Appendix A

Contract Description Language Specification

DISCLAIMER: The work presented in this appendix is under development. It is not final. Some parts are not finished or could change in future versions of the document.

The Contract Description Language (CDL) is an interface description language to specifically describe the service contract of a system. This specification describes the CDL parts and structure, giving different examples. The proposed specification is work in progress. A draft to show the intended version of the CDL.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "OPTIONAL" and "MAY" in this document are to be interpreted as described in BCP 14 RFC2119 RFC8174 when, and only when, they appear in all capitals, as shown here.

Notes:

- It is defined the usage of curly braces ({}) to mark a section of a URL path as replaceable using path parameters.
- Media types are used all over the different elements and definitions. The media type definitions SHOULD be in compliance with RFC6838.

1 Document Structure

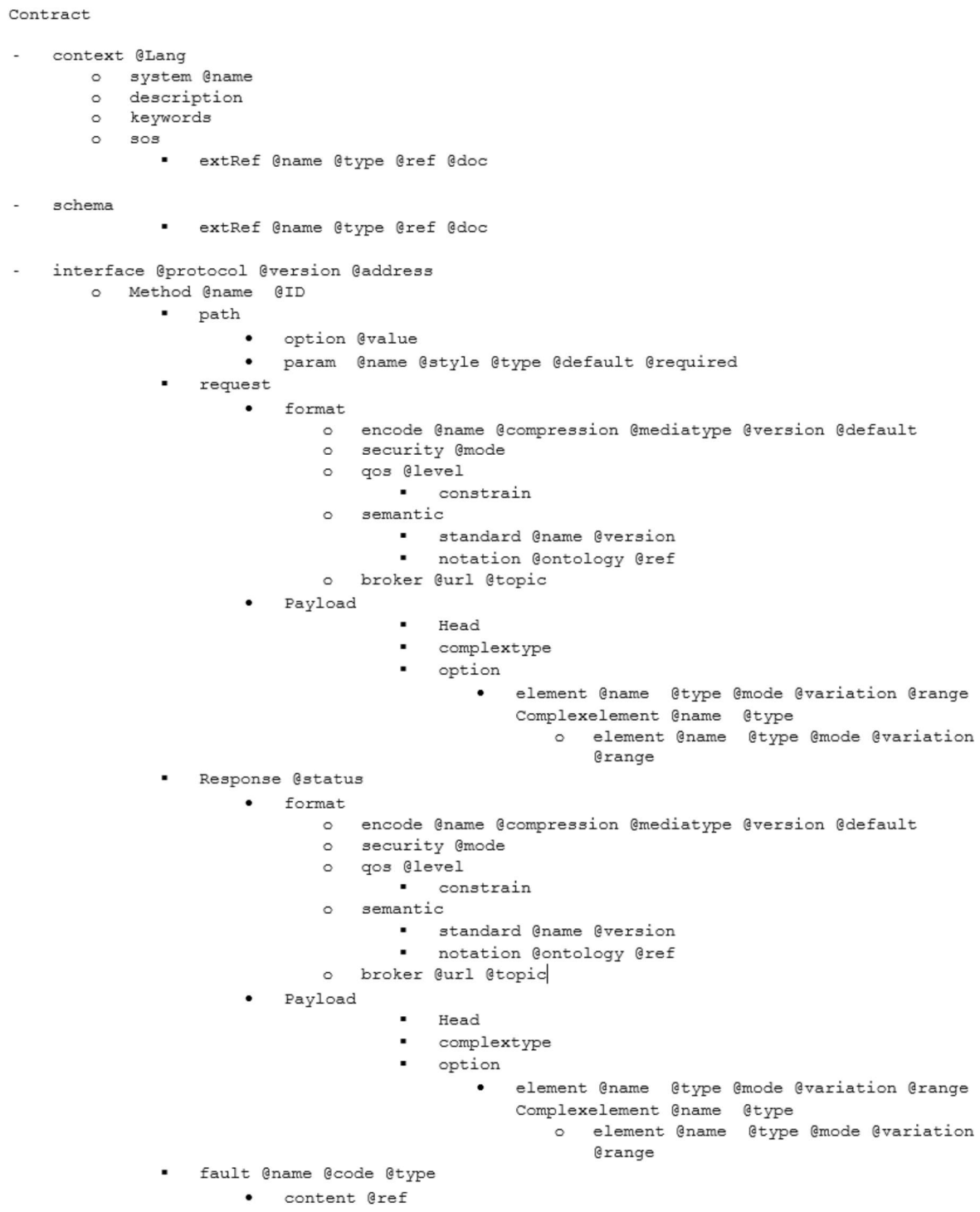

Figure 1: Document structure summary.

