Validating CC/PP and UAProf Profiles

Charles Smith[1], Mark H. Butler
Information Infrastructure Laboratory
HP Laboratories Bristol
HPL-2002-268
October 11th , 2002*

E-mail: chasmi@hplb.hpl.hp.com, smithcr@tcd.ie, mark-h_butler@hp.com

CC/PP, UAProf, validation, capability description, delivery context, device independence

CC/PP and UAProf are two related standards, proposed by the W3C and the Open Mobile Alliance respectively, that allow devices to communicate their capabilities to other devices such as web servers. This paper explores ways of guaranteeing interoperability in CC/PP aware devices by validating their capability descriptions. Firstly it explains what validation can be performed on CC/PP and UAProf profiles. Secondly it investigates two methods of performing validation: using an XML Schema derived automatically from the RDF Schema describing the vocabulary and a more comprehensive validation technique implemented using an RDF parser. Methods of adding additional information to RDF Schemas that can aid the validation process are also discussed.

[1] Trinity College Dublin, Dublin, Eire.

Validating CC/PP and UAProf Profiles

Charles Smith (chasmi@hplb.hpl.hp.com, smithcr@tcd.ie)[*]
Mark H. Butler (mark-h_butler@hp.com)
Hewlett Packard Laboratories, Bristol UK
11 September 2002

Abstract

CC/PP and UAProf are two related standards, proposed by the W3C and the Open Mobile Alliance respectively, that allow devices to communicate their capabilities to other devices such as web servers. This paper explores ways of guaranteeing interoperability in CC/PP aware devices by validating their capability descriptions. Firstly it explains what validation can be performed on CC/PP and UAProf profiles. Secondly it investigates two methods of performing validation: using an XML Schema derived automatically from the RDF Schema describing the vocabulary and a more comprehensive validation technique implemented using an RDF parser. Methods of adding additional information to RDF Schemas that can aid the validation process are also discussed.

Keywords

CC/PP, UAProf, Validation, Capability Description, Delivery Context, Device Independence

1 Introduction

As the variety of computing devices increases, there is a growing need for flexible and application independent capability negotiation. CC/PP (Composite Capabilities / Preferences Profiles)[1] provides a standard way for devices to transmit their capabilities and user preferences when requesting web content. Servers and proxies receiving this information can then provide content appropriate to the particular device. Technologies such as CC/PP are essential to the problem of device independence[2] since they allow different devices to specify their capabilities in a uniform way.

Currently virtually all CC/PP capable devices use UAProf[3], a standard developed by the Open Mobile Alliance (formerly the WAP Forum) to allow Internet enabled phones to send a profile of their capabilities to a server. UAProf predates CC/PP so CC/PP is designed to be backwardly compatible with UAProf. Furthermore UAProf, unlike CC/PP, defines a set of vocabularies for describing device capabilities. CC/PP, on the other hand, is designed to be vocabulary and application agnostic. This is achieved by leveraging XML namespaces so that different profiles may use one or more vocabularies to describe device capabilities.

The CC/PP and UAProf data format is based on RDF[4] and represents device capabilities as a two level hierarchy consisting of *components* and *properties* e.g. HardwarePlatform and ColorCapable respectively. Therefore in essence a CC/PP profile takes the form of a structured set of property and value pairs. A CC/PP vocabulary provides application specific information shown in Table 1. CC/PP and UAProf recommend that RDF Schema[5] should be

[*] Charles Smith is a summer intern at HP Labs Bristol from Trinity College Dublin, Eire.

used to define vocabularies and CC/PP gives a schema that should be extended by specific vocabularies to define component types and properties. However these schemas only define a subset of vocabulary information as indicated in Table 1. Note currently UAProf places some of this information in comments in the schema making it more difficult for processors to extract. In addition UAProf does not currently define the set of values a property can take, instead giving example property values without defining their meaning. Ideally values as well as properties should be defined within vocabularies otherwise there is a danger that vendors will use different values to mean the same thing or use the same value to mean different things. Furthermore unless these values are defined it is possible that different vendors will use different capitalization schemes or introduce other variations that make interpretation difficult such as variations due to local language e.g. "Yes" and "No" versus "Ja" and "Nein".

Vocabulary Information	**Is information currently expressed in RDF Schemas associated with vocabularies?**
The set of valid property names.	Yes
The set of valid component names.	Yes
The parent components for each property.	Yes
The data type of each property i.e. literal, Boolean, positive integer, rational or custom.	Forthcoming - when RDF Core reaches a decision on datatyping. Stored in comments in UAProf.
Whether each property is single or multi-valued.	Yes
For multi-valued properties, whether those values are ordered or unordered.	Yes
In the case of UAProf, how to merge multiple values of the same property.	No Stored in comments in UAProf.
Where a property can take a defined set of values, a vocabulary may explicitly define the allowable set of values and explain the meaning of each value.	No

Table 1 - CC/PP Vocabulary Information

Previous work at HP Labs has resulted in DELI[6], a CC/PP processor that can be used in Java Servlets, allowing application developers to use CC/PP information in web applications without needing to worry about CC/PP profile structure, protocol or resolution. This work has also identified a number of potential interoperability issues with CC/PP that could be obstacles to widespread deployment [7],[8] some of which are due to errors in profiles. For example profiles have been released that are not valid RDF, or in some cases not valid XML, that spell property names incorrectly, or that contain other errors relating to profile structure. These problems are generally identified and resolved over time, but it would be useful to be able to validate profiles i.e. guarantee that profiles are correct before they are released. Furthermore as the companies responsible for authoring profiles are often different to those creating CC/PP processors, it is possible there may be disagreements about responsibility when interoperability problems occur. By having a clear validation procedure that is fair and everyone can understand, such disagreements are avoided and companies can instead concentrate on whether their profiles or processor are compliant.

