capable of holding all typically used IoT data formats, in particular the ones encountered in the Wise-IoT use cases. As shown in Figure 8 and Figure 9, a JSON inspired approach was taken for the
Internal Data Format where an instance of the format has a list of Data Values where each value can be the root of an Internal Data Format instance again or a real value. Similar to JSON an object has name and value where value can be object again. This format should be capable of holding all kinds of tree formats as well as relationship based formats.

![Internal Data Format structure](image)

As shown in Figure 8 and Figure 10, each of the Internal Data Format subfields has an actual value like Name or Type and an additional MetaData field or a list of MetaData fields to support contextual information like e.g. data type, unit of the value or timestamp etc. The Adaptive Semantic Module provides a set of constants to be used when filling up the MetaData fields, in order to support the communication between Input Translators and Output Translators.

![Data Identifier, Data Type and MetaData structure](image)

For a seamless interaction between Output Translators and Input Translators there has to be an agreement on where certain MetaData is placed. For instance a timestamp could be connected to the actual value or to the DataIdentifier or a unit type (e.g. cm or kg) could be connected to the DataType or to a specific value. This contract between the Translators might have to be further enforced in future by providing specific setters and getters for common MetaData.
3.3.2 Context-Aware Auxiliary Gateway

Context-Aware Auxiliary Gateway (CAG) is one of the dockerized MMG modules developed for converting IoT information based on resource mapping rules. More specifically, it gets the Entity information from the FIWARE platform (NGSI), translates the information and stores it in the oneM2M platform based on resource mapping rules. For example, if users who would like to get information about their devices on a oneM2M server can also get information about their devices natively registered with a FIWARE server without specific handling. With the resource mapping rule, the data over FIWARE NGSI and oneM2M Mca should be interoperable by the resource mapping rules. The resource mapping rules are not fixed but they can be changed according to the scenarios. In the Wise-IoT project, CAG is in charge of converting data from FIWARE NGSI to oneM2M Mca based on smart parking scenarios of Santander smart city. Since this component is dockerized, the MMG Manager can load and run it from the MMG Repository whenever users would like to use it. Therefore it acts as a gateway between the FIWARE platform and oneM2M as described in Figure 11.

- **Step1**: Receiving FIWARE Entity information from the web-based MMG (Morphing Mediation Gateway) Manager which provides users with pop-up windows for selecting the specific Entity and also CAG has implemented the REST API for receiving the Entity information dynamically.
- **Step2**: CAG gets the FIWARE device information based on Entity Name and Entity Type.
- **Step3**: CAG converts FIWARE Device information to oneM2M standard.
- **Step4**: After all FIWARE device information is converted into oneM2M standard, CAG subscribes FIWARE devices for tracking devices status changes.
- **Step5**: CAG receives notification messages from Context Broker when registered FIWARE device status value is changed.
- **Step6**: Finally, CAG updates the device information stored in oneM2M server.
3.3.3 Z-Wave-oneM2M

Z-Wave-oneM2M is one of the MMG modules used by the Morphing Mediation Gateway (MMG) manager. The essence of this module is to coordinate Z-wave devices (e.g., sensors) with the help of Z-Wave controllers to create oneM2M resources in oneM2M server. The data then can be used to build oneM2M applications for different purposes. The Z-Wave based devices are managed at nodes with the help of Z-Wave controllers. The nodes in general are located in different places and communicate over IP network to Z-Wave-oneM2M.

![Diagram of Z-Wave-oneM2M module interaction with MMG Manager]

Z-Wave-oneM2M interacts with the MMG manager over a set of defined interfaces in order to provide data from different nodes. The MMG manager acts as a dashboard and provides an IP address of the oneM2M based server and other configurations to Z-Wave-oneM2M. The MMG manager is able to show the number of the Z-Wave devices attached to a specific node. The MMG Manager can send instructions to Z-Wave-oneM2M about specific Z-Wave controller in order to add or remove any Z-Wave device.

Z-Wave-oneM2M controls multiple Z-Wave controllers attached to different nodes. A single Z-Wave controller has the capability to manage multiple Z-Wave devices. As new data arrives (e.g., the data can be a temperature value), it is collected at the node by Z-Wave controller and then sent to Z-Wave-oneM2M running inside the MMG Manager. This data is then translated into oneM2M format and written to oneM2M based server in form of oneM2M based resources.
There are vast number of scenarios in which this module can be helpful. For example, when a Z-Wave device captures an unusually high temperature e.g. in case of fire then the situation can be propagated instantly from Z-Wave-oneM2M to oneM2M server. Then an emergency-handling application can identify and locate the situation and take steps to handle it. Since Z-Wave based devices can capture several different attributes such as temperature, humidity, luminance, motion and so on, Z-Wave-oneM2M can be used in variety of different applications and scenarios.