2 CDL Format

CDL is an interface description language defined as an XML document. Each element defined in this specification is an XML tag that may have attributes and other elements nested. The specification could be model using other formats, such JSON. Nevertheless, XML is the preferred format and the use of XML schemas as an extension of the main CDL document is recommended.

All field names in the specification are case sensitive. This includes all fields that are used as keys in a map, except where explicitly noted that keys are case insensitive. The schema exposes two types of fields: Fixed fields, which have a declared name, and Patterned fields, which declare a regex pattern for the field name. Patterned fields MUST have unique names within the containing object.

Note: Despite the use of XML as a format for the document, the request and response bodies are not required to be XML.

3 Contract

The *contract* element is the root element of the CDL document. *contract* refers to the Service Contract concept defined in SOA. The *contract* element refers to the namespaces used during the definition of the document. A *contract* element MUST have:

- One *context* element.
- Zero or more *schema* elements.
- One of more *interface* elements.

Each of these elements are defined and explained in the following sections.

```
<?xml version="1.0" encoding="UTF-8"?>
<contract xmlns="http://cdl/2018/01" xmlns:xsd="http://www.w3.org/2001/XMLSchema" />
```

Figure 2: Example of the first two lines of a CDL document.

4 Context

The *context* element provides extra information about the service. The *context* element has one attribute:

- xml:lang - Description of the natural language of the service. Abbreviation of the languages according to the ISO 639-1 Language Codes (e.g, “en” for English).

And the following children elements:

Element	Type	Attributes	Description
system	String	@name	OPTIONAL. Definition of the system name where the service is running.
description	String	–	REQUIRED. Briefly description of the service.
keywords	String	–	OPTIONAL. List of keywords related to the service. Keywords can be used for future compatibility test.
SoS	[extRef]	–	OPTIONAL. List of *extRef* elements that point to other SoS resources that are related to the service.

Table 1: Context children elements summary.

4.1 extRef

The *extRef* element defines an external reference from the document. This reference can be a xml schema, or a link to an external resource or service. It is followed by a brief description. The *extRef* element MUST have the following attributes:

- *name* - Identifier of the external reference. If the same reference is used in other section of the document the reference name MUST be the same (unique external references identifiers).
- *ref* - Reference to the resource.
- *type* - Type of reference. Possible types are url and namespace.
- *doc* - Definition of the external reference.

```
<extRef name="location"
ref="http://plantdescription/location?service=temp"
type="xml:url"
doc="Geographical location of the sensor" />
```

Figure 3: Example of estRef.

5 Schema

The *schema* element defines external references which point to schemas or links. The stated links are used in other sections of the document. The *schema* element makes use of the *extRef* element previously defined.

```
<schema>
<extRef name="" ref="" type="" doc="" />
<extRef name="" ref="" type="" doc="" />
</schema>
```

Figure 4: Example of schema.

6 Interface

In the same way that the service interface specifies the service operations, the element *interface* contains the methods and operations of the service. Each *interface* element has two attributes:

- *protocol* - The protocol attribute (type xs:string) defines the application protocol used in the interface.
- *version* - Protocol version.
- *address* - The address attribute (type xs:anyURI) provides the base URI for each child method.

The *interface* element is followed by one or more method elements.

```
<interface protocol="HTTP" version="1.1"
address="http://192.168.1.40:8888/example" />
```

Figure 5: Example of interface.

7 Method

A *method* element describes an operation or resource. Each method has two attributes:

- *name* (String) - This attribute provides the method definition. The value depends on the protocol used, Table 2.
- *ID* (String) - This attribute provides a identifier. In the case of HTTP and CoAP refers to the service identifier. For MQTT to the client identifier.

In addition to the attributes, the element *method* has the following child elements:

Protocol	Operations
HTTP	Get, Post, Put, Patch, Options, Connect, Delete
CoAP	Get, Post, Put, Delete
MQTT	Connect, Disconnect, Publish, Subscribe, Unsubscribe, Close

Table 2: Operations described for each protocol.

- One *path* element.
- Zero or one *request* element.
- Zero or one *response* element.
- Zero o more *fault* elements.

Elements are explained in the following sections.

8 Path

The *path* element describes the specific path for a resource or operation. A *path* element can have one or more optional values. The element *option* describes the values that the path can take. Paths, alike the urls, can have different parameters, the parameters are defined by the element *param*.

Element	Attributes	Description
option	@value (String)	REQUIRED. Values that can be taken by the path.
param	@name @format @type @default @required	OPTIONAL. Description of a path parameter.

Table 3: Path children elements summary.

Definition of the *param* attributes.

- *name* (String) - REQUIRED. Definition of the parameter name. Parameter name has to be the same than the path parameter, declared between {}.
- *format* (String) - REQUIRED. Style of the parameter (query, single, matrix, header)*.
- *type* (String) - REQUIRED. Data type of the parameter.
- *default* (String) - OPTIONAL. Default value of the parameter.
- *required* (Boolean) - OPTIONAL. The parameter is optional if this flag is set to false and compulsory if it is set to true. The default value is false.