This paper explores techniques for validating CC/PP profiles using XSLT[9] and XML Schema[10]. It also explores an alternative validation method implemented using an RDF parser. As the only vocabularies currently in common use are the UAProf vocabularies this paper concentrates on UAProf, but the same techniques can be applied to any CC/PP vocabulary.

2 Validation

In order to validate CC/PP profiles, there must be a set of rules that determine what constitutes a valid profile. According to the CC/PP Structure and Vocabularies Working Draft[11] a CC/PP profile MUST meet the following constraint: a profile must be valid XML and a valid XML serialisation of RDF. The W3C's RDF validation service[12] can be used for this task. In addition it MAY contain the following resources: firstly it may contain resources identified with a component property associated with either a CC/PP namespace or a UAProf namespace. Secondly it may contain resources identified with a default or Default property associated with either a CC/PP namespace or a UAProf namespace.

However it is the authors' belief that these requirements are insufficient to guarantee interoperability. Therefore we argue that the other important element of validation is ensuring that profiles conform to the vocabulary or set of vocabularies they reference when they declare the namespaces for a profile. However this type of validation is controversial for two reasons: firstly some members of the RDF community think such validation constrains how RDF may be used. However, as CC/PP is a specific RDF application it is free to define additional restrictions on RDF usage. Secondly there is no guarantee in CC/PP that an RDF Schema description of a vocabulary will be available from the same URI used for a namespace in a profile. However, here we will make the assumption that vocabulary authors and profile authors follow recommended CC/PP best practice and make RDF Schemas available from the appropriate URIs. Alternatively appropriate schemas may be available from other sources e.g. they are distributed with some CC/PP processors. If the correct schema is available, we propose that it is also possible to use the following validation rules, derived from the vocabulary information in Table 1:

- The profile only uses properties defined in that schema.
- The profile only uses components defined in that schema.
- That the profile only places properties in components specified in the schema.
- If the schema contains XML Schema datatyping information for specific properties that the profile agrees with that datatyping information.
- If the schema specifies that a property is a bag or sequence that the profile uses a bag or sequence for that property.
- If the schema defines a set of allowable values for a property, that the profile only uses values from that set. For example consider a property called "Keyboard". This might be able to possess the following values: QWERTY, DVORAK, T9, PHONE and ON-SCREEN.

3 Validation using XML Schema

It is important to note that although RDF Schema and XML Schema are both schema languages they perform slightly different roles: RDF Schema's primary aim is to provide a machine-readable description of a particular vocabulary rather than provide mechanisms for validating data. XML Schema, on the other hand, can be used to validate XML documents and enforce strict structural and datatype constraints. Therefore one solution to the validation problem in CC/PP would be to use XML Schema to validate profiles. In order to use XML Schema in this way, it is necessary to solve another related problem: in the XML serialisation of RDF it is possible to serialise a single RDF graph in several different ways, making the required XML Schema complex and unwieldy. The solution proposed here is to use XSLT to convert a profile to a constrained form of RDF that maintains all the information from the original serialisation. After this the profile can be validated using XML Schema, to ensure that it is both syntactically correct and that it uses all referenced vocabularies correctly. This process is shown diagrammatically in Figure 1.

