A Formal and Tooled Framework for Managing Everything as a Service

www.occiware.org

Deliverable D2.3.1

OCCI Specific Language - Structural Part

OCCIware project is supported by Fonds national pour la Société Numérique (FSN) and the following competitive clusters Systematic, Minalogic, PICOM, Images & Réseaux and Solutions Communicantes Sécurisées.
Abstract
The Open Cloud Computing Interface (OCCI) is a RESTful Protocol and API for all kinds of management tasks. Unfortunately, OCCI designers describe their own OCCI extensions and configurations informally in natural language illustrated by UML diagrams. Thus OCCI lacks of a domain-specific description language to define both OCCI extensions and configurations unambiguously, precisely, and completely. We call it the OCCIware Description Language (OCCIwareDL).

The goal of this deliverable is to precisely define the structural part of OCCIwareDL. More precisely, this deliverable presents the abstract syntax then describes the textual syntax of OCCIwareDL.

OCCIwareDL is a declarative language at the intercession between an Interface Description Language (IDL) and an Architecture Description Language (ADL). The IDL nature of OCCIwareDL is used to precisely define OCCI extensions. The ADL nature of OCCIwareDL is used to precisely define OCCI configurations.
Document versioning

<table>
<thead>
<tr>
<th>Version</th>
<th>Changes</th>
<th>Author(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>Initial version.</td>
<td>Philippe Merle</td>
</tr>
<tr>
<td>0.2</td>
<td>Added BNF and syntax diagrams in Chapter Textual Syntax. Added appendix on OCCIware BNF. Added first draft of Abstract. Added first draft of Chapter Introduction. Added first draft of Chapter Abstract Syntax. Updated rules <code>&lt;ImportDecl&gt;</code>, <code>&lt;KindDecl&gt;</code>, <code>&lt;MixinDecl&gt;</code> and <code>&lt;ActionDecl&gt;</code> as suggested by Vincent Zurczak from Linagora. Wrote Chapter Textual Notation. Added <code>use</code> extensions in <code>&lt;ConfigurationDecl&gt;</code> Revisited <code>&lt;LinkDecl&gt;</code>: added ‘target’ and removed <code>&lt;LinkDecl&gt;</code> Wrote Chapter Conclusion.</td>
<td>Philippe Merle</td>
</tr>
<tr>
<td>0.3</td>
<td>Replaced EEnumType by EEnum. Updated rule PositiveInteger: replaced <code>[0-9]*</code> by <code>[0-9]+</code></td>
<td>Philippe Merle</td>
</tr>
<tr>
<td>0.4</td>
<td>Used new OCCIware template.</td>
<td>Philippe Merle</td>
</tr>
<tr>
<td>0.5</td>
<td>Add XML schema and syntax description.</td>
<td>Jean Parpaillon</td>
</tr>
<tr>
<td>1.0</td>
<td>Final version.</td>
<td>Jean Parpaillon</td>
</tr>
<tr>
<td>1.1</td>
<td>Corrected typos, removed graphical notation and updated final version.</td>
<td>Philippe Merle</td>
</tr>
</tbody>
</table>

Document reviews

<table>
<thead>
<tr>
<th>Review</th>
<th>Date</th>
<th>Ver.</th>
<th>Reviewer(s)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plan</td>
<td>May 6th 2015</td>
<td>0.1</td>
<td>Philippe Merle</td>
<td></td>
</tr>
<tr>
<td>Brouillon</td>
<td>July 1st 2015</td>
<td>0.5</td>
<td>Jean Parpaillon</td>
<td></td>
</tr>
<tr>
<td>Validation</td>
<td>July 23rd</td>
<td>0.5</td>
<td>Olivier Barais</td>
<td></td>
</tr>
<tr>
<td>BAT</td>
<td>July 24th</td>
<td>1.0</td>
<td>Jean Parpaillon</td>
<td></td>
</tr>
<tr>
<td>Deliverable</td>
<td>D2.3.1</td>
<td>Status</td>
<td>Final</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>--------</td>
<td>---------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>Expected</td>
<td>May 31, 2015</td>
<td>Dissemination</td>
<td>Public</td>
<td></td>
</tr>
<tr>
<td>Delivered</td>
<td>February 1, 2016</td>
<td>Version</td>
<td>1.1</td>
<td></td>
</tr>
</tbody>
</table>
Contents

1 Introduction 7

2 Abstract Syntax 9

3 Textual Syntax 11
   3.1 Grammar ........................................ 11
   3.2 Extension Declaration ................................ 11
   3.3 Kind Declaration ................................ 12
   3.4 Mixin Declaration ................................ 13
   3.5 Attribute Declaration .............................. 14
   3.6 Action Declaration ................................ 15
   3.7 Data Type Declaration .............................. 16
   3.8 Enum Type Declaration ............................. 17
   3.9 Configuration Declaration ......................... 18
   3.10 Resource Declaration ............................. 19
   3.11 State Declaration ................................. 19
   3.12 Link Declaration ................................. 20
   3.13 Miscellaneous .................................... 20
      3.13.1 ID ........................................ 20
      3.13.2 URI ......................................... 21
      3.13.3 STRING ..................................... 21
      3.13.4 PositiveInteger ............................... 21
      3.13.5 Integer ..................................... 21

4 XML Based Textual Syntax 23
   4.1 Document ......................................... 23
   4.2 Extension ......................................... 23
   4.3 Category Attribute Groups .......................... 24
   4.4 Kind ............................................... 25
   4.5 Mixin ............................................... 25
   4.6 Attribute Definition ................................ 26
   4.7 Action Definition ................................ 26
   4.8 Configuration ..................................... 27
   4.9 Entity ............................................. 27
   4.10 Resource .......................................... 27
   4.11 Link ............................................... 28
   4.12 Attribute Instance ................................. 29