*Valid parameter styles are:

- *query*. Specifies a URI query parameter.
- *single*. Specifies a one single element that is substituted in the URI.
- *header*. Specifies a message header (e.g, http header).
- *matrix*. Specifies a matrix URI Component.

```
<method name="POST" id="create-car" >
<path>
<option value="/car" />
<option value="/car/{carcolor}" />
<param name=”carcolor” style=”single” type=”xs:string” required=true />
</path>
```

Figure 6: Example of path.

9 Request

The *request* element describes the details of the request message sent during the operation. The *request* element has no attributes. It has two child element, format and payload.

10 Format

The *format* element defines the characteristic and details of the service communication.

Element *format* has the following child elements, Table 4; each element is explained in detail in the following subsections.

11 Encode

The *encode* element has the following attributes:

- *name* (String) - REQUIRED. Name of the representation format (eg. JSON, XML, CBOR).
- format (String) - REQUIRED. Style of the parameter (query, single, matrix, header).

Element	Attributes	Body	Description
encode	@name @compression @mediatype @version @default	[option]	REQUIRED. The *encode* element describes the syntactic format used.
security	@mode	–	REQUIRED. Description of the service security.
qos	@level	[constrain]	OPTIONAL. Description of the parameters necessary to monitor the quality of service.
semantic	–	[standard] [notation]	REQUIRED. Semantic information of the message.
broker	@URL @topic	–	OPTIONAL. MQTT broker details. Only used in MQTT communications.

Table 4: Format children element summary.

- *compression* (String) - OPTIONAL. Definition of the compression type, in case of use.
- *mediatype* (String) - OPTIONAL. Media type of the format. Media types have to follow the RFC 6838.
- *version* (String) - OPTIONAL. Version used.
- *default* (Boolean) - OPTIONAL. If the default flag is set to true that encoding is the format by default. This flag is only used if more than one encoding is defined.

Multiple encodings can be defined for the same request. The *option* element is used to this purpose. The *option* element SHOULD make of use the same attributes than the *encode* element.

```
<request>
    <format>
        <encode name="JSON" mediatype="application/json" version="2.3.0" />
    </format>
```

Figure 7: Example of the format element with a encode description.

```
<request>
    <format>
         <encode>
           <option name="JSON" mediatype="application/json" version="2.3.0" />
           <option name="XML" mediatype="application/xml" version="1.0" />
           </encode>
    </format>
```

Figure 8: Example of a format element with optional encodes.

12 Security

The element *security* describes the security mechanism and settings used in the service communication. The *security* element has one attribute:

- *mode* (String) - Defines the level of security used. Valid values are "not secure" and "secure".

If the mode is different to "not secure", the element *security* has the following elements to define the security scheme:

Element	Body	Description
type	string	REQUIRED. Type of security scheme. Valid values are "certificate", "token", "httpbasic".
description	string	OPTIONAL. Description of the security schema.
auth	[authconf]	REQUIRED. Authorization configuration details.

Table 5: Definition of the security mode elements.

12.1 Authconf

The *authconf* element defines the authorization details. The security types considered for this version of the specification are: certificates and tokens.

Attribute	Type	Description
domain	string	REQUIRED. Application domain. Valid values "intracloud" and "intercloud".
scheme	string	OPTIONAL. Name of the HTTP Authorization scheme used in the Authorization header as defined in RFC7235.
identityAuth	string	REQUIRED. URLof the Authorization system used for this service. It MUST be in the form of a URL. (e.g., the URL of the Arrowhead Authorization system).
certificate	string	OPTIONAL. Type of certificate used.
certAuthority	string	OPTIONAL. Certificate Authority URL. It MUST be in the form of a URL.
rootCertificate	string	OPTIONAL. Root certificate required to follow the chain of trust.
password	string	OPTIONAL. Password required accessing the system.

Table 6: Definition of authoconf element attributes.

```
<security mode="secure">
          <type "certificate" />
          <auth>
                    <domain "intracloud" />
                    <certificate "X.509" />
                    <root "ltu.arrowhead.eu" />
          <auth/>
<security/>
```

Figure 9: Example of the security element.

13 QoS

The *qos* element defines the parameters necessary to monitor the quality of service. The main attribute used is the level attribute:

- *level* (Integer) - The attribute level depends on the QoS definition. In the case of MQTT the levels are defined as follows (0 - at most one, 1 - at least one, 2 - exactly one). For other protocols the level attribute can be set to 3, and be defined using the *constrain* element.

The element *constrain* will be include in future versions of this document. It is not

part of the current version of the specifications.