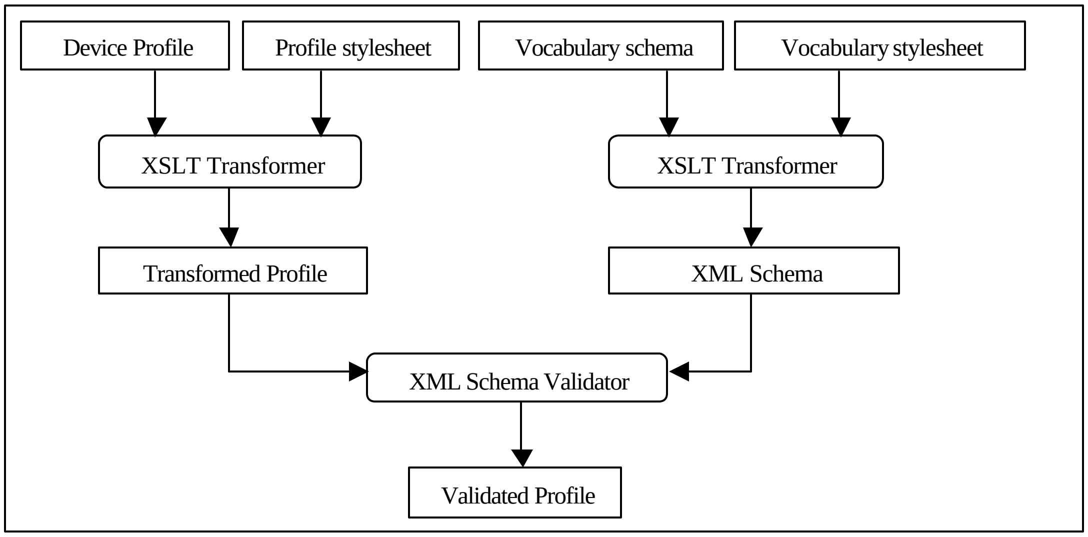

Figure 1 - Validating CC/PP Profiles using XSLT and XML Schema

3.1 Profile format conversion

One complexity when dealing with the XML serialisation of RDF is it is possible to represent the same underlying RDF model in many different ways necessitating the use of a specialist RDF parser[13]. One solution to this problem is to constrain the XML serialisation of RDF so that it is possible to process the profile using a standard XML parser. Here we use an XSLT stylesheet to transform UAProf profiles into this form so they can be validated using XML Schema. This transformation also automatically corrects some common errors made in device profiles and where necesssary converts the profile to an alternative serialisation that produces and identical RDF graph to the original form. This involves a number of individual operations. Firstly it ensures that all RDF specific attributes are qualified with the correct namespace prefix. This is required by RDF syntax according to the RDF Core Working Group, which has decided:

"On 25th May 2001, the WG decided that ALL attributes must be namespace qualified."[14]

Secondly it ensures that type statements are not omitted from components. In CC/PP, it is possible to omit type statements from components because the use of type statements was not mandatory in early versions of UAProf. However the latest version of the UAProf specification states that components in the profile must correspond to the components defined in the schema and must be identified by the `rdf:type` attribute.

Type statements are important to a CC/PP processor as discussed in [7]:

“Many people seem to find it counter-intuitive that you need to declare that the component is called HardwarePlatform twice. It is important to note the first use of HardwarePlatform defines the instance name whereas the second use is to define the type. This is just as if we defined an object in Java e.g.

HardwarePlatform hardwarePlatform;

In Java we would not expect the complier to determine the object type from the instance name, but some CC/PP implementers omit the type statements from components. This means the CC/PP processor cannot recognise components as the ID is just a local name.”

If a component is not typed correctly, the stylesheet will attempt to determine its type from its `rdf:ID` or `rdf:about` attribute.

Thirdly the stylesheet ensures that where possible all component resources are expressed using the RDF typedNode syntax. Thus component definitions such as:

```
<rdf:Description rdf:ID=”HardwarePlatform”>
  <rdf:type rdf:resource=”http://www.wapforum.org/profiles/UAPROF/ccppschema-
20020710#HardwarePlatform”>
  …
</rdf:Description>
```

becomes:

```
<prf:HardwarePlatform rdf:ID=”HardwarePlatform”>
  …
</prf:HardwarePlatform>
```

Fourthly any properties that use references to resources internal to the document are changed to express those resources “inline”. For example:

```
<prf:OutputCharSet rdf:resource=”#genid1”/>

<rdf:Bag rdf:ID=”genid1”>
   <rdf:li>US-ASCII</rdf:li>
</rdf:Bag>
```

will be changed to:

```
<prf:OutputCharSet>
  <rdf:Bag>
     <rdf:li>US-ASCII</rdf:li>
  </rdf:Bag>
</prf:OutputCharSet>
```

Fifthly properties expressed as attributes using the RDF abbreviated syntax are changed to give the properties as elements instead. Thus:

```
<prf:component>
  <prf:BrowserUA rdf:ID="BrowserUA" prf:BrowserName="Microsoft"/>
</prf:component>
```

will be expressed as:

```
<prf:component>
  <prf:BrowserUA rdf:ID="BrowserUA">
    <prf:BrowserName>Microsoft</prf:BrowserName>
  </prf:BrowserUA>
</prf:component>
```

Finally the datatype constraints of XML Schema also require some changes to Boolean values, as UAProf uses "yes" and "no" rather than the standard XML Schema values of "true" and "false". These transformed profiles act like a canonical form of the profile, which is valid RDF, can be validated, and meets the UAProf specification.

Here is an example profile before transformation:

```
<?xml version="1.0"?>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:prf="http://www.wapforum.org/profiles/UAPROF/ccppschema-20020709#">
  <rdf:Description ID="MSIE">
    <prf:component>
      <rdf:Description ID="HardwarePlatform">
        <prf:ColorCapable>Yes</prf:ColorCapable>
        <prf:Keyboard>Qwerty</prf:Keyboard>
        <prf:Vendor>Microsoft</prf:Vendor>
        <prf:SoundOutputCapable>Yes</prf:SoundOutputCapable>
        <prf:StandardFontProportional>Yes</prf:StandardFontProportional>
      </rdf:Description>
    </prf:component>
    <prf:component>
      <rdf:Description ID="SoftwarePlatform">
        <prf:CcppAccept-Charset rdf:resource="#genid1"/>
      </rdf:Description>
    </prf:component>
    <prf:component>
      <rdf:Description ID="BrowserUA" prf:FramesCapable="Yes"/>
    </prf:component>
  </rdf:Description>
  <rdf:Bag rdf:ID="genid1">
    <rdf:li>US-ASCII</rdf:li>
    <rdf:li>ISO-8859-1</rdf:li>
    <rdf:li>UTF-8</rdf:li>
    <rdf:li>ISO-10646-UCS-2</rdf:li>
  </rdf:Bag>
</rdf:RDF>
```

Here is the same profile after transformation:

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:prf="http://www.wapforum.org/profiles/UAPROF/ccppschema-20020709#">
  <rdf:Description rdf:ID="MSIE">
    <prf:component>
      <prf:HardwarePlatform rdf:ID="HardwarePlatform">
        <prf:ColorCapable>true</prf:ColorCapable>
        <prf:Keyboard>Qwerty</prf:Keyboard>
        <prf:Vendor>Microsoft</prf:Vendor>
        <prf:SoundOutputCapable>true</prf:SoundOutputCapable>
        <prf:StandardFontProportional>true</prf:StandardFontProportional>
      </prf:HardwarePlatform>
    </prf:component>
    <prf:component>
      <prf:SoftwarePlatform rdf:ID="SoftwarePlatform">
        <prf:CcppAccept-Charset>
          <rdf:Bag>
            <rdf:li>US-ASCII</rdf:li>
            <rdf:li>ISO-8859-1</rdf:li>
            <rdf:li>UTF-8</rdf:li>
            <rdf:li>ISO-10646-UCS-2</rdf:li>
          </rdf:Bag>
        </prf:CcppAccept-Charset>
      </prf:SoftwarePlatform>
    </prf:component>
    <prf:component>
      <prf:BrowserUA rdf:ID="BrowserUA">
        <prf:FramesCapable>true</prf:FramesCapable>
      </prf:BrowserUA>
    </prf:component>
  </rdf:Description>
</rdf:RDF>
```

There is a deficiency with this stylesheet approach as it cannot process one syntax used for RDF container constructs when items in a container are identified by numerical property names (_1, _2 etc). For more information about this problem see [15].

3.2 Schema conversion

Next an XSLT stylesheet is used to convert the UAProf RDF Schema into XML Schema in order to validate profiles in the constrained RDF form. This stylesheet extracts datatype information from the `rdf:type` and `rdfs:range` properties of the UAProf properties in the RDF Schema. This information is then used to perform type checking and also catch common errors such as misspelled property names. The XML schema ensures that profiles only use the properties and components defined in the schema, and that properties are located in the correct components. The schema also checks datatypes, and checks the syntax of bags and sequences. Note in order to use this approach it is necessary to fix some small errors in older versions of the UAProf RDF Schema, and also add some information as RDF inside the schema to indicate the data type. It is our expectation that these issues will be resolved in future versions of the schema. For an in-depth explanation of the revised schema, see Appendix A.

Given the following property declared in the RDF Schema:

```
<rdf:Description rdf:about="&ns-prf;ColorCapable">
  <rdf:type rdf:resource="&ns-rdf;Property"/>
  <rdfs:domain rdf:resource="&ns-prf;HardwarePlatform"/>
  <rdfs:range rdf:resource='&ns-prf;Boolean'/>
  <prf:resolutionRule rdf:resource='&ns-prf;Override'/>
</rdf:Description>
```

- The `rdf:about` attribute of the `<rdf:Description>` element gives the name of the property being described.
- The `<rdf:type>` element specifies that this is a property in RDF rather than a class.
- The `<rdfs:domain>` element specifies that this property belongs to the HardwarePlatform component.
- The `<rdfs:range>` element specifies that the datatype of this property is Boolean.
- The `<prf:resolutionRule>` element specifies the resolution rule of the property i.e. how to merge multiple occurrences of this property.

The stylesheet generates the following XML schema fragment:

```
<xsd:complexType name="HardwarePlatformType">
  <xsd:all>
```

```
    <xsd:element name="ColorCapable" minOccurs="0">
      <xsd:complexType>
        <xsd:simpleContent>
          <xsd:extension base="types:SingleBoolean"/>
        </xsd:simpleContent>
      </xsd:complexType>
    </xsd:element>
    …
    …
    …
  </xsd:all>
</xsd:complexType>
```

- The `<xsd:complexType>` called “HardwarePlatformType” contains all the properties that can occur within the HardwarePlatform component.
- The `<xsd:all>` declaration indicates that the properties can occur in any order.
- The `<xsd:element>` indicates that the property called ColorCapable is valid in this component. The `minOccurs` attribute states that the element is optional i.e. a profile does not have to contain it.
- The `<xsd:complexType>` element indicates that some properties can contain child elements.
- The `<xsd:simpleContent>` element indicates that this property only contains text.
- The `<xsd:extension>` element indicates the datatype of the property.

Note that the resolution rule information is not used here. Although it is needed by a UAProf processor it is not necessary for validation.

3.3 Analysis of stylesheet approach

The stylesheet approach to validation presented here has a number of advantages: it provides a simple mechanism for validation that makes use of existing tools e.g. XSLT and XML Schema. Furthermore using this functionality in a program is simple, since there are several open source XML Schema parsers and XSLT transformers available such as Apache Xerces and Apache Xalan[16]. It also makes use of existing information e.g. the RDF Schemas for UAProf. However, by making some small changes to this existing information we can provide much more powerful validation that enables us to easily identify common errors in UAProf profiles.

The downside of performing validation in this way is that both profiles and vocabularies must be transformed before they can be validated. Ideally it should be possible to validate profiles without any changes, as validating transformed profiles can lead to error messages being difficult to interpret as they refer to a different profile than the one presented by the user. However the transformation of profiles does have the useful effect of correcting any minor errors in RDF syntax, without the need for user intervention.

Secondly because there are various versions of the UAProf vocabulary, each using a different namespace URI, it is necessary to have separate stylesheets to convert profiles and schema belonging to the different versions. This is due to a restriction in XSLT that prevents stylesheets from inserting namespace declaration attributes into a document. Since the required change is simply the substitution of one namespace with another, a simple Java application has been written to generate stylesheets for a given namespace URI. This application and example schemas can be obtained from[17].

As mentioned earlier, it would also be useful to have a mechanism to enumerate all the valid choices for the value of a property, and thus ensure that a profile conforms to this. This could be specified in RDF schema as follows:

```
<rdf:Description rdf:about="&ns-prf;Keyboard">
  <rdf:type rdf:resource="&ns-rdfs;Property"/>
  <rdfs:domain rdf:resource="&ns-prf;HardwarePlatform"/>
  <rdfs:range rdf:resource='&ns-prf;Literal'/>
  <prf:resolutionRule rdf:resource='&ns-prf;Locked'/>
  <prf:allowableValues>
     <rdf:Bag>
       <rdf:li>Disambiguating</rdf:li>
       <rdf:li>Qwerty</rdf:li>
        <rdf:li>PhoneKeypad</rdf:li>
     </rdf:Bag>
  </prf:allowableValues>
  <rdfs:comment xml:lang="en">
      Description:  Type of keyboard supported by the device, as an
      indicator of ease of text entry.
      Type:          Literal
      Resolution:    Locked
      Examples:      "Disambiguating", "Qwerty", "PhoneKeypad"
   </rdfs:comment>
</rdf:Description>
```

The `<prf:allowableValues>` element contains a bag of all the values that are allowed for this property.

The XML schema could reflect this for single valued attributes as follows:

```
<xsd:element name="Keyboard" minOccurs="0">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:restriction base="types:SingleLiteral">
        <xsd:enumeration value=”Qwerty”/>
        <xsd:enumeration value=”Disambiguating”/>
        <xsd:enumeration value=”PhoneKeypad”/>
      </xsd:restriction>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

Here the `<xsd:restriction>` element enumerates all the possible values of the property.

Attempting to extend this idea to multi-valued properties exposes a weakness in the use of XML Schema to perform validation. Since multi-valued properties are defined in a profile using the RDF container types[*], the elements containing the property values are in the RDF namespace. As far as XML Schema is concerned, all `rdf:Bag` constructs are treated identically so the XML Schema cannot distinguish between an `rdf:Bag` associated with `prf:CcppAccept` from an `rdf:Bag` associated with `prf:HtmlVersionsSupported`. Hence it is not possible to define multiple elements with the same name (i.e. `rdf:Bag` and `rdf:li`) but different types and restrictions, since elements are differentiated by their names only. Furthermore a single XML Schema can only represent elements from a single namespace, although it may indicate that elements in a document come from other namespaces.

[*] Other RDF users have also requested that RDF Schema should provide constructs to constrain the types of members of a container. See http://www.w3.org/2000/03/rdf-tracking/#rdfs-constraining-containers

Therefore, we must consider the technique of using XML schema to be only a partial solution to the problem of validation. A more rigorous approach based on RDF needs to be developed.

3.4 Using the XML Schema validator

The basic process for using this validator is:

- Generate stylesheets for the required vocabulary URI from templates.
- Apply the schema stylesheet to the appropriate RDF schema.
- Apply the profile template to any profiles to be validated.
- Process the transformed profiles using a schema-aware validating XML parser.

For example, to validate profiles using the latest UAProf vocabulary using Xerces and Xalan:

[1] Create stylesheets:
```
java com.hp.hpl.stylesheetGenerator.Generator
http://www.wapforum.org/profiles/UAPROF/ccppschema-20020710#
```

This will create two stylesheets called `transform_uaprof_schema.xsl` and `transform_rdf_profile.xsl`.

[2] Transform the RDF schema:
```
java org.apache.xalan.xslt.Process -IN ccppschema-20020710.rdfs -XSL
transform_uaprof_schema.xsl -OUT RDF_uaprof_vocab.xsd
```

[3] Transform the device profile:
```
java org.apache.xalan.xslt.Process -IN profile.rdf -XSL transform_rdf_profile.xsl -
OUT transformed_profile.rdf
```

[4] Validate profile:
```
java dom.Writer -n -v -s transformed_profile.rdf
```

In addition to the schema created in stage 2, the schemas for RDF syntax and UAProf datatypes are required (`RDF_schema.xsd` and `UAProf_types.xsd`)

More information is available in the documentation accompanying the validation components.

4 Validating with an RDF parser

Performing validation of RDF documents using an RDF parser is more complex than validating XML documents, because there are no standardised tools available to accomplish this task. A prototype validator has been developed as an extension to DELI, making use of Jena[18], an open source RDF parser developed at HP Labs. This approach has the advantage of not requiring any transformations of profiles or schema, since Jena can parse RDF documents and RDF Schema.

To determine the structure to which profiles must adhere, the validator exploits the two level structure of UAProf profiles (profiles contain components, which contain properties). The UAProf vocabulary can be used to derive a list of valid component names, by analysing all resources having an `rdfs:subClassOf` property whose object is Component. Once this is done, it is possible to build a list of all properties that can belong to a particular component, since these will all have an `rdfs:domain` property arc to the component resource. Collection type information is determined by checking the `rdf:type` properties of device properties i.e. if a property is of type `rdf:Bag` or `rdf:Seq`, or a simple type if not declared to be otherwise.