5 Conclusion 31
Deliverable | D2.3.1 | Status | Final
--- | --- | --- | ---
Expected | May 31, 2015 | Dissemination | Public
Delivered | February 1, 2016 | Version | 1.1

A OCCIware BNF 35

B OCCI XML Schema 37
List of Figures

1.1 Overview of the OCCIware DESCRIPTION LANGUAGE ........................... 8
2.1 OCCIware Metamodel ................................................................. 9
3.1 Syntax Diagram for <Grammar> .................................................. 11
3.2 Syntax Diagram for <ExtensionDecl> ......................................... 12
3.3 Syntax Diagram for <ImportDecl> ............................................. 12
3.4 Syntax Diagram for <KindDecl> ............................................... 13
3.5 Syntax Diagram for <KindRef> .................................................. 13
3.6 Syntax Diagram for <MixinDecl> .............................................. 13
3.7 Syntax Diagram for <MixinRef> ............................................... 14
3.8 Syntax Diagram for <AttributeDecl> ........................................ 14
3.9 Syntax Diagram for <DataTypeRef> .......................................... 15
3.10 Syntax Diagram for <ActionDecl> ............................................ 15
3.11 Syntax Diagram for <DataTypeDecl> ........................................ 16
3.12 Syntax Diagram for <DataTypeAnnotation> ............................... 17
3.13 Syntax Diagram for <EnumTypeDecl> ....................................... 18
3.14 Syntax Diagram for <ConfigurationDecl> ................................ 18
3.15 Syntax Diagram for <UseDecl> ................................................ 18
3.16 Syntax Diagram for <ResourceDecl> ......................................... 19
3.17 Syntax Diagram for <StateDecl> .............................................. 19
3.18 Syntax Diagram for <LinkDecl> .............................................. 20
3.19 Syntax Diagram for <ID> .......................................................... 20
3.20 Syntax Diagram for <URI> ........................................................ 21
3.21 Syntax Diagram for <STRING> ................................................. 21
3.22 Syntax Diagram for <PositiveInteger> ...................................... 21
3.23 Syntax Diagram for <Integer> .................................................. 22
<table>
<thead>
<tr>
<th>Deliverable</th>
<th>Expected</th>
<th>Status</th>
<th>Dissemination</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>D2.3.1</td>
<td>May 31, 2015</td>
<td>Final</td>
<td>Public</td>
<td>1.1</td>
</tr>
<tr>
<td></td>
<td>February 1, 2016</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The Open Cloud Computing Interface (OCCI) is a RESTful Protocol and API for all kinds of management tasks [1]. For this purpose, the OCCI Core Model [1] defines a small set of general-purpose concepts: Category, Kind, Mixin, Attribute, Action, Entity, Resource, and Link. Unfortunately, this core model is informally specified in natural language and then subject to ambiguities, imprecision, and incompleteness as discussed in [2] and [3]. Currently, OCCI designers describe their own OCCI extensions, aka new categories such as Kind, Mixin, and Action, and their Attributes, and OCCI configurations, aka Entity such as Resource and Link instances, informally in natural language illustrated by UML diagrams. Thus OCCI lacks a domain-specific description language to define both OCCI extensions and configurations unambiguously, precisely, and completely.

In [2] and [3], we have defined a precise metamodel of the eight OCCI’s concepts, and added four missed concepts: Extension, EDataType, Configuration, and AttributeState. This metamodel, called OCCIware METAMODEL, could be viewed as the abstract syntax of a precise domain-specific description language for OCCI, that we call OCCIware DESCRIPTION LANGUAGE.

The goal of this deliverable is to precisely define the structural part of the OCCIware DESCRIPTION LANGUAGE. More precisely, this deliverable reminds the abstract syntax defined in [2] and [3], then describes the concrete textual syntax of the OCCIware DESCRIPTION LANGUAGE (OCCIwareDL).

OCCIwareDL is a declarative language at the intercession between an Interface Description Language (IDL) and an Architecture Description Language (ADL). The IDL nature of OCCIwareDL is used to precisely define OCCI extensions, aka sets of Kinds, Mixins and data types. The ADL nature of OCCIwareDL is used to precisely define OCCI configurations, aka sets of Resource and Link instances. As any language, OCCIwareDL is composed of the classical pair of a syntax and a semantics as illustrated in Figure 1.1.

This deliverable is dedicated to the syntax of OCCIwareDL, which is composed of an abstract syntax and two concrete syntaxes. The abstract syntax of OCCIwareDL defines all the concepts of OCCIwareDL, their attributes and their relations independently of any concrete syntax. In this sense, this abstract syntax could be seen as an Abstract Syntax Tree (AST). This abstract syntax is defined by the precise metamodel presented in [2] and in the OCCIware PROJECT DELIVERABLE 2.2.1 [3], and is reminded in Chapter 2. The concrete syntax of OCCIwareDL provides a fine-grain point of view of both OCCI extensions and configurations, aka

---

allows a designer to describe all the details of an OCCI extension or configuration. The textual syntax is concretely implemented as a BNF-based syntax (see Chapter 3) and an XML-based syntax (see Chapter 4).

The semantics of OCCIwareDL is composed of the static semantics and the dynamic semantics. The static semantics of OCCIwareDL defines the constraints between the concepts of OCCIwareDL, which can be checked at compilation time. This static semantics is defined with the Object Constraint Language (OCL) in [2] and in the OCCIware Project Deliverable 2.2.1 [3]. The dynamic semantics of OCCIwareDL defines how OCCIwareDL descriptions are interpreted and executed at runtime. This is the operational semantics of OCCIwareDL and will be presented into the OCCIware Project Deliverable 2.3.2 [5].