14 Semantic

The *semantic* element describes the standard or ontology used in the message exchange. The *semantic* element MUST have one of the following child elements. The elements are exclusive.

The use of *extRef* to point to an external reference to the standard or notation is OPTIONAL.

Element	Attributes	Body	Description
standard	@name @version	[extRef]	Definition of the semantic standard used.
notation	@ontology @ref	[extRef]	Definition of the ontology used.

Table 7: Semantic element children description.

```
<semantic>
        <standard name="SenMl" version="RFC8428" />
          <extRef name="SenML"
          ref="https://tools.ietf.org/html/rfc8428"
          type="xml:url"/>
</semantic>
```

Figure 10: Example of the semantic element with a standard description.

```
<semantic>
     <ontology name="IoT-Lite" ref=http://www.w3.org/iot-lite-20151126/" />
</semantic>
```

Figure 11: Example of the semantic element with a ontology reference.

15 Broker

A *broker* element defines a MQTT broker, it has two main attributes:

- *URL* (xs:anyurl) - Broker address.
- *topic* String) - MQTT topic.

16 Payload

The *payload* element defines the payload structure and content of the message. *Payload* elements can have the following children:

Element	Attributes	Body	Description
head	–	String	OPTIONAL. Head of the message.
complextype	–	String	DOPTIONAL. It describes the collection data type used (e.g., list, array). The message will be formed by an indeterminate repetition of the following *element* elements.
element	@name @type @mode @variation @range	–	OPTIONAL. Basic entity that describe an element in the payload. It usually refers to a pair of values.
complexelement	@name @type	[element]	OPTIONAL. Element formed by other elements. It has a name attribute and one or more *element* elements.

Table 8: Payload element children description.

16.1 Element

The attributes of this element are:

- *name* (String) - REQUIRED. It refers the key value.
- *type* (String) - REQUIRED. It refers the data type of the value.
- *mode* (String) - OPTIONAL. Describes value mode. Valid modes are optional and persistent.
- *variation* (String) - OPTIONAL. It refers to other names that can be used. The names are not used directly in the payload. They are only used to give more

semantic information, for example, when the payload is limited and abbreviations are used.

- *min* - OPTIONAL. Optional attribute that refers to the minimum value.
- *max* - OPTIONAL. Optional attribute that refers to the maximum value.

16.2 Complexelemnt

The attributes of this element are:

- *name* (String) - REQUIRED. It refers the key value.
- *type* (String) - REQUIRED. It refers the structure type of the elements. If the elements that are part of the complex element are present only once the type MUST be "single". Other options accepted are "List" or any type of collection, if the elements are present multiple times.

```
[
{"car":
{ "brand":"Volvo",
  "color":"red" }},
{"car":
{ "brand":"Audi",
  "color":"blue" }}
]
```

```
<payload>
            <complextype type="List"/>
            <complexelement name="car" type="single">

            <element name="brand" type="String"/>
            <element name="color" type="String"/>
            </complexelement>
</payload>
```

Figure 12: Relation between a payload with an array of compose Json object, with the corresponding definition of the payload element.

17 Response

The element *response* defines the response message sent as an answer for the request in that specific method. It has one attribute.

- *status* - REQUIRED. Defines the response status code expected.

Except for the *status* attribute, the element *response* has the same structure than the *request* element. In case of an empty response, only the *status* attribute is required.

```
{“temp”:25, “u”= “Celsius”}
```

```
<payload>
    <element name="temp" type="Integer" variation=”temperature” min=0 max=40/>
    <element name="u" type="String" variation=”unit”/>
</payload>
```

Figure 13: Relation between a payload with a simple json object that defines a temperature measure, with the corresponding definition of the payload element.

18 Fault

The *fault* element defines the errors that can happened during the invocation of the method. A *fault* element have the following attributes:

- *name* (String) - REQUIRED. Fault name.
- *code* (String) - REQUIRED. Status error code.
- *type* (String) - OPTIONAL. Error type.

And one child element.

- *content* [@ref] (String) - OPTIONAL. Definition of the fault, and reference to the error source.

```
<method name="POST" id="create-car" >
  <path>
          <option value="/car" />
  </path>
  <request>
      <format>
         <encode name=”JSON” mediatype=”application/json version=”2.3.0” />
         <security mode=”secure”>
                     <scheme token />
                     <authconf …
         </security>
         <qos level=3>
                     <constrain -----/>
         </qos>
         <semantic>
                      < notation ontology=”car-custom-AH” ref=”…” />
         </semantic>
    </format>
    <payload>
           <complexelement name=”car type=”single” >
                   <element name="brand" type="String"/>
                   <element name="color" type="String"/>
           </complexelement>
    </payload>
</request>
```

Figure 14: Example of a complete request element.

```
<response status=”200” />
```

Figure 15: Example of an empty response.