When using this validation technique, and provided that the RDF schema has the format recommended in Appendix A, datatypes can be extracted from `rdfs:range` properties and checked by matching values against regular expressions defined in the schema for each type. The UAProf vocabulary gives regular expressions for the datatypes it defines, and these are used in the validator. It became apparent, however, that many profiles do not adhere to these specified expressions, and nor in fact do many of the examples given in the UAProf specification itself. For example, the literal datatype has the following regular expression in the schema:

```
[A-Za-z0-9/.\-_]+
```

A large number of literals in profiles contain spaces, asterisks, semicolons and various other characters forbidden by this expression. Although this problem is easily solved by extending the expression to allow a wider variety of strings, ideally these regular expressions should be machine readable rather than written as XML comments to make it easier for RDF parsers to extract them and use them in profile validation. The fact that the existing regular expressions in the specification and real world profiles do not match is a further justification for an automated validation process for profiles.

To overcome this, the datatypes could be defined in the schema as follows:

```
<rdfs:Class rdf:about="&ns-prf;Boolean">
  <rdfs:label>Boolean value</rdfs:label>
  <rdfs:subClassOf rdf:resource="&ns-rdfs;Literal"/>
  <prf:regularExpression>(Yes)|(No)</prf:regularExpression>
  <rdfs:comment xml:lang="en">
    This class is used to represent any boolean attribute value
  </rdfs:comment>
</rdfs:Class>

<rdfs:Property rdf:about=”&ns-prf;regularExpression”>
  <rdfs:label xml:lang="en">Datatype regular expression</rdfs:label>
  <rdfs:domain rdf:resource=”&ns-rdfs;Literal”/>
  <rdfs:range rdf:resource=”&ns-rdfs:Literal”/>
  <rdfs:comment xml:lang="en">
    This property defines a regular expression for a datatype
  </rdfs:comment>
</rdfs:Property>
```

A brief outline of the validation process is as follows:

```
[1]  Identify all UAProf namespaces declared in the profile
[2]  For each UAProf namespace do
[3]  Begin
[4]    Identify all component properties in the namespace
[5]    For each component do
[6]    Begin
[7]      Identify all UAProf device attributes in the component
[8]      For each device attribute do
[9]      Begin
[10]       Attempt to find a definition of the attribute in a vocabulary schema
[11]       Fail if the attribute is not defined
[12]       Check that the attribute occurs in the correct component
[13]       Check that the attribute has the correct syntax for its collection type
[14]       If the attribute is a simple type then
[15]         Check that the attribute matches its given datatype
[16]       Else
[17]         Check that all the elements in the complex type match the datatype
```

```
      Endif
    End
  End
End
```

Since UAProf vocabularies do not at present contain sets of allowable values for device attributes, this validator is unable to enforce such a restraint, however this functionality could be added in a straightforward manner, and checking for conformance could be performed at the same time as datatype validation.

4.1 Using the RDF parser validator

The RDF parser validator is part of the DELI distribution; the class providing this functionality is *com.hp.hpl.deli.UAProfValidator*. The API is documented in the DELI documentation, however a short example is given here:

```
Workspace.getInstance().configure(null, "config/deliConfig.xml");
UAProfValidator validator = new UAProfValidator(System.out);
validator.setDefaultDatatypes();

String profileName = …

if(validator.validate(profileName)) {
       System.out.println("Profile is valid\n");
} else {
       System.out.println("Profile is not valid\n");
}
```

The first line is required to instruct DELI to load the required vocabulary information. The validator object is created, specifying the `System.out` stream as the stream to write status messages to. The validator is then instructed to use the default UAProf datatypes; if required an alternative set of datatypes can be loaded from a configuration file. The format of this file is as follows:

```
<?xml version="1.0"?>
<validator>
  <datatype>
    <name>Literal</name>
    <expression>[A-Za-z0-9/.\\-_ ]+</expression>
  </datatype>
  <datatype>
    <name>Dimension</name>
    <expression>[0-9]+x[0-9]+</expression>
  </datatype>
  …
  …
</validator>
```

Further datatypes can be added by including more `<datatype>` elements containing a name and regular expression for the new type.

There is also a command line interface to the validator, which can be executed as:

```
java com.hp.hpl.deli.UAProfValidator [list of profiles to validate]
```

For more information please see the documentation in the DELI distribution.

5 Conclusion

As discussed in Section 3, validation of profiles using XML Schema has the advantage that it makes use of existing technologies, however it has some deficiencies when it comes to coping with the full complexity of a general RDF document. The RDF parser validation approach is more thorough, but it requires specialised software. Both approaches require that RDF Schema is used in a controversial way as it was not intended to be used for validation of document structure. Table 2 compares these two approaches.

Vocabulary Information	**Is information currently expressed in RDF Schemas associated with vocabularies?**	**Can this information be validated using XSLT / XML Schema?**	**Can this information be validated using a custom RDF validator?**
The set of valid property names.	Yes	Yes	Yes
The set of valid component names.	Yes	Yes	Yes
The parent components for each property.	Yes	Yes	Yes
The data type of each property i.e. literal, Boolean, positive integer, rational or custom.	Forthcoming - when RDF Core reaches a decision on datatyping. Stored in comments in UAProf.	Yes	Yes
Whether each property is single or multi-valued.	Yes	Yes	Yes
For multi-valued properties, whether those values are ordered or unordered.	Yes	Yes	Yes
In the case of UAProf, how to merge multiple values of the same property.	No Stored in comments in UAProf.	Not relevant to validation	Not relevant to validation
Where a property can take a defined set of values, a vocabulary may explicitly define the allowable set of values and explain the meaning of each value.	No	Only for simple values	Yes

Table 2 - Comparing validation approaches

A further disadvantage of validating profiles in either manner, is that at present a specific validator needs to be written for each vocabulary. This is because as Table 1 and Table 2 show, not all information relevant to a vocabulary is currently encoded in the related RDF Schema. In fact in CC/PP there is no requirement that a vocabulary provides the appropriate

RDF Schema, so an application cannot be guaranteed to be able to retrieve a particular information field from a vocabulary schema. Therefore although it is possible to write a validator for UAProf profiles, this validator will need modification to cope with other vocabularies. This is part of a more serious issue of vocabulary independence that is outside the scope of this document.