This deliverable is organized as follows. Chapter 2 reminds the abstract syntax of OCCIwareDL, aka the metamodel presented in [2] and [3]. Chapter 3 defines the concrete BNF-based textual syntax of OCCIwareDL. Chapter 4 defines the concrete XML-based textual syntax of OCCIwareDL. Chapter 5 concludes on future perspectives. Finally, Appendixes A and B provide the whole BNF grammar and XML schema of OCCIwareDL, respectively.
Chapter 2

Abstract Syntax

This chapter defines the abstract syntax of OCCIwareDL. This abstract syntax captures all the concepts of OCCIwareDL independently of any concrete syntax, aka the textual syntax and the graphical notation. This abstract syntax is encoded by a metamodel, more precisely the OCCIware Metamodel defined in [2] and [3]. Figure 2.1 gives a graphical representation of this ECore metamodel. The eight concepts defined by the OCCI Core Model [1] are represented by boxes in white color, aka Category, Kind, Mixin, Attribute, Action, Entity, Resource, and Link. The four missed concepts added in [2] and [3] are represented by colored boxes, aka Extension, EDataType, Configuration, and AttributeState. For details, the reader can refer to the OCCIware Project Deliverable 2.2.1 [3].

Figure 2.1: OCCIware Metamodel

Let us note that as both Category and Entity are abstract ECore classes, aka classes that can not be instantiated directly, then these two concepts are not provided by the two concrete textual syntax defined in Chapter 3 and 4 respectively. Moreover, the EDataType class has a subclass called EEnum then this class is explicitly available in the two concrete syntaxes.
<table>
<thead>
<tr>
<th>Deliverable</th>
<th>D2.3.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected</td>
<td>May 31, 2015</td>
</tr>
<tr>
<td>Delivered</td>
<td>February 1, 2016</td>
</tr>
<tr>
<td>Status</td>
<td>Final</td>
</tr>
<tr>
<td>Dissemination</td>
<td>Public</td>
</tr>
<tr>
<td>Version</td>
<td>1.1</td>
</tr>
</tbody>
</table>
Chapter 3

Textual Syntax

This chapter defines precisely the textual syntax of OCCIwareDL as a Backus–Naur Form (BNF)\(^1\) grammar illustrated by syntax diagrams (or railroad diagrams)\(^2\). For this purpose, we used:

- The \LaTeX{} syntax package \(^3\) for encoding the BNF grammar of the textual syntax of OCCIwareDL.
- The Railroad Diagram Generator online service \(^4\) for checking the BNF grammar and generating syntax diagrams automatically.

The whole BNF is given in Appendix A.

3.1 Grammar

![Syntax Diagram for Grammar](image)

\[
\langle \text{Grammar} \rangle := \langle \text{ExtensionDecl} \rangle \mid \langle \text{ConfigurationDecl} \rangle
\]

An OCCIwareDL text file contains the declaration of one OCCI extension (cf Section 3.2) or one OCCI configuration (cf Section 3.9).

3.2 Extension Declaration

\[
\langle \text{ExtensionDecl} \rangle := \text{‘extension’ ‘ID’ ‘:’ ‘URI’ ImportDecl\(^*\) ( KindDecl \mid MixinDecl \mid DataTypeDecl \mid EnumTypeDecl \)\(^*\)}
\]

The declaration of an OCCI extension starts with the \textit{extension} keyword followed by both the \textsc{name} and \textsc{scheme} of the extension to declare. An extension may:

\(^1\)http://en.wikipedia.org/wiki/Backus-Naur_Form
\(^2\)http://en.wikipedia.org/wiki/Syntax_diagram
\(^3\)http://en.wikipedia.org/wiki/LaTeX_syntax_package
\(^4\)http://en.wikipedia.org/wiki/Railroad_Diagram_Generator
import one or more other extensions (cf Section 3.2),
• declare one or more kinds (cf Section 3.3),
• declare one or more mixins (cf Section 3.4),
• declare one or more data types (cf Section 3.7),
• declare one or more enum types (cf Section 3.8).

The importation of an extension starts with the import keyword followed by the scheme of the extension to import and optionally an alias to identify the imported extension later. When no alias is provided then the extension can be referenced by its name.

Example:

extension extensionA : 'http://www.occiware.org/extension-a'
import 'http://www.occiware.org/extension-b'
import 'http://www.occiware.org/extension-c' as C

3.3 Kind Declaration

(kindDecl) ::= ‘kind’ (ID) (‘extends’ (KindRef) )? ‘{’ (‘title’ (STRING) )? ( ⟨AttributeDecl⟩ | ⟨ActionDecl⟩ )* ‘}’

The declaration of a kind starts with the kind keyword followed by the name of the kind to declare. A kind may:

• extend a parent kind,
Figure 3.4: Syntax Diagram for \texttt{<KindDecl>}

- have a title,
- declare one or more attributes (cf Section 3.5),
- declare one or more actions (cf Section 3.6).

Figure 3.5: Syntax Diagram for \texttt{<KindRef>}

\[
\langle \text{KindRef} \rangle ::= \langle \text{ID} \rangle \ (',\ ' \langle \text{ID} \rangle )?
\]

The reference to a kind may be the \texttt{name} or a scoped name of the kind to reference.

Examples:

kind K1 extends Resource \{ title "K1 Resource" \}
kkind K2 extends C.K \{\}
kkind K3 \{\}

3.4 Mixin Declaration

Figure 3.6: Syntax Diagram for \texttt{<MixinDecl>}

\[
\langle \text{MixinDecl} \rangle ::= \texttt{mixin} \langle \text{ID} \rangle \ (\texttt{depends} \langle \text{MixinRef} \rangle \ (',\ ',\ \langle \text{MixinRef} \rangle )*)? \ (\texttt{applies} \langle \text{KindRef} \rangle \ (',\ ',\ \langle \text{KindRef} \rangle )*)? \ \texttt{\{ \ (\texttt{scheme} \langle \text{URI} \rangle )? \ (\texttt{title} \langle \text{STRING} \rangle )? \ (\langle \texttt{AttributeDecl} \rangle | \langle \texttt{ActionDecl} \rangle )* \}\}
\]

13 OCChware
The declaration of a mixin starts with the **mixin** keyword followed by the **name** of the mixin to declare. A mixin may:

- depend on one or more other mixins,
- be applicable to one or more kinds,
- define its own **scheme**,
- have a **title**,
- declare one or more attributes (cf Section 3.5),
- declare one or more actions (cf Section 3.6).