### 3.3.4 OCF-oneM2M

With OCF-oneM2M interworking technical specification [4] implementation, oneM2M applications can control and get data from OCF devices. The OCEAN open source community provides OCF IPE as well as the &Cube oneM2M gateway platform, the Mobius server platform. Figure 14 shows the system composition for OCF devices to be used in oneM2M system with the open sources.
For the interworking, firstly the OCF devices and resources are represented in oneM2M resources and exposed into the gateway platform (i.e. MN-CSE). The other application (e.g. registered to the server platform) can use oneM2M discovery API to search for an OCF device matching the filter criteria in the query.

For control, the application can send control message in <contentInstance> resource create request to the gateway platform. It triggers notification message and is sent to the OCF IPE. The IPE then translates the message into OCF message and send the OCF protocol message to the corresponding OCF device which has OCF server capability.

The OCF IPE observes OCF resources and when it gets notifications for the update, it triggers a new <contentInstance> resource creation that could be consumed by oneM2M application as updated OCF resources.

3.3.5 LoRa-oneM2M

LoRa-oneM2M interworking technology is deployed in WISE-IoT firstly for the smart parking use case. In the oneM2M specification, the IPE (Interworking Proxy Entity) is a type of application entity (AE) which translates two different protocols. The LoRa IPE consists of LoRa G/W interworking S/W and oneM2M AE so it can provide message exchange between LoRa and oneM2M.

Usages of this LoRa IPE is not limited to LoRa parking sensors as depicted in Figure 15, but also can be applied for the other type of LoRa devices (e.g. LoRa trackers).

Figure 15: Smart parking service deployment using LoRa-oneM2M interworking

Figure 16 shows the oneM2M resource structure for the LoRa IPE, which can be generically used not only for parking service but the other services. This structure provides the uplink and downlink message transfer between LoRa G/W and a oneM2M platform. When uplink message is generated by LoRa device, it gets saved in the oneM2M platform (e.g. IN-CSE) as a <contentInstance> resource by LoRa IPE. In the figure below, the newly created uplink message resource path is “mobius/<appEUI>/<deviceEUI>/up/<msg>”. Note that the resource name in pointy bracket (e.g. <appEUI> resource) means the resource instance name is not fixed. In contrast, when the LoRa IPE gets notification from the platform for downlink message, it sends the message to the LoRa G/W and
get forwarded to the corresponding LoRa device. To perform this, the LoRa IPE subscribes each LoRa devices’ downlink message <container> resource.

Two information in a LoRa protocol message are needed for oneM2M system interworking: application EUI (Extended Unique Identifier) and device EUI. Application EUI represents application service (e.g. parking service) identifier that service provider defines. Each appEUI resource contains device EUI <container> resources as children, which represents individual LoRa device having the same service identifier. The LoRa IPE uses this two identifiers in a LoRa message to determine resource path for uplink message resource creation and two identifiers in a oneM2M resource to send LoRa downlink message to the LoRa G/W.

![Figure 16: oneM2M resource structure for LoRa IPE](image)

### 3.3.6 GS1-oneM2M

![Figure 17 GS1-oneM2M MMG Docker Instance](image)
In the WISE-IoT project, GS1-Oliot is used to collect bus information data from Busan city. In order to make the data available on the WISE-IoT platform, more specifically the oneM2M server, the GS1-oneM2M MMG works as a mediator to sync the data between GS1-Oliot and oneM2M.

To push the data from GS1-Oliot to oneM2M, the oneM2M Accessing Application of the MMG uses the EPCIS querying interface to get data from GS1 and publish it to oneM2M using the MQTT binding. Additionally, the bus information is stored in oliot-EPCIS in the form of events and master data (EPCIS data model). Therefore, it is the responsibility of the GS1-oneM2M MMG module to translate the Oliot-EPCIS data model based information into oneM2M data model before publishing to the oneM2M server.

Similarly, information collected by another platform can be collected back to GS1-Oliot through the oneM2M Capturing Application which subscribes to the oneM2M server. It converts the oneM2M based data into EPCIS events and master data before storing it into the EPCIS repository. The position of the MMG in integrating GS1 with WISE-IoT is shown in Figure 17.

The dockerized instance of GS1-oneM2M MMG will be stored in the WISE-IoT docker repository. The MMG can then be used with minimum configuration, which enables easy interactions with the Oliot architecture to collect bus information data.