Appendix A

This Section describes some problems with previous versions of the UAProf RDF Schema.

[1] In some places in the schema there are references to "#HardwarePlatform" whereas in others they refer to "#HardwarePlatform ". The trailing space is turned to escape characters by parsers so these are considered different resources.

[2] Current work on RDF Schema has made some changes to the original specification:

“Resolution: On 2nd August 2001, the RDFCore WG resolved: Multiple domain and range constraints are permissible and will have conjunctive semantics”[19]

i.e. this section of the schema

```
<rdf:Description rdf:about="&ns-prf;defaults">
   <rdfs:type rdf:resource="&ns-rdfs;Property"/>
   <rdfs:domain rdf:resource="&ns-prf;HardwarePlatform"/>
   <rdfs:domain rdf:resource="&ns-prf;SoftwarePlatform"/>
   <rdfs:domain rdf:resource="&ns-prf;WapCharacteristics"/>
   <rdfs:domain rdf:resource="&ns-prf;BrowserUA"/>
   <rdfs:domain rdf:resource="&ns-prf;NetworkCharacteristics"/>
   <rdfs:domain rdf:resource="&ns-prf;PushCharacteristics"/>
   <rdfs:comment>
   An attribute used to identify the default capabilities.
   </rdfs:comment>
</rdf:Description>
```

which is trying to say:

"defaults are a property and can be found on HardwarePlatform, SoftwarePlatform, WapCharacteristics, BrowserUA, NetworkCharacteristics and PushCharateristics component"

actually says:

"defaults are a property and can only be found on a component that belongs to all of the following: HardwarePlatform, SoftwarePlatform, WapCharacteristics, BrowserUA, NetworkCharacteristics and PushCharacteristics."

therefore it should be changed to:

```
<rdf:Description rdf:about="&ns-prf;defaults">
  <rdfs:type rdf:resource="&ns-rdfs;Property"/>
  <rdfs:domain rdf:resource="&ns-prf;Component"/>
  <rdfs:comment>
      An attribute used to identify the default capabilities.
   </rdfs:comment>
</rdf:Description>
```

[3] In the authors' opinion, instead of having namespaces in the entire document, it is preferable to use entity declarations so the namespaces are defined once at the top. This removes the danger of using different namespaces to refer to the same object, a common mistake in some of the earlier UAProf schemas.

[4] Attributes should be qualified with a namespace e.g. `rdf:resource` and `rdf:about`. Even though this only generates warnings in the validator, using attributes without qualifying them and without a default namespace is an incorrect use of XML.

[5] It is preferable for the schema to use `rdf:about` not `rdf:ID`. This is what the forthcoming CC/PP Working Draft says on the subject:

"This specification uses "`rdf:about`" to specify the URI's of resources. This was a deliberate choice to ensure that such URI's are absolutely and unambiguously specified. This is also a different to UAProf, which uses both "`rdf:about`" and "`rdf:ID`".

CC/PP allows "`rdf:ID`" attributes or "`rdf:about`" attributes. However, the values of "`rdf:ID`" attributes represent URI's that are relative to the base URI of the document. When a document is moved to another location on the web the meaning of the value of an "`rdf:ID`" attribute changes. The meaning is undefined when the RDF is contained in a document with no base URI, e.g. when encapsulated in a message. The RDFCore WG has a Working Draft that proposes that RDF should support "`xml:base`" attributes. If this addition to RDF achieves recommendation status, then it would be appropriate to use "`rdf:ID`" attributes in conjunction with an "`xml:base`" attribute instead of "`rdf:about`" attributes. For now we recommend that CC/PP profiles SHOULD use "`rdf:about`" and that the URI's of resources are fully specified."

Therefore `rdf:ID` or `ID` should be changed to `rdf:about`, and fully qualified base URI's should be used wherever possible.

[6] In the old schemas, the data type and the resolution rule where hidden in the comments fields. This makes things very difficult for processors e.g. the DELI UAProf processor currently has to parse the comments fields to determine data type and resolution rule. It's much better to represent them in the schema e.g.

```
<rdf:Description rdf:about="&ns-prf;ColorCapable">
  <rdf:type rdf:resource="&ns-rdf;Property"/>
  <rdfs:domain rdf:resource="&ns-prf;HardwarePlatform"/>
  <rdfs:range rdf:resource="&ns-prf;Boolean"/>
  <prf:resolutionRule rdf:resource="&ns-prf;Override"/>
</rdf:Description>
```

- The `<rdfs:type>` property indicates that this is a property of a device. For multi valued device properties, this property is also used to identify that the device property is a bag or a sequence.
- The `<rdfs:domain>` property determines the parent component of the device property.
- The `<rdfs:range>` property gives the data type of the device property.
- The `<prf:resolutionRule>` property gives the resolution rule associated with the device property.