![Syntax Diagram for <MixinRef>](image)

\[
\texttt{<MixinRef> ::= \langle ID \rangle ( ','. \langle ID \rangle )?}
\]

The reference to a mixin may be the **name** or a scoped name of the mixin to reference.

**Examples:**

mixin M1 {}
mixin M2 { scheme ‘http://www.occiware.org/extension-a/scope’ }
mixin M3 { title "M3 Mixin" }
mixin M4 depends M1, C.M applies K1, C.K2 {}

### 3.5 Attribute Declaration

![Syntax Diagram for <AttributeDecl>](image)

\[
\texttt{<AttributeDecl> ::= ‘attribute’ ‘mutable’? ‘required’? (ID) ‘:’ <DataTypeRef> ( ‘*’ )? ( ‘=’ <STRING> )? ( ‘?’ ( ‘description’ <STRING> )? ‘?’ )?}
\]

The declaration of an attribute starts with the **attribute** keyword followed by both the **name** and the **type** of the attribute to declare. An attribute may:

- be **mutable**, 
-
• be required,
• have a default value,
• have a description.

3.6 Action Declaration

The declaration of an action starts with the action keyword followed by the name of the action to declare. An action may:

• declare one or more parameters,
• have a title.

The declaration of a parameter includes both the name and type of the parameter to declare.
Examples:

```plaintext
action start()
action stop(p : StopMethod)
action doSomething(p1 : Boolean, p2 : String, p3 : Number)
action foo() { title "This is the foo action" }
```

### 3.7 Data Type Declaration

![Syntax Diagram for <DataTypeDecl>](image)

The declaration of a data type starts with the `datatype` keyword followed by both the name and the implementation type of the data type to declare. A data type may:

- be serializable,
- have one or more data type annotations.

```plaintext
DataTypeDecl ::= 'datatype' ('serializable')? 'ID' ':' ('STRING' ( '{' (DataTypeAnnotation)+ '}' ) )?
```

Several annotations may be attached to a data type:

- `minExclusive` to set the exclusive minimal value of a number type.
- `minInclusive` to set the inclusive minimal value of a number type.
- `maxExclusive` to set the exclusive maximal value of a number type.
- `maxInclusive` to set the inclusive maximal value of a number type.
- `totalDigits` to set the total number of digits of a real type.
- `fractionDigits` to set the number of digits of the fraction part of a real type.
- `length` to set the length of a string type.
- `minLength` to set the minimal length of a string type.
- `maxLength` to set the maximal length of a string type.
- `WhiteSpace` to set the minimal or maximum number of white spaces.
- `pattern` to set a regular expression.
• \texttt{length} to set the exact length of a string type.

• \texttt{minLength} to set the minimal length of a string type.

• \texttt{maxLength} to set the maximal length of a string type.

• \texttt{WhiteSpace} to ...

• \texttt{pattern} to define a regular expression for values of a data type.

Examples:

\begin{verbatim}
datatype serializable Boolean : "boolean"
datatype serializable Number : "int"
datatype serializable String : "java.lang.String"
datatype serializable PositiveInteger : "int" { minInclusive = 0 }
datatype serializable Vlan : "int" { minInclusive = 0 maxExclusive = 4096 }
\end{verbatim}

3.8 Enum Type Declaration

\(\langle\text{EnumTypeDecl}\rangle ::= \text{'enum}' \langle \text{ID} \rangle \{ \langle \text{ID} \rangle \{ ',', \langle \text{ID} \rangle \} \} \text{' '}\rangle\)
The declaration of an enum type starts with the `enum` keyword followed by the `name` of the enum type to declare. An enum type may have one or more literal values.

Example:

```
enum StopMethod { graceful, acpioff, poweroff }
```

### 3.9 Configuration Declaration

The declaration of a configuration starts with the `configuration` keyword. A configuration may:

- use one or more extensions (`cf` Section 3.2),
- declare one or more resources (`cf` Section 3.10).

```
configuration

use 'http://www.occiware.org/extension-a'
use 'http://www.occiware.org/extension-b'
```

The use of an extension starts with the `use` keyword followed by the `scheme` of the extension to use and optionally an alias to identify the used extension later. When no alias is provided then the extension can be referenced by its `name`.

**Example:**

```
configuration
use 'http://www.occiware.org/extension-a'
use 'http://www.occiware.org/extension-b'
```
3.10 Resource Declaration

The declaration of a resource starts with the `resource` keyword followed by both the id and the kind of the resource to declare. A resource may:

- reference one or more mixins (cf Section 3.4).
- declare one or more attribute states (cf Section 3.11).
- declare one or more links (cf Section 3.12).

Examples:

resource 'c1' : Infrastructure.Compute {}  
resource 'c2' : Infrastructure.Compute mixins M1, C.M {}  
resource 'net1' : Infrastructure.Network {}

3.11 State Declaration

The declaration of an attribute state starts with the `state` keyword followed by both the name and the value of the state to declare.

Examples:

state status = "active"  
state counter = "10"
3.12 Link Declaration

\[
\text{\textit{LinkDecl}} := \text{`link`} \langle \text{URI} \rangle `:` \langle \text{KindRef} \rangle ( `\text{mixins}` \langle \text{MixinRef} \rangle ( `:` \langle \text{MixinRef} \rangle ) \ast )? `\text{target}` \langle \text{URI} \rangle `\text{\{} `\text{StateDecl}` \ast `\text{\}}`
\]

The declaration of a link starts with the \textit{link} keyword followed by the both \textit{id} and the \textit{kind} of the link to declare and the \textit{id} of the resource targeted by this link. A link may:

- reference one or more mixins (cf Section 3.4).
- declare one or more attribute states (cf Section 3.11).