3.3.7 sensiNact-oneM2M

The sensiNact platform is used for the smart skiing use case that is deployed in Europe. This platform, developed by the CEA, and hosted in the Eclipse Foundation as an open source project, proposes its own agnostic data model. It means that the data model does not depend on the targeted environment where the platform is deployed. sensiNact allows to connect to the physical devices or upper layer platform like oneM2M using bridges, i.e., dedicated components enabling the translation from a data model to another data model.

sensiNact solution is based on the OSGi specification [3], a new bridge can be added or removed at runtime according to the needs of the users or of the environment. Thus, the integration of a new protocol into the platform does not affect the others components already available. sensiNact also limits the development time related to the integration of a new protocol because the developer only focuses on the protocol and the data model.

---

1 https://projects.eclipse.org/projects/technology.sensinact/
In the Wise-IoT project, sensiNact is deployed in Chamrousse, for the smart skiing use case. sensiNact can be deployed in a distributed way. In imply that two deployed instances of sensiNact can be connected using an Eastwest interface. The two instances of sensiNact, one in Chamrousse and one in the backend Wise-IoT platform instance, share the resources but from the user point of view there is only one system. The sensiNact-oneM2M MMG module is only composed of a light version of sensiNact and of the oneM2M MQTT binding. The corresponding architecture is presented in Figure 18.

When the Wise-IoT platform wants to gather data from the sensiNact deployed use case site, the MMG just have to instantiate the sensiNact-oneM2M MMG module. The later connects to the on-site sensiNact and can then push data to oneM2M platform using MQTT binding².

Finally, to be supported by the MMG, the sensiNact-oneM2M MMG module, i.e., the dedicated oneM2M bridge and the core, are packaged in a Docker container and stored into the MMG R2 repository. Thus, it can be instantiated for interconnecting smart skiing devices to the Wise-IoT platform.

### 3.3.8 Insator-oneM2M

Insator is an enterprise IoT platform that supports developing smart solutions and intelligent services providing an IoT data pipeline, analytics capabilities and interoperability with enterprise systems. It covers on premise and cloud environment both targeting enterprise businesses. By supporting various IoT connectivity protocols, Insator applied oneM2M standard certification from July in 2017 and developed OCF standard adaptor for Samsung Electronics devices in last Jan. Furthermore, In order to support customized protocols to connect equipment and devices, Insator also have message schima modeling tool so that developers can leverage it when they need to build their own customized adaptor to connect particular devices.

² [http://onem2m.org/images/files/deliverables/TS-0010-MQTT_protocol_binding-V1_0_1.pdf](http://onem2m.org/images/files/deliverables/TS-0010-MQTT_protocol_binding-V1_0_1.pdf)
In the WISE-IoT project, Insator is deployed in Alpensia, for the skiing resort management use case.

In order to archive Insator-oneM2M interoperability, Insator applied bridge pattern to implement the different kind of standards. The corresponding architecture is presented in Figure 18. Once the front station collect the data from LoRA Gateway(IPE), internal message converter of Insator-oneM2M MMG which is responsible for translating the data format based on IoT standards changes the data for Insator Engine. After changing the data as an internal message format, Data Bus which is a message queue deliver it to Insator Engine so that the oneM2M service store the data into the database. Once the oneM2M comes in Insator as a uplink message, oneM2M proxy (IN-AE) get the message payload to parse the data based on the resource model we created for the resort management application(in-AE). Insator-oneM2M MMG also provide resource CRUD and the subscribe/notify functions for the resort management application.
4 Conclusions

With the Release 2 of the Morphing Mediation Gateway, a powerful and flexible tool for enabling the interworking between different IoT platforms on the one hand, and between different device & communication technologies and IoT platforms has been made available to Wise-IoT. It serves as a basis for implementing the use cases in the different pilot sites and allowing application developers to focus on using their preferred interface of one IoT platform – in Wise-IoT the NGSI interface of FIWARE or the Mca interface of oneM2M.

As information can easily be made available on the oneM2M and FIWARE platforms at the same time using the Morphing Mediation Gateway, applications become portable across different sites regardless of the primary IoT platform originally deployed on one site, e.g. oneM2M in Busan and FIWARE in Santander – always under the assumption that the right kind of information is available. Of course, the requirement for the translation to be feasible is that the underlying general concepts are compatible. Then even significantly different representations can then be translated as needed.

Overall, the Morphing Mediation Gateway plays an important role in fulfilling the objectives of the Wise-IoT project, in particular the connection of exiting platforms, the integration of smart objects, semantics as the basis for interoperability and the achievement of application and service portability in complex and global IoT scenarios.
5 References

[4] TS-0024 release 2, oneM2M and OIC Interworking