Work on datatypes is currently one of the issues being considered by the RDF Core Working Group, and a current proposal[20], compatible with the use of the rdfs:range property, also allows datatypes to be explicitly defined locally as follows:

```
<some:Property rdf:type="&datatypeURI;">&lexicalForm;</some:Property>
```

For the purposes of CC/PP vocabularies, global datatyping (i.e. using rdfs:range) is the most useful method, since each device property will have a single datatype, which will not vary between different instances of that property in a profile.

[7] To satisfy internationalization concerns, in rdfs:comment and rdfs:label language should be defined using xml:lang e.g.

```
<rdfs:label xml:lang="en">Component: SoftwarePlatform</rdfs:label>
<rdfs:comment xml:lang="en">
   The SoftwarePlatform component contains properties of the device's
   application environment, operating system, and installed software.
</rdfs:comment>
```

[8] The bag collection type should be in the RDF namespace, not the RDF schema namespace, and likewise for the Property class.

```
<rdf:Description rdf:about="&ns-prf;InputCharSet">
  <rdf:type rdf:resource="&ns-rdfs;Property"/>
  <rdf:type rdf:resource="&ns-rdfs;Bag"/>
  …
</rdf:Description>
```

Therefore the fragment above should be:

```
<rdf:Description rdf:about="&ns-prf;InputCharSet">
  <rdf:type rdf:resource="&ns-rdf;Property"/>
  <rdf:type rdf:resource="&ns-rdf;Bag"/>
  …
</rdf:Description>
```

[9] The use of `rdf:type` to identify a property as being a bag or sequence is, strictly speaking, incorrect. Rather, `rdf:range` should be used to identify that the object of a device property is such a container type. Therefore, a property defined as a bag should be described as:

```
<rdf:Description rdf:about="&ns-prf;BluetoothProfile">
  <rdf:type rdf:resource="&ns-rdf;Property"/>
  <rdf:range rdf:resource="&ns-rdf;Bag"/>
  …
</rdf:Description>
```

Expressing the property in this way, however, leaves no way to define the datatype of the elements inside the container. As mentioned in Section 3.3, expressing a constraint such as this has been ruled to be currently beyond the scope of RDF Schema. To allow the validation of multi-valued properties, we chose to leave the definition of such properties in its current form in the UAProf schemas, since the information is machine readable and the data required for validation can be extracted if the schemas are formatted in this way. This is done with a view to bringing the approach into line with the RDF Schema specification when it is finalised by the RDF Core WG.

[1] *W3C CC/PP Working Group*
http://www.w3.org/Mobile/CCPP/

[2] *W3C Device Independence Activity*
http://www.w3c.org/2001/di/

[3] *OMA / WAP Forum UAProf Specification*
http://www1.wapforum.org/tech/documents/WAP-248-UAProf-20010530-p.pdf

[4] *Resource Description Framework (RDF) Model and Syntax Specification*
W3C Recommendation 22 February 1999
Ora Lassila, Ralph R. Swick
http://www.w3.org/TR/REC-rdf-syntax/

[5] *Resource Description Framework (RDF) Schema Specification 1.0*
W3C Working Draft 30 April 2002
Dan Brickley, R. V. Guha
http://www.w3.org/TR/rdf-schema/

[6] *DELI: A Delivery context Library for CC/PP and UAProf*
HP Labs Technical Report 2001-260
Mark H. Butler
http://www-uk.hpl.hp.com/people/marbut/DeliUserGuideWEB.htm

[7] *CC/PP and UAProf: Issues, Improvements and Future Directions*
HP Labs Technical Report 2002-35
Mark H. Butler
http://www.hpl.hp.com/techreports/2002/HPL-2002-35.html

[8] *Some Questions and Answers On CC/PP and UAProf*
HP Labs Technical Report 2002-73
Mark H. Butler
http://www-uk.hpl.hp.com/people/marbut/someQuestionsOnCCPP.htm

[9] *XSL Transformations (XSLT) Version 1.0*
W3C Recommendation 16 November 1999
James Clark
http://www.w3.org/TR/xslt

[10] *W3C XML Schema Activity*
http://www.w3.org/XML/Schema

[11] *Composite Capabilities / Preference Profiles (CC/PP) Structure and Vocabularies*
W3C Working Draft 15 March 2001
Graham Klyne, Franklin Reynolds, Chris Woodrow, Hidetaka Ohto
http://www.w3.org/TR/CCPP-struct-vocab/

[12] *W3C RDF Validation service*
http://www.w3.org/RDF/Validator/

[13] *Co-Parsing of RDF and XML*
HP Labs Technical Report 2001-292
Jeremy Carroll
http://www-uk.hpl.hp.com/people/jjc/docs/r292.pdf

[14] *W3C RDF-Core Working Group Issue Tracking*
http://www.w3.org/2000/03/rdf-tracking/#rdf-ns-prefix-confusion

[15] *W3C RDF-Core Working Group Issue Tracking*
http://www.w3.org/2000/03/rdf-tracking/#rdf-containers-otherapproaches

[16] *Apache Xerces XML parser and Apache Xalan XSLT stylesheet processor*
http://xml.apache.org/

[17] DELI SourceForge Site,
http://delicon.sourceforge.net/

[18] *HPL Semantic Web activity*
http://www.hpl.hp.com/semweb/

[19] *W3C RDF-Core Working Group Issue Tracking*
http://www.w3.org/2000/03/rdf-tracking/#rdfs-domain-and-range

[20] *w3c-rdfcore-wg@w3.org from August 2002: The latest proposal*
http://lists.w3.org/Archives/Public/w3c-rdfcore-wg/2002Aug/0114.html