Examples:

\[
\text{link `net11` : NetworkInterface mixins IPNetworkInterface target `net1` \{ }
\]

3.13 Miscellaneous

3.13.1 ID

\[
\text{\langle ID \rangle} := ( [a-zA-Z] | `_` ) ( [a-zA-Z] | [0-9] | `_` )^*
\]

Figure 3.18: Syntax Diagram for \textit{<LinkDecl>}

Figure 3.19: Syntax Diagram for \textit{<ID>}

OCClware
An identifier (or name) must start with a letter or the underscore character and then may contain a series of letters, digits, and underscore characters.

### 3.13.2 URI

\[
\langle \text{URI} \rangle ::= \text{""} [\text{any character}]^* \text{""}
\]

An URI must start and end with a quote character.

### 3.13.3 STRING

\[
\langle \text{STRING} \rangle ::= \text{""} [\text{any character}] \text{""}
\]

A string must start and end with a double quote character.

### 3.13.4 PositiveInteger

\[
\langle \text{PositiveInteger} \rangle ::= [0-9]+ 
\]

A positive integer is a series of digits.

### 3.13.5 Integer

\[
\langle \text{Integer} \rangle ::= '-'? \langle \text{PositiveInteger} \rangle 
\]

An integer is a positive integer optionally preceded by the minus character.
Figure 3.23: Syntax Diagram for <$\text{Integer}$>
Chapter 4

XML Based Textual Syntax

This chapter defines another textual syntax of OCCIwareDL as an XML Schema\(^1\). While not as compact as the syntax presented in Chapter 3, a wide number of parsers, validators and associated tools are already available for implementation.

This syntax is mostly based on preliminary work available at \([10]\) and implemented in erocci runtime \([11]\). The full XML schema is available in Appendix B.

4.1 Document

Listing 4.1: OCCI XML Document Root Elements

\[
<\text{xs:element name="extension" type="occi:extensionType"}>
<\text{xs:element name="configuration" type="occi:configurationType"}>
\]

An OCCI XML document contains the declaration of one OCCI extension (see Section 4.2) or one OCCI configuration (see Section 4.8).

4.2 Extension

Listing 4.2: XML Extension Type

\[
<\text{xs:complexType name="extensionType"}>
<\text{xs:sequence}>
<\text{xs:element name="import" type="extensionImportType" maxOccurs="unbounded" minOccurs="0"} />
<\text{xs:element name="kind" type="occi:kindType" maxOccurs="unbounded" minOccurs="0"} />
<\text{xs:element name="mixin" type="occi:mixinType" maxOccurs="unbounded" minOccurs="0"} />
</\text{xs:sequence}>
<\text{xs:attribute name="name" type="xs:string"}>
<\text{xs:attribute name="scheme" type="xs:anyURI"}>
<\text{xs:attribute name="version" type="xs:string"}>
<\text{xs:attribute name="status"}>
<\text{xs:restriction base="xs:string"}>
<\text{xs:enumeration value="draft"}>
<\text{xs:enumeration value="experimental"}>
<\text{xs:enumeration value="stable"}>
\]

\(^1\)https://en.wikipedia.org/wiki/XML_Schema_%28W3C%29
The extension element type is used to declare an OCCI extension. The following attributes are not part of the metamodel. They have been added for easing maintenance and reuse of extensions.

- version,
- status, with possible values:
  - draft: the extension must be considered as ongoing work;
  - experimental: the extension has been implemented at least once, but not officially part of OCCI specifications, as defined by OCCI Working Group;
  - stable: the extension has been implemented and is part of the OCCI specifications.

Listing 4.3: Extension Element Example

```xml
<import scheme="http://schemas.ogf.org/occi/core#" />
</extension>
```

4.3 Category Attribute Groups

Listing 4.4: Category ID Attribute Group

```xml
<xs:attributeGroup name="categoryIdGroup">
  <xs:attribute name="scheme" type="xs:anyURI"/>
  <xs:attribute name="term" type="xs:string" use="required"/>
</xs:attributeGroup>
```

The categoryIdGroup attribute group is used to identify uniquely an OCCI Category or derivative of, i.e., Kind, Mixin and Action as of OCCI Core specification version 1.2.

For instance, it is used for building parent (see Section 4.4), depends and applies (see Section 4.5) elements.

**Note:** wherever scheme attribute is omitted, it is equal to the one of the parent element.

Listing 4.5: Category Attribute Group

```xml
<xs:attributeGroup name="categoryGroup">
  <xs:attributeGroup ref="occi:categoryGroup"></xs:attributeGroup>
  <xs:attribute name="title" type="xs:string"/>
</xs:attributeGroup>
```

The categoryGroup attribute group is used to describe an OCCI Category or derivative of. For instance it is used to build the kind and mixin elements.
4.4 Kind

Listing 4.6: XML Kind Type

```xml
<xs:complexType name="kindType">
  <xs:sequence>
    <xs:element name="parent" type="occi:categoryType" minOccurs="1" maxOccurs="1" />
    <xs:element name="attribute" type="occi:attributeSpecType" minOccurs="0" maxOccurs="unbounded" />
    <xs:element name="action" type="occi:actionSpecType" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attributeGroup ref="occi:categoryGroup"/>
</xs:complexType>
```

The `kindType` element type is used to declare an OCCI Kind.

Listing 4.7: Kind Element Example

```xml
<kind term="compute" title="Compute resource">
  <parent scheme="http://schemas.ogf.org/occi/core#" term="resource"/>
  <attribute .../>
  <action .../>
</kind>
```

4.5 Mixin

Listing 4.8: XML Mixin Type

```xml
<xs:complexType name="mixinType">
  <xs:sequence>
    <xs:element name="depends" type="occi:categoryType" minOccurs="0" maxOccurs="unbounded" />
    <xs:element name="applies" type="occi:categoryType" minOccurs="0" maxOccurs="unbounded" />
    <xs:element name="attribute" type="occi:attributeSpecType" minOccurs="0" maxOccurs="unbounded" />
    <xs:element name="action" type="occi:actionSpecType" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attributeGroup ref="occi:categoryGroup"/>
</xs:complexType>
```

The `mixinType` element type is used to declare an OCCI Mixin.

Listing 4.9: Mixin Element Example

```xml
<mixin term="ipnetworkinterface" title="IP Network Interface Mixin" scheme="http://schemas.ogf.org/occi/infrastructure/networkinterface#/">
  <applies scheme="http://schemas.ogf.org/occi/infrastructure#/" term="networkinterface"/>
  <attribute .../>
</mixin>
```
4.6 Attribute Definition

Listing 4.10: XML AttributeSpec Type

```xml
<xs:complexType name="attributeSpecType">
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="type" type="occi:attributeTypeType" use="required"/>
  <xs:attribute name="default" type="xs:string"/>
  <xs:attribute name="required" type="xs:boolean" default="false"/>
  <xs:attribute name="mutable" type="xs:boolean"/>
</xs:complexType>
```

Listing 4.11: XML Attribute Type Type

```xml
<xs:simpleType name="attributeTypeType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="string"></xs:enumeration>
    <xs:enumeration value="boolean"></xs:enumeration>
    <xs:enumeration value="integer"></xs:enumeration>
    <xs:enumeration value="float"></xs:enumeration>
  </xs:restriction>
</xs:simpleType>
```

The `attributeSpecType` element type is used to declare an attribute definition inside an OCCI Category definition. Attribute type is defined by the `attributeTypeType` type and is currently limited to a subset of XSD built-in data types. Allowed data types are: `string`, `float`, `integer`, `boolean`.

Listing 4.12: Attribute Spec Element Example

```xml
<attribute name="occi.compute.memory" type="float" title="System RAM (GB)"/>
<attribute name="occi.compute.state" use="required" default="inactive" immutable="true" title="System state" type="string"/>
```

4.7 Action Definition

Listing 4.13: XML ActionSpec Type

```xml
<xs:complexType name="actionSpecType">
  <xs:sequence>
    <xs:element name="attribute" type="occi:attributeSpecType" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attributeGroup ref="occi:categoryGroup"/>
</xs:complexType>
```

The `actionSpecType` element type is used to declare an action definition inside an OCCI Category definition.

Listing 4.14: Action Element Example

```xml
<action term="start" scheme="http://schemas.ogf.org/occi/infrastructure/compute/action#" title="Start the system"></action>
```
4.8 Configuration

Listing 4.15: XML Configuration Type

```xml
<xs:complexType name="configurationType">
  <xs:sequence>
    <xs:element name="resource" type="occi:resourceType" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="name" type="xs:string"></xs:attribute>
  <xs:attribute name="id" type="xs:anyURI"></xs:attribute>
  <xs:attribute name="version" type="xs:string"></xs:attribute>
</xs:complexType>
```

The `configuration` element type is used to declare an OCCI configuration. Optionally, configuration may be given a name, a version and a URI identifying this configuration.

Listing 4.16: Configuration Element Example

```xml
<configuration xmlns="http://schemas.ogf.org/occi" name="Project Foo" version="201507010001" id="http://example.org/occi/project15">
  <resource ...
  <resource ...
</configuration>
```

4.9 Entity

Listing 4.17: XML Entity Type

```xml
<xs:complexType name="entityType">
  <xs:sequence>
    <xs:element name="kind" type="occi:categoryType" minOccurs="1" maxOccurs="1" />
    <xs:element name="mixin" type="occi:categoryType" minOccurs="0" maxOccurs="unbounded" />
    <xs:element name="attribute" type="occi:attributeType" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="id" type="xs:anyURI"></xs:attribute>
  <xs:attribute name="title" type="xs:string"></xs:attribute>
</xs:complexType>
```

The `entity` type is the base definition of elements representing OCCI instantiable entities, like `Resource` and `Link`.

4.10 Resource

Listing 4.18: XML Resource Type

```xml
<xs:complexType name="resourceType">
  <xs:complexContent>
    <xs:extension base="occi:entityType">
      <xs:sequence>
        <xs:element name="summary" type="xs:string" minOccurs="0" maxOccurs="1" />
      </xs:sequence>
    </xs:extension>
</xs:complexType>
```
The resource element type is used to declare an OCCI Resource instance. It is an extension of the entity element type.

Listing 4.19: Resource Element Example

```xml
<resource id="http://example.org/occi/project15/myresource1" title="A Rare Resource">
  <kind scheme="#" term="compute"/>
  <attribute name="occi.compute.arch" value="x64"/>
</resource>
```

4.11 Link

Listing 4.20: XML Inline Link Type

```xml
<xs:complexType name="inlineLinkType">
  <xs:extension base="occi:entityType">
    <xs:attribute name="target" type="xs:anyURI"/>
  </xs:extension>
</xs:complexType>
```

Listing 4.21: XML Link Type

```xml
<xs:complexType name="linkType">
  <xs:complexContent>
    <xs:extension base="occi:inlineLinkType"/>
    <xs:attribute name="src" type="xs:anyURI"/>
  </xs:complexContent>
</xs:complexType>
```

The inlineLinkType and linkType element types are used to declare an OCCI Link instance. A resource instance can contain the definition of links originating from it, ie whose src attribute equals the id of the resource. In this case, the inlineLinkType type must be used.

In the case the link is defined outside the definition of a resource, the src attribute of the link must be explicitated, and the linkType type used.

Listing 4.22: Inline Link Element Example

```xml
<resource id="http://example.org/occi/project15/mycompute1" title="A Compute">
  <kind scheme="#" term="compute"/>
  <attribute .../>
  <link id="http://example.org/occi/project15/nic1" title="a NIC" target="http://example.org/occi/project15/network1">
    <attribute .../>
  </link>
</resource>
```
4.12 Attribute Instance

The `attributeType` element type is used to declare an OCCI attribute instance of an entity.
<table>
<thead>
<tr>
<th>Deliverable</th>
<th>D2.3.1</th>
<th>Status</th>
<th>Final</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected</td>
<td>May 31, 2015</td>
<td>Dissemination</td>
<td>Public</td>
</tr>
<tr>
<td>Delivered</td>
<td>February 1, 2016</td>
<td>Version</td>
<td>1.1</td>
</tr>
</tbody>
</table>
Chapter 5

Conclusion

Currently, OCCI designers describe both their own OCCI extensions and configurations in informal natural language illustrated by semi-formal UML diagrams. This is subject to ambiguities, imprecision, and incompleteness as discussed in [2] and the OCCIWARE PROJECT DELIVERABLE 2.2.1 [3]. Thus OCCI lacks of a domain-specific description language to define both OCCI extensions and configurations unambiguously, precisely, and completely.

This deliverable defines the textual syntax of the OCCIWARE DESCRIPTION LANGUAGE (OCCIWAREDL). OCCIWAREDL is a declarative language at the intercession between an Interface Description Language (IDL) and an Architecture Description Language (ADL) [4]. The IDL nature of OCCIWAREDL is used to precisely define OCCI extensions, aka sets of kinds, mixins and data types. The ADL nature of OCCIWAREDL is used to precisely define OCCI configurations, aka sets of resource and link instances.

The two concrete textual syntaxes of OCCIWAREDL defined in this deliverable are used in the OCCIWARE PROJECT DELIVERABLE 2.4.1 [12] to describe the set of standard OCCI Extensions used in the OCCIWARE project.

As perspective, the BNF-based textual concrete syntax of OCCIWAREDL presented in Chapter 3 will be implemented in the OCCIWARE PROJECT DELIVERABLES 3.2.2 [15] and 3.2.3 [14]. This implementation will be based on the Eclipse XText technology [1]. The XML-based textual concrete syntax of OCCIWAREDL presented in Chapter 4 will be implemented in the OCCIWARE RUNTIME, namely erocci. A graphical notation of OCCIWAREDL will be implemented in the OCCIWARE PROJECT DELIVERABLES 3.3.1 [16], 3.3.2 [16], and 3.3.3 [17]. This implementation will be based on the Eclipse Sirius technology [2]. The dynamic semantics of OCCIWAREDL will be defined into the OCCIWARE PROJECT DELIVERABLE 2.3.2 [4].

Acknowledgment

This work is supported by the OCCIware Research and Development Project (www.occiware.org) funded by French Programme d’Investissements d’Avenir (PIA). Thanks to Vincent Zurczak from Linagora for its suggestions on the BNF-based textual syntax of OCCIWAREDL.

Bibliography


Appendix A

OCCIware BNF

\[
\text{(Grammar)} := \text{(ExtensionDecl)} | \text{(ConfigurationDecl)}
\]

\[
\text{(ExtensionDecl)} := \text{‘extension’ (ID) ‘:’ (URI) ImportDecl* ( (KindDecl) | (MixinDecl) | }
\]

\[
\text{(DataTypeDecl)} | \text{(EnumTypeDecl)*)}
\]

\[
\text{(ImportDecl)} := \text{‘import’ (URI) ( ‘as’ (ID) )?}
\]

\[
\text{(KindDecl)} := \text{‘kind’ (ID) (‘extends’ (KindRef) )? ‘{’ (‘title’ (STRING) )? ( (AttributeDecl) }
\]

\[
\text{| (ActionDecl)*) ‘}’}
\]

\[
\text{(KindRef)} := (ID) ( ‘.’ (ID) )?
\]

\[
\text{(MixinDecl)} := \text{‘mixin’ (ID) (‘depends’ (MixinRef) ( ‘,‘ (MixinRef) )* )? (‘applies’ (KindRef) }
\]

\[
\text{ ( ‘,’ (KindRef) )* )? ‘{’ (‘scheme’ (URI) )? ( ‘title’ (STRING) )? ( (AttributeDecl) | }
\]

\[
\text{ (ActionDecl)*) ‘}’}
\]

\[
\text{(MixinRef)} := (ID) ( ‘.’ (ID) )?
\]

\[
\text{(AttributeDecl)} := \text{‘attribute’ ‘mutable’? ‘required’? (ID) ‘:’ (DataTypeRef) ( ‘*’ )? ( ‘=’ }
\]

\[
\text{ (STRING) )? ( ‘{’ (‘description’ (STRING) )? ‘}’ )?}
\]

\[
\text{(DataTypeRef)} := (ID) ( ‘.’ (ID) )?
\]

\[
\text{(ActionDecl)} := \text{‘action’ (ID) ‘{’ ( (ID) ‘:’ (DataTypeRef) ‘*’ )? ( ‘,’ (ID) ‘:’ (DataTypeRef) }
\]

\[
\text{ ‘*’ )? ‘}’ ( ‘{’ ‘title’ (STRING) ‘}’ )?}
\]

\[
\text{(DataTypeDecl)} := \text{‘datatype’ (‘serializable’)? (ID) ‘:’ (STRING) ( ‘{’ (DataTypeAnnotation) + }
\]

\[
\text{ ‘}’ )?
\]

\[
\text{(DataTypeAnnotation)} := \text{‘minExclusive’ ‘=’ (Integer) }
\]

\[
\text{| ‘maxExclusive’ ‘=’ (Integer) }
\]

\[
\text{| ‘minInclusive’ ‘=’ (Integer) }
\]

\[
\text{| ‘maxInclusive’ ‘=’ (Integer) }
\]

\[
\text{| ‘totalDigits’ ‘=’ (PositiveInteger) }
\]

\[
\text{| ‘fractionDigits’ ‘=’ (PositiveInteger) }
\]

\[
\text{| ‘length’ ‘=’ (PositiveInteger) }
\]

\[
\text{| ‘minLength’ ‘=’ (PositiveInteger) }
\]

\[
\text{| ‘maxLength’ ‘=’ (PositiveInteger) }
\]
<table>
<thead>
<tr>
<th>Deliverable</th>
<th>D2.3.1</th>
<th>Status</th>
<th>Final</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected</td>
<td>May 31, 2015</td>
<td>Dissemination</td>
<td>Public</td>
</tr>
<tr>
<td>Delivered</td>
<td>February 1, 2016</td>
<td>Version</td>
<td>1.1</td>
</tr>
</tbody>
</table>

```
\[ \text{'whiteSpace'} \text{'}\text{=} \langle \text{PositiveInteger} \rangle \\
\text{'pattern'} \text{'}\text{=} \langle \text{STRING} \rangle \\
\langle \text{EnumTypeDecl} \rangle \text{::=} \text{'enum'} \langle \text{ID} \rangle \{ \langle \text{ID} \rangle \{ \langle \text{ID} \rangle \} \} \} \\
\langle \text{ConfigurationDecl} \rangle \text{::=} \text{'configuration'} \langle \text{UseDecl} \rangle \} \langle \text{ResourceDecl} \rangle \} \\
\langle \text{UseDecl} \rangle \text{::=} \text{'use'} \langle \text{URI} \rangle \{ \text{'as'} \langle \text{ID} \rangle \} \} \\
\langle \text{ResourceDecl} \rangle \text{::=} \text{'resource'} \langle \text{URI} \rangle \{ \langle \text{KindRef} \rangle \{ \text{'mixins'} \langle \text{MixinRef} \rangle \{ \langle \text{ID} \rangle \} \} \} \} \} \langle \text{StateDecl} \rangle \} \langle \text{LinkDecl} \rangle \} \} \\
\langle \text{StateDecl} \rangle \text{::=} \text{'state'} \langle \text{ID} \rangle \text{'}\text{=} \langle \text{STRING} \rangle \\
\langle \text{LinkDecl} \rangle \text{::=} \text{'link'} \langle \text{URI} \rangle \{ \langle \text{KindRef} \rangle \{ \text{'mixins'} \langle \text{MixinRef} \rangle \{ \langle \text{ID} \rangle \} \} \} \} \} \langle \text{StateDecl} \rangle \} \} \} \} \\
\langle \text{ID} \rangle \text{::=} \langle \text{[a-zA-Z]} \rangle \langle \_ \rangle \langle \text{[a-zA-Z]} \rangle \langle \text{[0-9]} \rangle \langle \_ \rangle \} \\
\langle \text{URI} \rangle \text{::=} \langle \text{''} \rangle \langle \text{[any character]} \rangle \langle \text{''} \rangle \\
\langle \text{STRING} \rangle \text{::=} \langle \text{''} \rangle \langle \text{[any character]} \rangle \langle \text{''} \rangle \\
\langle \text{PositiveInteger} \rangle \text{::=} \langle \text{[0-9]} \rangle \} \\
\langle \text{Integer} \rangle \text{::=} \langle \_ \rangle \} \langle \text{PositiveInteger} \rangle \\
```
Appendix B

OCCI XML Schema

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://schemas.ogf.org/occi"
  elementFormDefault="qualified" version="0.2"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xl="http://www.w3.org/2008/06/xlink"
  xmlns:occi="http://schemas.ogf.org/occi">
<xs:element name="parent" type="occi:categoryType" minOccurs="1" maxOccurs="1" />
<xs:element name="attribute" type="occi:attributeSpecType" minOccurs="0" maxOccurs="unbounded" />
<xs:element name="action" type="occi:actionSpecType" minOccurs="0" maxOccurs="unbounded" />
</xs:sequence>
<xs:attributeGroup ref="occi:categoryGroup"></xs:attributeGroup>
</xs:complexType>

<xs:complexType name="mixinType">
  <xs:sequence>
    <xs:element name="depends" type="occi:categoryType" minOccurs="0" maxOccurs="unbounded" />
    <xs:element name="applies" type="occi:categoryType" minOccurs="0" maxOccurs="unbounded" />
    <xs:element name="attribute" type="occi:attributeSpecType" minOccurs="0" maxOccurs="unbounded" />
    <xs:element name="action" type="occi:actionSpecType" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attributeGroup ref="occi:categoryGroup"></xs:attributeGroup>
</xs:complexType>

<xs:complexType name="attributeSpecType">
  <xs:attribute name="name" type="xs:string" use="required" />
  <xs:attribute name="type" type="occi:attributeTypeType" use="required" />
  <xs:attribute name="default" type="xs:string" />
  <xs:attribute name="required" type="xs:boolean" default="false" />
  <xs:attribute name="mutable" type="xs:boolean" />
</xs:complexType>

<xs:simpleType name="attributeTypeType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="string"/>
    <xs:enumeration value="boolean"/>
    <xs:enumeration value="integer"/>
    <xs:enumeration value="float"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="actionSpecType">
  <xs:sequence>
    <xs:element name="attribute" type="occi:attributeSpecType" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attributeGroup ref="occi:categoryGroup"></xs:attributeGroup>
</xs:complexType>

<xs:complexType name="categoryType">
  <xs:attributeGroup ref="occi:categoryIdGroup"></xs:attributeGroup>
</xs:complexType>

<xs:attributeGroup name="categoryIdGroup">
</xs:attributeGroup>
<xs:attribute name="title" type="xs:string" />
</xs:complexType>

<xs:complexType name="attributeType">
  <xs:attribute name="name" type="xs:string" />
  <xs:attribute name="value" type="xs:string" />
</xs:complexType>

</xs:schema>